A Framework for Collaborative Writing with Recording and Post-Meeting Retrieval Capabilities

Matt Bouamrane† **David King**‡
†Department of Computer Science
Trinity College
Dublin, Ireland
{bouamrane,luzs}@cs.tcd.ie

Saturnino Luz† **Masood Masoodian**‡
‡Department of Computer Science
The University of Waikato
Hamilton, New Zealand
{dnk2,m.masoodian}@cs.waikato.ac.nz

ABSTRACT

From a HCI perspective, elucidating and supporting the context in which collaboration takes place is key to implementing successful collaborative systems. Synchronous collaborative writing usually takes place in contexts involving a "meeting" of some sort. Collaborative writing meetings can be face-to-face or, increasingly, remote Internet-based meetings. The latter presents software developers with the possibility of incorporating multimedia recording and information retrieval capabilities into the collaborative environment. The collaborative writing that ensues can be seen as an activity encompassing asynchronous as well as synchronous aspects. In order for revisions, information retrieval and other forms of post-meeting, asynchronous work to be effectively supported, the synchronous collaborative editor must be able to appropriately detect and record meeting metadata. This paper presents a collaborative editor that supports recording of user actions and explicit metadata production. Design and technical implications of introducing such capabilities are discussed with respect to document segmentation, consistency control, and awareness mechanisms.

CATEGORIES AND SUBJECT DESCRIPTORS

H.5.3 [**Group and Organization Interfaces**]: Computer-supported cooperative work; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems

GENERAL TERMS

Collaborative Editing, Multimedia Indexing and Retrieval

KEYWORDS

Synchronous Collaboration, Infrastructure, Usability

INTRODUCTION

The final version of a text is rarely the sole outcome of collaborative writing. Arriving at the final text is often a laborious process which involves extensive discussion, access to external data sources, and many revisions [7]. Although the *product* of these activities is certainly reflected in the final text, the *process* by which collaborators agree on a final form is seldom recorded anywhere other than in the group's collective memory. In scenarios described elsewhere [8] as *joint writing*, synchronous interaction acts as a focal point to a number of activities performed prior to the collaborative editing session. While there is widespread belief that recording audio, and to a lesser extent video, produced during such synchronous sessions provides valuable additional information sources for collaborators [5, 6], little research has been done on how the design of a synchronous collaborative editor should be adapted to allow effective integration of the editor into an environment that incorporates multimedia recording and information retrieval.

In order to address that issue we have designed and implemented *RECOLED* (REcording COLlaborative EDitor), an XML-based collaborative editing environment that incorporates detailed logging of text editing actions. Our main design assumption is that, recorded text editing sessions combined with speech recordings can provide a valuable resource for post-meeting information retrieval and therefore support the asynchronous phase of the writing process. The focus of research in collaborative editing has generally been on groupware distribution architectures, centralised, replicated or hybrid [9], or on concurrency and consistency issues [10]. This paper, on the other hand, focuses on the issue of collecting meeting metadata. In what follows we describe the idea of annotating and timestamping text segments in ongoing meetings, and discuss the requirements and design issues that arise when a collaborative editor is immersed in an environment that encompasses meeting recording and post-meeting retrieval capabilities.

BACKGROUND

The general collaborative writing scenario assumed in this paper is one in which geographically dispersed users work on a shared editor supported by an audio (speech) channel [4]. Since in the scenario we have investigated collaborators are not assigned specific *a priori* roles, the processes of *communication*, *discussion*, *argumentation*, *clarification* and *negotiation* are critical to successful collaborative writ-

ing. The audio channel presents a suitable medium for those processes. It does not require extra screen real estate, and it is the fastest and arguably the richest and most fluid medium of human-to-human communication. Increases in bandwidth, reduced prices and improved performances of hardware as well as the development of powerful sound software all contribute to making audio an ideal complement for a shared editor in collaborative writing tasks.

