Design Issues and Model for a Distributed Multi-User Editor

Michael Koch
Institut für Informatik, Technische Universität München, Germany
E-mail: kochm@informatik.tu-muenchen.de

Abstract.
The collaborative editing of documents is a very common task nowadays. Writing groups are often distributed over many locations because of the globalization of organizations and the increasing interdisciplinarity of tasks. Since many writers already use computers for their jobs, providing computer support for the collaborative writing process has been identified as an important goal. Numerous tools for computer supported collaborative writing have already emerged but in most cases have not come into widespread usage. In this article the requirements of users for a collaborative editor are analyzed. Providing as much flexibility as possible to the users is identified as a basic need. According to the requirements summary a model for a group editing environment is presented. The model covers cooperative work in local and wide area networks using synchronous and asynchronous cooperation. Finally, an application of the model is presented in the form of the multi-user editing environment IRIS.

Key words: Awareness, Collaborative Editing, IRIS, Multi-User Editor, Optimistic Concurrency Control

1. Introduction

Today, the primary use of computers by far is for document processing. Most people who have to write large texts use computers to support that task. In a survey Dorner discovered that in the year 1992 74% of a large number of professional writers already used computers for writing and another 11% were considering buying one (Dorner, 1992).

When considering computer tool support for the writing process one has to take into account the fact that the size and amount of textual documents have drastically increased. This is because tasks nowadays have become more and more complex. The critical point at which some documents become too large for a single person to handle was reached long ago. As a consequence, the creation of these documents is only manageable by teams. The interdisciplinarity of tasks has also contributed to that need. Because of the globalization of organizations the members of these teams are often distributed over many locations. Economic and organizational factors are creating work groups spread over continents.

It is therefore a primary requirement for editing software to support multiple distributed users working on the same document. The rapid development of communications technology makes providing that support easier and more economical even for non-academic teams.

Most computer supported writing systems are designed for the single user. This normally means that for projects requiring multiple authors the coordination of the writing is done amongst the authors. That is acceptable if the authors are working closely together, but leads to major problems when the co-authors are dispersed and working at different times.

Nowadays, distributed teams often use the following scheme to produce a document together: Work begins with a division of work and responsibilities. Then the co-authors produce their parts of the document separately. Finally the parts are distributed for annotating and for assembling into the final document. Communication between co-authors is done by telephone, fax or e-mail. A more improved, but still insufficient way to cooperate that is widely used is networked word processing by means of a standard PC package with a central file server (Newman and Newman, 1993). Better support for the task of collaboratively writing a document is needed (Beck and Bellotti, 1993; Dillon, 1993; Neuwirth et al., 1994).

In recent few years several systems have been developed to address this problem (examples are listed in the next section). However Beck, who has studied co-authoring in academia, found no evidence of use of special collaborative writing tools, except among those groups close to the development teams (Beck and Bellotti, 1993). Instead, face to face meetings, conventional workstation and printing technology, telephone, fax and post appeared to be the means by which information was exchanged and activities were managed.

The goal of this article is to propose a model for a group editor that takes into account the way groups of writers do collaborate. Section 2 substantiates the lack of success of recent approaches. A list of requirements for group writing tools is collected and a basic architecture for a group editor is presented. In Section 3 a model for the core of the proposed editor model (the storage service) is introduced. Section 4 then describes our prototype editing environment IRIS. Finally, in Section 5 some conclusions are drawn and our plans for the future are presented.

2. Requirements for a group editor

One important class of software emerging in the area of computer supported collaborative work (CSCW) is the class of programs supporting collaborative writing. Collaborative writing is defined in (Lay and Karis, 1991) as: *'process in which authors* [e.g. editors, graphics experts, users, reviewers] *with differing expertise and responsibilities interact during the invention and revision of a common document'.* Applications supporting that task are called *group editors* (Ellis et al., 1991).

Many tools have already been proposed to support collaborative writing for different media (text, graphic, structured documents, outlines). There are tools to support synchronous editing (e.g. GROVE (Ellis and Gibbs, 1989; Ellis et al., 1990), MACE (Newman-Wolfe and Pelimuhandiram, 1991), SASE and SASSE (Baecker et al., 1993), CAVEDRAW (Lu and Mantei, 1991), GROUPDESIGN (Beaudouin-Lafon and Karsenty, 1992), GROUPDRAW (Greenberg et al., 1992)) and to support asynchronous editing (e.g. CES (Greif et al., 1986), QUILT (Fish et al., 1988;

Leland et al., 1988), PREP (Neuwirth et al., 1992; Neuwirth et al., 1990), GROUP-WRITER (Malcolm and Gaines, 1991), SHARED BOOKS (Lewis and Hodges, 1988), MESSIE (Sasse et al., 1993)).