The ability to record, index and retrieve audio and textual contents produced during collaborative writing sessions is also regarded as a vital aspect of the scenario described above. In [2], we presented a model for multimedia information visualisation and retrieval which uses *temporal mappings* between text and speech segments as a basis for a tool that supports off-line browsing of collaboratively produced text and related audio contexts. As an example of the interrelatedness of text and audio in such recordings, consider the following collaboratively written text fragments (*t1* and *t2*) extracted from a corpus of student-supervisor meetings [3]:

> *(t1) Is the mobile visualization an improvement over a simple text based itinerary? (simple conventional paper-base)* (*clarify what is being compared!*)
>
> *(t2)* (*also find out if general user preference exist as far as a number of interface options i. do they like clocks on turning points or on the line;* [...]

The first segment (*t1*) is *temporally related* to six audio segments. In one of those audio segments (s3), the speaker makes an elliptical reference to the second text segment (*t2*), as shown in the following transcribed speech fragment:

> *(s3) ok* [*pause*] *because* [*pause*] *yes, to start with you're saying that over here* [*telepoints at (t1)*] *but as you get down to these parts* [*telepoints at (t2)*] *you are talking about text-based interfaces on the mobile phone versus the graphical form on the phone* [...]

Text segment (*t2*), by its turn, is linked to another seven audio segments, each of which might be linked to a number of text segments, and so on. The nature of these text-audio links is such that it would be nearly impossible for a user to perform meaningful post-meeting revisions without accessing both media. Annotation and timestamping of text events is therefore a vital step of content indexing in such contexts. RECOLED uses XML tags to identify and timestamp different text *segments*. Figure 1 shows an example of annotated text document. Users, actions and timestamps are identified for each text segment, assumed in this case to correspond to paragraphs.

```
<?xml version="1.0"?>
<!DOCTYPE comapdoc SYSTEM "file:comapdoc.dtd">
<?xml-stylesheet type="text/css" href="comapdoc.css"?>
<comapdoc>
<meeting date="20030304">
  <description>
Student-supervisor meeting:  User testing of a visualisation
tool ...
  </description>
  <!--participant details omitted-->
</meeting>
<section level="1">
  <segment id="1">
    <header level="1">
    <timestamp agent="A" action="insert" start="35" end="36"/>
Aim of Testing
    </header>
  </segment>
  <segment id="2">
    <timestamp agent="A" action="insert" start="35" end="36"/>
    <timestamp agent="A" action="point" start="83" end="86"/>
    <timestamp agent="A" action="point" start="100" end="104"/>
    <timestamp agent="B" action="point" start="122" end="125"/>
    <timestamp agent="B" action="point" start="146" end="148"/>
    <timestamp agent="A" action="modify" start="167" end="183"/>
    <timestamp agent="A" action="point" start="208" end="210"/>
    <timestamp agent="A" action="point" start="473" end="477"/>
Is the mobile visualization an improvement over a simple text based itinerary?
      (simple conventional paper-base)
              (clarify what is being compared!)
  </segment>
  <segment id="3">
    <timestamp agent="A" action="insert" start="35" end="36"/>
    <timestamp agent="A" action="point" start="235" end="238"/>
    <timestamp agent="B" action="modify" start="247" end="257"/>
    <timestamp agent="B" action="modify" start="268" end="281"/>
    <timestamp agent="A" action="point" start="273" end="274"/>
    <timestamp agent="A" action="point" start="406" end="411"/>
    <timestamp agent="A" action="point" start="437" end="446"/>
    <timestamp agent="A" action="point" start="461" end="465"/>
    <timestamp agent="A" action="point" start="473" end="477"/>
Hypothesis:  Visualization once understood by user allows the user to do all tasks done
by text method just as well but also allows user to make estimates and determine how
events interrelate to each other in ways that a text only interface could not.
  </segment>
.
.
.
```

Figure 1: RECOLED internal document format

DESIGN ISSUES

As mentioned above, the most distinctive feature of the shared editors currently under development is the generation of metadata and timestamps. Editing operations on shared editors are of a complex nature due to the distributed nature of the applications and the concurrent access to a shared document. Timestamping introduces further programming complexities in the editor design. However, it builds upon existing technical features of shared editors. Timestamping is a parallel task to common shared editing operations and does not interfere with the nature of theses operations. More precisely, it is a subordinate task whose purpose is to accurately describe editing operations whenever they take place.

Metadata and Timestamping Requirements

One of the key requirements to timestamping is that it should be transparent to the users, and introduce no further constraints or complexity to the editing task. In other words, all operations theoretically possible on previous shared editors should be possible with the inclusions of timestamps. In this section, we examine how this functionality fits into existing CSCW design issues [1], and how it might sometimes require additional consideration.

Low response time When designing a shared editor, documents are generally replicated. Updates are performed locally in order to achieve a low response time and then multicast to the other peers or sent to a central server for propagation to various clients. All these operations are transparent to the users. Timestamping, in the context of this paper, involves identifying those operations performed by a user that are deemed relevant and to trigger an event which automati-

cally generates a corresponding timestamp. This can be performed at the client side (replicated architecture) or at the server side (centralised or semi-centralised architecture). In either case, at current processing speeds, the time overhead of generating and managing timestamps is negligible in comparison to overall system performance constraints.

Awareness Awereness is traditionally implemented by a list of users, and by the use of telepointers and other awareness widgets. In this case, timestamping is used to retain information about gestures performed. As we will see below the time and occurrences of gestures take a very significant proportion of meeting time. RECOLED introduces a number of enhancements to existing awareness mechanisms in order to exploit this fact. These enhancements are described in detail below.

Concurrency control There are various approaches to concurrency issues when designing a shared editor. In the case of locking, some part of the document becomes only accessible to one user. In this case timestamping merely records the editing operation being currently performed by the user holding the lock and does not interfere with this process. Timestamping applied to other concurrency mechanisms has not been explored and would need further research.

Reversible operations Undo operations are of complex nature in group editing and this option has yet to be implemented in our editor. In terms of timestamping, allowing undos to happen while keeping track of these operations could in some instances offer useful clues for post-meeting processing.

Text Segmentation and Granularity

Currently we identify three significant operations for timestamping:

- *Insertion*, which describes the creation of a new text segment,
- *Modification*, which describes changes occurring in an existing segment, and
- *Gesturing* which describes the use of a telepointer or other deictic widgets used on one or more segments.

Further nuances such as *Deletion* timestamping could be introduced if needed. In our current implementation, a *Modification* action does not discriminate between the insertion of further text in a segment or deletion of some part of the segment. This is mainly because we are currently interested in measuring a level of activity on a particular paragraph rather than obtaining a completely accurate description of every editing operations.

The format of timestamps used in RECOLED, as illustrated in Figure 1, consists of attributes for `agent`, `action`, `start`, and `end`. The `start` and `end` record the duration of an action in seconds, measured from the start of the meeting.

The text unit in our editor is a *paragraph*, defined as a segment of text separated by at least two consecutive newline characters. Changes on a single line would be too fine grained for post-meeting processing, and therefore, the granularity for the detection and recording of a change has been assumed to be the paragraph. Timestamps are generated when an editing or gesturing (telepointing) operation takes place and are associated with the paragraph in which they were generated. Hence, each paragraph maintains a linked list of all timestamps describing the editing operations which occurred on a particular paragraph.

To this end, a segmentation method has been implemented on the clients. If a text is loaded prior to the meeting, the text is parsed and segmented into paragraphs. Otherwise, the insertion of text after two or more consecutive newlines triggers the creation of a new paragraph. In order to accurately detect the nature of changes so as to generate the correct timestamps, each client maintains a state of the current document with the number of all paragraphs and their content. Merging or splitting existing paragraphs introduce further difficulties in maintaining a coherent list of timestamps and are ongoing research issues.