According to Beck and Bellotti (1993) these tools (especially the synchronous ones) are not used by writing teams. Many surveys (e.g. (Tatar et al., 1991; Grudin, 1990)) identify incorrect assumptions about human cooperation that have become enshrined in the design as the main reason for this lack of usage. This means that the problem is a missing or poor understanding of the way in which groups collaborate. Beaudouin-Lafon and Karsenty (1992) state that in many applications technology has been used for the sake of technology and not to satisfy the requirements of the user.

As a consequence we must first investigate what people want to have before designing a (new) collaborative editor. This investigation should neither be restricted to academic writing groups nor to any other single group using computers to support their writing process. Rather than starting a new case study I investigated already existing reports of numerous studies on collaborative writing to get a comprehensive summary of issues and needs. These studies can roughly be divided into those suggesting procedures to be followed in order to do or to support collaborative writing (e.g. (Sharples et al., 1993)) and those presenting research results based on ethnographic and laboratory case studies of professional writing (e.g. (Beck and Bellotti, 1993; Beck, 1993; Dillon, 1993; Murray and Hewitt, 1994)). Observations of the use of prototype collaborative writing tools (e.g. (Fish et al., 1988; Malcolm and Gaines, 1991; Neuwirth et al., 1990)) that shed some light on the nature of the computer support required are also placed in the second group.

The issues that were identified can be classified into the following three major areas: stages of the document writing process; interaction and cooperation; group awareness and information. (Sharples presented a related classification in (Sharples et al., 1993).)

2.1. DOCUMENT WRITING STAGES

The document writing process has many stages. The compilation of the final text is only one of them. Other stages are: planning of document content, planning of document structure, reviewing the document. Flower and Hayes developed a model of the cognitive process involved in a writing task. This model identifies three main stages: planning, translation (translate plans to text) and reviewing (Flower and Hayes, 1981).

One might think that all these stages have to be supported by different modes in a collaborative writing tool. The problem here is, that while most writing processes proceed through a basic sequence of actions this sequence is different for different writing groups (Dillon, 1993). One group may do a detailed plan, another group just a vague plan, another one may do no planning at all. Groups or teams (these terms will be used synonymously) differ very much in their way of structuring the writing task.

It is, thus, clear that each mechanism to support the different stages will constrain users if it does not support all possibilities. No single approach is applicable in all cases. It thus seems better not to support different stages and workflow management at all. The problem here is that this solution is not satisfactory either. While some users want to have as much flexibility as possible (social protocols) other user groups want to be supported by technical protocols. Additionally the way of interaction may change over time for long lasting projects. Hence, technical support should be provided in as flexible a way as possible without being enforced.

The best and most flexible solution will be if the basic editing application does not support stages of document writing but provides interfaces to external workflow applications. These well developed tools can be used to provide the requested functionality. For the same reason one should not integrate functionality for brainstorming and group decision support into the core editor application but use existing tools and build interfaces to them. For the core editing application it is only important to provide asynchronous and synchronous text modification functionality.

Another point that can be learned by investigating the document production process is that it may last from several minutes (for a memo) to several years (for a book). Especially for the long lasting projects, a mechanism for keeping track of document versions and for restoring older versions would be of great benefit (Beaudouin-Lafon, 1990).

2.2. Interaction and cooperation

In terms of the process of text compilation nearly all reports (e.g. (Dourish and Bellotti, 1992; Irish and Trigg, 1989; Sharples and O'Malley, 1988)) mention the existence of different phases in this process. Sometimes the co-authors work very closely together and sometimes the authors work on their own. Different writing situations require integration of asynchronous and synchronous styles of work. All levels of cooperation from private editing to strict WYSIWIS ('what you see is what I see') should be available on request.

Private editing means that one author changes parts of the text but does not want the others to see his updates until the changes are finished. The updating author should be able to select when his updates will be visible to the other authors. If the updates are distributed to the applications of the other authors they should only be used to prevent conflicts and should not be displayed.