Another issue lies in establishing timestamp time boundaries. Although detecting the start of an editing operation is relatively easy, determining the end leaves more scope for ambiguity and eventually has to be decided by a hard choice in the design of the editor. If a user writes a sentence and then stops for a few seconds to think about it and then resumes writing, does that constitute one or two modifications? In this case, should we generate one or two timestamps? A threshold of a few seconds of inactivity has to be arbitrarily chosen in order for the timestamping to operate efficiently. Eventually, this issue seems to be secondary as post-processing of timestamps is likely to merge editing operations of similar nature which are closely related in time.

IMPLEMENTATION

We have developed two alternative prototypes which employ different architectures, consistency control strategy and awareness mechanisms. These alternative implementations have enabled us to assess the effects of various techniques on the metadata collection and timestamping functionality as well as general usability issues. In what follows we discuss how certain design choices may impact on these factors, and describe the current choice of prototype in greater detail.

System Architecture and Timestamping

One of our prototypes uses a fully replicated peer-to-peer architecture. Timestamping takes place on the client side and the timestamp is multicast along with the editing operation to other peers. This requires a synchronisation algorithm among peers. The user interface comprises two windows: the *shared view* which is non editable and the *personal view*

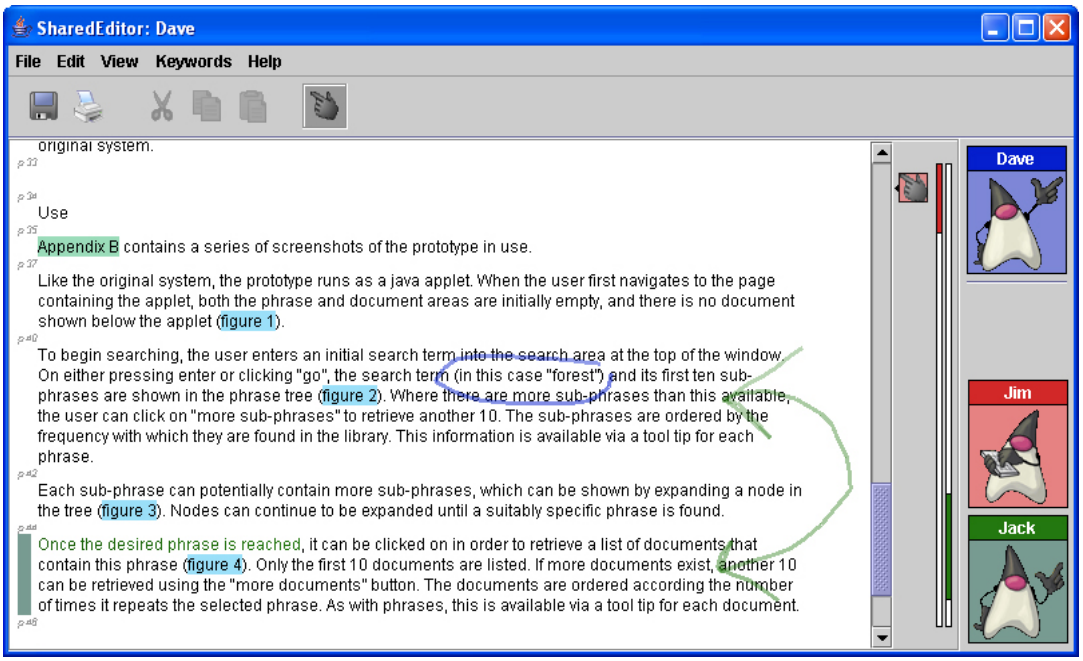

Figure 2: Synchronous collaboration in RECOLED

where the user can perform editing operations. When the user deems his operation to be finished, he can trigger the sending of the modified paragraph. In this case, the timestamp measures the time elapsed between the beginning of the editing operation and the sending of the modification. When using a telepointer, the information is sent immediately and the end of the timestamp corresponds with the release of the widget.