When examining global access restrictions and the control of concurrent access we often find static roles that define those operations that are allowed and those operations that are not allowed for groups of users. As Sharples states, roles clearly exist within writing groups (Sharples et al., 1993), but these roles cannot be assumed to be static. Experienced writers often wish to change roles to do some work that otherwise would be too complicated. Roles should not be used to constrain the access of authors. They may be used for information about collaborators and for selecting or filtering information. They should inform rather than constrain (Beck and Bellotti, 1993). A socially mentioned approach between (supposedly trusted) collaborators for access restrictions is far more flexible (Jones, 1993). Indeed, Ellis et al. (1991) found that users of their unconstrained multi-user editor GROVE implemented social protocols to mediate interaction. They additionally mention in their paper that support for technically enforced/supported control was provided and was used. This is similar to what has already been mentioned with document writing stages. Technical support has to be provided but not enforced.

If there are no access restrictions, we have to face access conflicts. The strategy to avoid these conflicts should be to inform about working areas of other users. If conflicts occur nevertheless, they should be recognized and presented to the users for later resolution. The users should not be forced to resolve conflicts immediately. Different parallel versions of the document should be allowed to exist temporarily (Gaines and Malcolm, 1993).

Communication among the co-authors is another very important factor. Many reports state that collaboration requires effective communication between group members to establish a shared understanding of the task (e.g. (Beaudouin-Lafon, 1990; Sharples et al., 1993)). There have to be possibilities for synchronous, asynchronous, one-to-one and one-to-many communication. Like support for document stages, this communication does not have to be provided by the group editor itself. It is possible to use external tools and standards (e.g. e-mail). Additionally there have to exist means to express the context in which a message was created and sent (Beck and Bellotti, 1993; Sharples et al., 1993). Especially for asynchronous messages a possibility should also exist to link the communication about the document to changes in the document.

2.3. Information and awareness

One general finding of recent CSCW research is that group awareness is very important to successful collaboration. This means that co-authors are aware of the actions of other authors and of everything else that concerns the common project. To establish awareness we need information about current events (synchronous information) and about the history of events (asynchronous information). Since users rely on the displayed information which, due to the software, has no guaranteed validity there has to be additional information on the 'quality' of the awareness information (see Section 3.3 for more details).

The main issue in the area of information is that the provision of information should cost as little as possible (Dourish and Bellotti, 1992). This is important because disparity between those who do the work (information providers) and those who get the benefit (information consumers) will demotivate users. Because of this the publishing of most of the information has to be done by the system automatically. There should be possibilities for the user to publish extra information or additional comments.

Synchronous information

To become aware of what the other co-authors are doing, information about the current working areas and about current updates have to be provided. It should be possible to switch to WYSIWIS processing using this information. To preserve privacy it should not be possible to use all of the information anonymously. For example, information about every keystroke should only be available if the authors agree on tight coupling.

Asynchronous information

Many reports (e.g. (Beck and Bellotti, 1993; Sohlenkamp and Chwelos, 1994)) state that a key need of people editing documents asynchronously is the ability to know what changes others have made in their absence. This history information is important for making judgments about the completeness of and confidence in parts of the document. Therefore we need data about the initiator of updates, the time an update was initiated and the place where the update occurred in the document. The history can also be extended to a version tree. With additional storage of differences between the old and the new document versions one can restore the document to any version. That might be useful for comparing the present document with former versions.

There are different ways of displaying the history. The editing history may be presented in text form or as a quick replay of the changes. The time of the last (writing) access to a part of the document may be represented by the use of color. For example in the graphic editor GroupDesign Beaudouin uses 'red' for recently changed (hot) objects and blue for static (cold) objects (Beaudouin-Lafon and Karsenty, 1992).

Another important class of information is information which is not directly related to the shared document. This includes, for example, information about other work duties of co-authors. Dillon (1993) and Beck and Bellotti (1993) mentioned that co-authors may be engaged in a range of other unrelated activities requiring their time and attention. Real group awareness cannot be achieved if information of this kind is not distributed. In this context there is once again the need for interfaces to other tools. Possible sources of information are local/shared calendar tools or group decision support systems.

2.4. Summary of requirements

If we summarize the requirements we find one important point: There must not be any constraints on the work of an author. More precisely one can say that any collaborative writing software must support the writers' normal working practices

(Sharples and O'Malley, 1988). A group writing tool should allow each individual author to plan and write in whatever style and format they feel most suited to (McAlpine and Golder, 1994). This requirement can be expressed by the following statement:

Authors working with a group editor want to have at least the facilities and the accessibility they have working with their single user editor.

First a group editor has to be usable by the single user. The authors need to:

- be able to read *and* update (write) any displayed text
  - no technical access restrictions for group members (use of social protocols instead; 'weak' locks and access modes to support social protocols)
  - the possibility of having private areas, adjustable granularity and chooseable time for making updates public
- get immediate answers to their actions (low response time)
- be able to use their favorite user interface or at least to be able to largely customize the standard interface to their needs

In addition to these requirements concerning the support of the action '*work*' there are requirements concerning support for '*cooperation*':

- history information about updates to the document (together with the differences between document versions),
- on-line information about co-authors and about their updates,
- communication among co-authors (synchronous, asynchronous, 1:1, 1:n),
- interfaces to allow integration of external tools.

To fulfill these requirements we need an integrated environment. A central component of this environment will be the *document storage*. This component has to provide a reliable storage mechanism and mechanisms for generating and distributing information for group awareness. The storage service will be used by different user interfaces to edit the document. Support for tightly coupled cooperation (video conferences, window sharing, telepointers), for direct communication (e-mail, news, ...) or for workflow management, brainstorming etc. will not be integrated into the editor user interface applications. There will instead be interfaces for calling existing external applications.

In the context of integration the following functionality is needed:

- possibility to call other applications (especially applications to communicate with other co-authors) from the editor and to transfer parameters to these applications,

– standard syntax for common identifiers (e.g. user names),

– possibility to display information provided by other applications,

– possibility to configure the way information is obtained: To show background information about another user it should be possible to contact the user's editor (if running) and receive information from that process, to execute the 'finger' command or to contact a distributed calendar tool.

In the following section I will try to outline a possible implementation in the form of an architecture for the storage service (Section 3). Then I will present our prototype group editing environment Iris as an example of an implementation of the whole model (Section 4).

3. Storage service model

In the context of providing full editing functionality (allowing every user to read and write each part of the shared document at any time) it is important to consider the handling of concurrent access and the managing of how to join and leave sessions. Then the information requirements should be tackled. This is difficult because network failures may prevent the availability of information.

For the distribution of the data across the network a number of criteria can be deduced from the editor environment issues:[1] *availability* — users should be able to gain access to data when they need it; *consistency* — users should see identical (or, at least, consistent) views of shared data even though they may be working at different places or different times; and *responsiveness* — the access process to the data should not interfere with the interactive response of the system.

3.1. Concurrency control

The first decision to be made is whether the shared document should be stored in a central server or be replicated on the workstations. The replication approach must be taken since access to document parts becomes impossible when the server is inaccessible. For the same reason partial replication cannot be used. Hence, every co-author has to get his own copy (replica) of the document.

Access to the replicated document by the user can now be addressed. Co-authors working synchronously implies concurrent access. Concurrent reading is not a problem but concurrent writing is. If two users attempt to modify the same part of the text simultaneously, especially with replicated data, the outcome will be unpredictable. That is not desirable because there should be a one-copy-view on the document.

[1] ? (?) presented a similar set of criteria in the context of the CSCW toolkit Prospero.

? (**?**) define concurrency control as the activity of coordinating the potentially interfering actions of processes that operate in parallel. On the one hand this is a technical problem. Approaches of conventional distributed systems research can be used. On the other hand, concurrency control in applications supporting cooperative work is also a human problem. Both must be considered together.

Traditional methods for concurrency control are serialization or privileged access through locking. These 'pessimistic' methods guarantee that no conflicts occur. Their main task is inconsistency avoidance. To establish that guarantee it is sometimes unavoidable to restrict access. Especially if the communication network is partitioned, access will only be possible in one of the partitions. Embedding these formal turn management policies and protocols in the interface is too restrictive. Hence, pessimistic concurrency control cannot be used in our model. We need an optimistic multiple-reader multiple-writer concurrency control that tolerates network partitions.

Optimistic methods do not bother avoiding conflicting updates. All they guarantee is to detect conflicts after they have occurred. An update is just applied to the local replica and then distributed to be applied to the other replicas. Hence low response time at the local application is possible. If a conflict between two updates occurs it has to be detected. Machinery is needed in the system for detecting conflicts, for automatic resolution when possible, and for confining damage and preserving evidence for manual repair. The group editor should mainly provide feedback to support the users in their own policies. It should notify about conflicts and support the resolution of conflicts by presenting the conflicting updates in a profitable way.