The second prototype was designed using a semi-centralised architecture. In this case, timestamping takes place at the server and is subordinate to an automatic, optimistic locking mechanism. When a user edits a paragraph, the server will automatically assign the client with a lock on this paragraph, unless the editing request is refused. The data unit in this case is the character which makes this implementation highly responsive as changes are propagated in real time. After a few seconds of inactivity, the server releases the lock and a timestamp is generated. This second choice introduces a small latency in the capture of timestamps. However, it offers more responsiveness and seems to be more natural to users. It also appears to minimise possible editing conflicts. We describe this second implementation in greater detail in the next section.

RECOLED

The basic architecture of RECOLED consists of a single server and several clients, communicating through TCP-IP connections. The system is semi-centralised in that clients cannot directly communicate with each other and must relay all communication through the server. However it is also semi-distributed in that each client maintains a copy of the document on which modifications can be made instantly. This ensures high local responsiveness regardless of network conditions.

To maintain consistency between the copies of the document, a paragraph based concurrency control mechanism is used. This organises the document into paragraphs, so that modifications are made on these rather than on the document as a whole. A locking scheme ensures that each paragraph can only be modified by one client at a time. This locking scheme is optimistic in order to maintain local responsiveness.

Unfortunately this locking scheme introduces two problems in terms of the user's perception of the system:

- The user will not have complete access to the document at all times.
- There will be occasions when the user's edits are not carried out. This will occur when an assumed lock is denied by the server.

These problems are quite significant, particularly because they are very difficult for the user to anticipate as they are caused by other users' interactions with the document. If left unchecked these two problems would result in very negative and frustrating use. RECOLED takes three main approaches towards minimising the effects of these issues.

Reduce the likelihood of these issues being encountered

RECOLED seeks to reduce the likelihood that users will attempt to modify the same region of text, which is the root cause of both of these problems. This is achieved by the use of group awareness mechanisms which allow users to coordinate their access to the document more effectively. Much of the interface is directed towards providing different types

Total Editing Operations	Total Pointing Operations	Editing Mean Duration (in s)	Pointing Mean Duration (in s)
268	54	4.2	8.5
64	22	5.1	6.4
128	36	3.9	5.7
84	22	7.2	6.2

Table 1: Detail of Editing Operations on 4 Separate Meetings

of group awareness.

For instance, the avatars on the right hand side of Figure 2 identify other users in the group and provide a rough indication of their current activities. They also provide a colour key as each user is assigned a unique colour which is used extensively within the interface. Shared scrollbars to the left of the avatars show where each user is located in the document and how much of the document they can see. Within the document itself, ownership of modifications is shown by shading text in the colour of the person who typed it.

Highlight these issues when they are encountered RECOLED highlights the effects of the locking mechanism by transparently showing the mechanism itself in the interface. This is achieved by using the regions to the left of each paragraph (Figure 1) which change colour to indicate the status of the paragraph lock. White indicates that the paragraph is unlocked, and the user is free to edit it. Grey indicates that a lock has been requested but has not yet been confirmed, which means that there is a chance that any modifications to this paragraph will be ignored. Any other colour indicates that the paragraph is exclusively locked by someone else who is represented by that colour. A locked paragraph cannot be edited by anyone other than the current owner. However, the lock is automatically released if the owner is not active for a specific period of time.

Allow users to communicate directly to avoid or resolve conflicts These issues become much less significant when users are given the means to communicate directly and efficiently. Because of this, RECOLED is intended to be used in conjunction with the Robust Audio Tool (RAT) [11]. With the flexibility, richness and efficiency of oral communication, it is expected that users will be able to quickly resolve conflicts by negotiating access, making requests or suggestions, or forming author/scribe relationships. RECOLED complements this by allowing users to gesture over the document, which assists in pointing out locations and illustrating concepts.

POST-MEETING PROCESSING

The idea behind capturing the time and nature of editing and gesturing operations is that we believe that this information, once processed, will highlight high level of activity and offer some precious insight into the meeting's focus. One interesting and immediate result of our timestamps collections is that it shows how important the use of gestures is in a meeting if this option is available. Table 1 shows that in a sample of four meetings from our corpus, the total number of pointing operations account for a significant number of editing operations and are generally of longer duration. In other words, gesturing takes a significant part in the collaborative writing task. Awereness widgets are used to bring other participants attention to some part of the text, and could also be used to relate different segments together. Therefore, these gestures are likely to highlight parts of text which participants focused on or which were the subject of discussions. This valuable information would be lost without the use of timestamps as the final document would contain no records of gestures.