For implementing this optimistic concurrency control we have to assign a unique identifier (timestamp and site identifier for example) to every update. Then we have to maintain a history of updates together with the document replicas. These histories will be exchanged temporarily (or after an update occurred) in multicast or general epidemic propagation (Demers et al., 1988). By comparing the remote history with the local one, an instance of the editor can determine if there are outstanding updates and whether these updates conflict with other updates. This concept was first used in software repositories (e.g. RCS (Tichy, 1982) and CVS (Berliner, 1990)).

If different parallel versions of an object exist in the local storage the editor has the problem of determining which version to display. We define the branch of the version tree that was first displayed as the current branch on the local machine (see Figures 1 and 2). This is important because the user should not be irritated by an automatic version switch. Nevertheless, there has to be additional information about the existence of other parallel versions and the possibility to switch explicitly between versions must exist.

If conflicts occur they should be resolved in due time. This normally cannot be handled by the storage service. Here it will be best to present the conflict to the co-authors. But there must not be the constraint of having to resolve conflicts immediately when they are detected. It must be possible to continue work with conflicting versions.

As stated in the last section it should be possible to work on a part of the document privately. This means that updates to this part should not be presented to co-authors without the author's consent. This can be achieved by not distributing private updates. To avoid conflicting updates other authors should be informed that a newer version exists which is still private. In this way private versions can be used as a substitute for locks to show intentions on parts of the text. If another author decides to do an update even though a private version exists he/she should be allowed to do so.

In order to implement that information we have to at least distribute information that an edit is occuring and the location of the edit. Since the local access layer should be allowed to delete the local replica of a object when no application is working with it, we must also distribute the whole update at that time. That is why we decided to mark a private update with a special flag. If an update is marked as private it is distributed but not presented. Other authors see the last non-private version of the object.

Figures 1 and 2 show an example with three users during and after a network partition: The initial state of the document was 'x'. Then a network failure separated Site 3 from Sites 1 and 2. In every partition an update was carried out (leading to state 'y' in one partition and to state 'z' in the other partition). After that update User 2 changed the 'y' to 'p' as private update (Fig. 1). Then the network failure that caused the partition disappeared and the sites exchanged their information. The conflict was recognized (branch in version trees) but the display did not change (Fig. 2). The authors have to resolve the conflict by merging the different versions later.

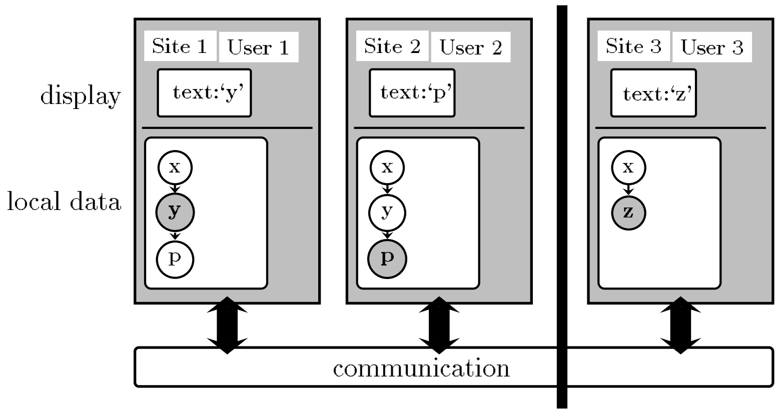

Fig. 1. Version management in the group editor - during partition

Even though conflicts do not cause the loss of data it is desirable to minimize the likelihood of conflicting updates. As past experience has shown, it is profitable

16/02/1996

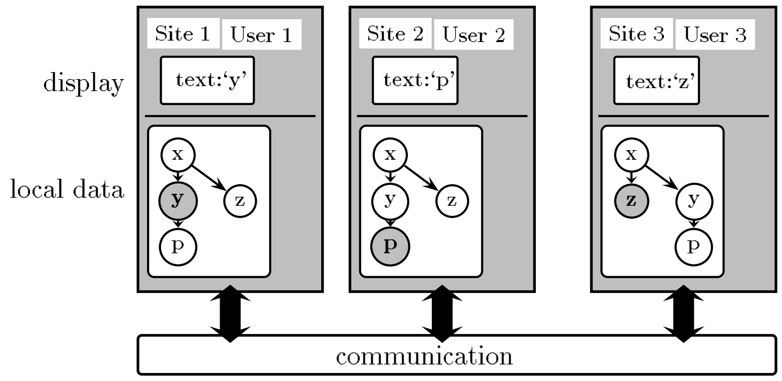

Fig. 2. Version management in the group editor - after partition

to divide the document into several parts (objects) and to update and read per object. We have to decide what we want to use as the granularity of updates. One possibility for dividing the document is to break it down based on some document model. This can once again be too restrictive for the co-authors. The best solution will be a dynamic granularity (Pacull et al., 1994). The granularity should be determined by the user. The user determined dynamic granularity can be extended by services of the user interface which automatically determine and set the granularity, based on the operations of the user. (See Section 4 for information on the solution in our prototype IRIS.)

With this framework we are able to read and write objects after having joined a session. The problem of how to present conflicts and who should be allowed to resolve conflicts will not be discussed in detail here.

3.2. JOINING AND LEAVING SESSIONS

What must still be discussed is how a computer application that a co-author starts can enter a session and how it can leave a session[2]. When leaving a session it is important for nothing which the user has changed locally to be lost. If the local workstation is separated from the others by a network failure, the local data must not be deleted. But the user should also be allowed to leave the session whenever he wishes to do so.

The system therefore must continue the editor task until it becomes clear that all the information has been transfered to another editor instance (more precisely to another storage service) or to a persistent repository. I call this concept 'passive replicas'. They are not passive in the normal meaning, but they are no longer

[2] In this context the term *session* describes the group of all co-authors currently working with an object. Joining a session means starting to work with the object, leaving a session means stopping work on the object (for some time).

changed any more by the users. The sole aim is to transfer their data to at least a single reachable interested process.

In this context we also can very easily solve the problem of joining a session: The new editor just has to try to load the text from a passive copy or from a repository (which can be seen as a special form of a passive copy). This can be done by sending a multicast to all applications which are interested in information about the document.

The requirement that a new user should be able to connect to the session even in the presence of network failures can be addressed by trying to guarantee that there is always a passive copy on a highly available host. This can be achieved by adding constraints to the transfer process of exiting editor processes (passive copies). For instance the editor process might only be allowed to transfer the data to another storage service in the local LAN-segment, rather than to any other active editor process. If the other hosts in the LAN have the same constraints and if we suppose that hosts in the same LAN are always available, the copy would be available when needed next time. Other applications of these constraints to the transfer process are currently being investigated (security, reliability issues etc.).

With persistent passive copies it is also possible to prepare a mobile workstation with local copies of document parts before the workstation is disconnected from the communication network. This technique is also mentioned as 'pre-fetching and hoarding' within the cache of the CODA distributed file system (Kistler and Satyanarayanan, 1992; Satyanarayanan et al., 1990). In CODA copies of critical files can be explicitly stored on the local workstation. When a disconnection occurs the local process logs all update activities and otherwise emulates server behavior. Upon reconnection the local updates are reintegrated by sending the log to the server for replay.

3.3. GROUP AWARENESS

The information that should be provided to enable group awareness has not yet been mentioned due to its irrelevancy to the storage service. The service should just distribute all available information without selectivity. According to Section 2 this includes information about co-authors and updates. Since information about updates is already distributed automatically by propagating the updates, we need only take care of the information about co-authors. This may be implemented by distributing a list containing the names of all local authors with every update and collecting the received lists. Detailed information about the authors may be sent on demand.

At present the display of group information is implemented in our prototype IRIS by doing just that. All available information is displayed. We plan, however, to enhance this further.

For the user interface it is important to know that not all pieces of information can be provided at any time with equal quality. The amount and quality of the information that can be provided depends on the status of the infrastructure. The application has to inform the user about the implications of the current state. For example in the case of network failures or mobile workstations one cannot determine exactly who is currently editing the document. So there should be a way to notify the user about the quality of displayed information. Possible sources of error are process failures, site failures (announced or unannounced) and network failures or the disconnection of mobile workstations. Some of these failures can be identified, others cannot. The state of the network can be viewed as an additional piece of information to be displayed.

This possible change in quality of information can be expressed (and presented to the user) in different qualities of an information service. There may be automatic degradation and restoration of the service that can be notified to the user. With this abstraction there is the additional possibility for the user to choose a specific service quality so the system does not have to try to provide more.