Metadata gathered using RECOLED has been effectively used in two systems: *HANMER*, a *HANdheld Meeting BrowsER* and COMAP, a desktop-based *Content MAPper*. COMAP and HANMER employ timestamping and other forms of metadata to define temporal and contextual *neighbourhoods*, or mappings, between text and speech segments. The concept of content mapping through neighbourhoods has been shown to provide an effective way to support browsing of synchronous audio and text produced in online collaborative writing sessions. A detailed description of those systems is beyond the scope of this paper but the interested reader is referred to the relevant publications [4, 2, 3].

A different prototype is also under development which uses timestamps collected during a meeting in order to identify speech and text clusters. Trials of the prototype have shown promising results in terms of isolating high level of activity during the meeting and in some cases, identifying semantic clusters. In this case, gesturing has proved useful in highlighting links between non-contiguous text segments. The implications of this fact are a subject of ongoing research.

CONCLUSIONS

We have presented a framework for the design of shared editors with the specific purpose of recording meeting metadata, in terms of timestamping text and gesture activity. We have discussed how this functionality fits in with existing shared editor design requirements, and identified cases in which it requires special consideration. RECOLED, a collaborative editor which records meeting metadata has been described in the context of possible applications for post-meeting processing and text revision. Future work will explore how timestamping interacts with more sophisticated concurrency and consistency mechanisms, and investigate how the alter-

native timestamping and update propagation strategies discussed above affect user performance in meeting mining and information retrieval tasks.

ACKNOWLEDGMENTS

This work has been supported by Enterprise Ireland through a Basic Research Grant.

REFERENCES

1. C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

2. S. Luz and M. Masoodian. Compact visualisation of multimedia interaction records. In *Proceedings of the 7th International Conference on Information Visualisation*, pages 536–541, London, 2003. IEEE Computer Society.

3. S. Luz and M. Masoodian. A mobile system for non-linear access to time-based data. In *Proceedings of Advanced Visual Interfaces AVI'04*, pages 454–457. ACM Press, 2004.

4. M. Masoodian and S. Luz. COMAP: A content mapper for audio-mediated collaborative writing. In M. J. Smith, G. Savendy, D. Harris, and R. J. Koubek, editors, *Usability Evaluation and Interface Design*, volume 1 of *Proceedings of HCI International 2001*, pages 208–212, New Orleans, LA, USA, Aug. 2001. Lawrence Erlbaum.

5. T. P. Moran, L. Palen, S. Harrison, P. Chiu, D. Kimber, S. Minneman, W. van Melle, and P. Zellweger. "I'll get that off the audio": A case study of salvaging multimedia meeting records. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Enhancing, Finding, & Integrating Audio*, pages 202–209, 1997.

6. C. M. Neuwirth, R. Chandhok, D. Charney, P. Wojahn, and L. Kim. Distributed collaborative writing: A comparison of spoken and written modalities for reviewing and revising documents. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 2, page 202, 1994.

7. S. Noël and J.-M. Robert. Empirical study on collaborative writing: What do co-authors do, use, and like? *Computer Supported Cooperative Work*, 13(1):63–89, 2004.

8. I. R. Posner and R. M. Baecker. How people write together. In R. M. Baecker, editor, *Readings in Computer Supported Collaborative Work*, pages 239–250. Morgan Kaufmann, 1993.

9. J. Roth and C. Unger. An extensible classification model for distribution architectures of synchronous groupware. In M. Dieng, editor, *Fourth International Conference on the Design of Cooperative Systems*, pages 113–127. IOS Press, 2000.

10. 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.

11. Ucl network and multimedia research group. www-mice.cs.ucl.ac.uk/multimedia.