Here are two examples of possible degradation of service:

- display of updates
  - 'immediate' display of all local and remote updates (considering the communication delays the local version is up-to-date)
  - some co-authors are reachable and their updates will be displayed but it is not clear whether all updates in the system will be displayed immediately
  - display of own updates (maybe of some remote updates but there can be no guarantee of its completeness)
  - display of local updates only
- display of the group composition
  - reliable display of the current composition
  - hints about possible group members
  - no display

Different substages may be inserted indicating how reliable the presented information is. All information about the infrastructure can be used here. The reachability of the co-authors is an especially important piece of information which should be determined. Possible states are: all co-authors reachable, a defined sub-class of co-authors can be reached, some co-authors that were announced as being unreachable cannot be reached, some co-authors are unreachable without announcement. Announced unreachability may happen with mobile workstations.

We are currently investigating these aspects in the context of the storage service that has to provide the additional information and in the context of the user interface that has to notify the user.

4. Group editing environment IRIS

The storage service outlined in the last section must now be integrated into an editing environment. That effort has led to the development of the current architecture of our prototype group editing environment IRIS[3][4].

First we had to decide how documents should be modeled in IRIS. In this context we also decided to tackle the problem of update granularity. Since 1986 many papers have shown that documents should be defined and modeled as logical hierarchical structures (Coombs et al., 1987). A logical structure can also help to partition the document into individual objects.

A document in IRIS is a collection of objects (text, graphic, video, bibliographic references, ...) with relations between them. The relationships can be seen as a structure on the objects. This structure may be a linear list, hierarchical tree (ODA, SGML) or a hypermedia graph.

At present IRIS supports only *hierarchical structure trees*. The leaf nodes of the tree contain references to objects with the content of the document parts. Apart from navigating within the documents and assembling objects of different media types the structure is mainly used for achieving dynamic access and replication granularity. The manner in which a document has to be partitioned into content objects is not regulated. The only restriction is for data of different media types (e.g. test, graphic, video) to be stored in different objects. Hence one might partition the document per paragraph, per chapter or partly per paragraph and partly in another granularity. This partitioning can be done by the authors or automatically by the user interface applications.

According to the architecture presented in Sections 2 and 3 we separated the user interface from the underlying access layer (=storage service):

- *access layer*: This layer is responsible for managing the data objects and the structure objects of the documents. The main task of a local instance of the access layer is to communicate with remote instances of the access layer in order to synchronize access and distribute information. Not all objects are replicated on all sites. The local access layer only replicates objects currently in use by user interfaces.

- *user interface layer*: On top of the access layer, user interfaces are built. The programmer can concentrate on how to present the data that is available from the access layer and how to interact with the user. The user interface layer is a family of specialized editors rather than a single editor application. There may be different applications editing the same object synchronously.

[3] In Greek mythology, IRIS is a female messenger of the gods who reliably delivers messages from Zeus to human beings, and vice versa. This role of mediator in human-computer-human interaction together with high reliability is also demanded on our group editing environment.

[4] The IRIS project is based on former work on evaluating pessimistic concurrency control in widely distributed editing groups (Borghoff and Schlichter, 1995; Schlichter and Borghoff, 1992).

These layers consist of separate processes that communicate. Aside from the modularization gains, a big advantage of this concept is that it is very easy to integrate standard document editing applications into the system as user interfaces. There are two possibilities: If you have access to the source code of the single user editor or if the editor is highly programmable you can replace the access layer of the editor by calls to the IRIS access layer. We used that approach to implement a GNU-EMACS user interface with Elisp. If you do not have access to the source or do not want to change the standard application, you must implement a shell for the editor that receives the document or a subtree of the document from the IRIS access layer. The shell presents it to the editor as a file and transfers the data in the changed file back to the access layer after the standard editor has been terminated or whenever the temporary file was changed. We used that second approach to implement a text editor (e.g. VI, AXE) and a FRAMEMAKER user interface.

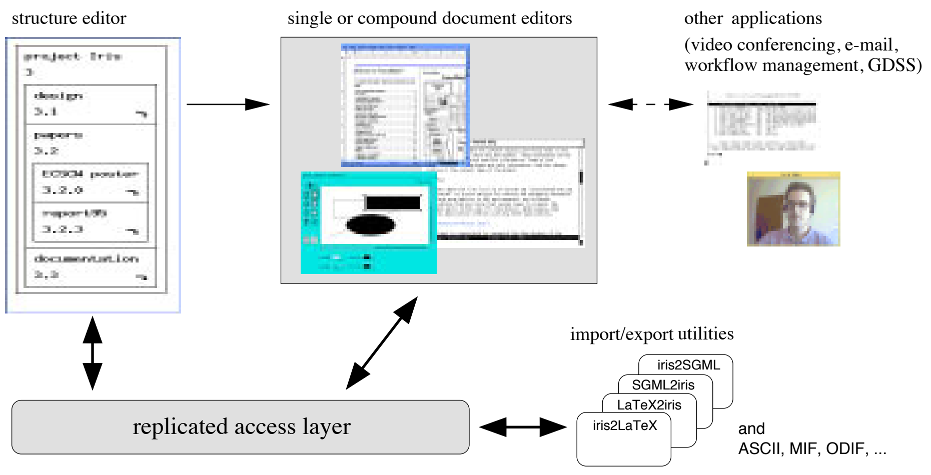

Fig. 3. User view of the IRIS environment

At present the different user interface editors (collaboration aware ones and standard editors) are loosely integrated by a structure editor which allows navigating within the structure (see Figure 3 left part). When a node is selected for editing the structure editor calls the specific editor applications. It is possible to call an editor for a whole subtree. Here the editor linearizes the content objects and allows the user to edit them as if they were a single object. These applications combine limited structure editing functionality with the ability to edit contents of different media types. A similar approach has already been used in some single user editors (e.g. QUILL (Chamberlin, 1990)). We are currently working on extending that approach to a full compound document editor (see OPENDOC standard (Adler, 1995; Nelson, 1995)).

As stated in Section 2 functionality for tightly coupled cooperation, direct communication, workflow management, brainstorming or group decision support is not integrated in the user interface applications. This functionality should be made available through integration of external applications. We are currently using a video conferencing system with IRIS to provide support for tightly coupled cooperation (direct communication, window sharing, WYSIWIS).[5] In addition to the synchronous communication, tools for asynchronous direct communication (e-mail and news) have been integrated. The integration of a workflow management tool is currently in progress.

The access layer is able to handle objects of different media types. To do that in an efficient and extensible way we have further separated it into different parts (medium-specific access specialists) (Teege and Koch, 1994). This makes it possible to handle the concurrency control or replication policy differently for different medium types. Figure 4 shows the architecture with the communication relationships between the components.

For communication between access specialists and for locating all members of an editing group we are using our own multicast service, such that messages sent by one member are received by all participants in the same network partition. This service uses hierarchical group names to allow the selection of interesting messages.

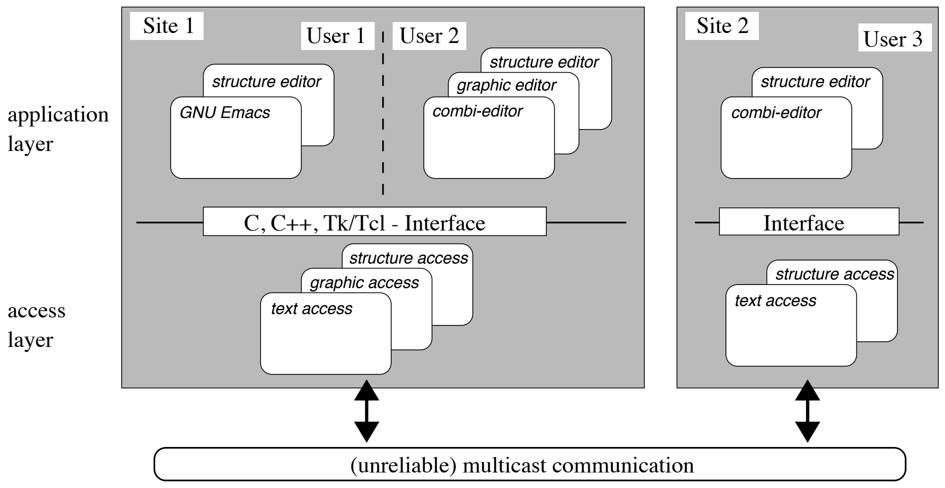

Fig. 4. Architecture of the IRIS environment

As the structure of a document is represented as a data object itself, it is possible to add other structure types to the system without changing existing applications. New or updated applications can use the extended functionality. Both application types can work collaboratively on the same shared content objects. As one example we are currently working on extending the tree structure by annotation links.

[5] This aspect of our research is funded by the Verein zur Förderung eines Deutschen Forschungsnetzes e.V. – DFN-Verein (Society for the development of a German research network).

At present we have implemented optimistic access specialists with the passive copy concept for text, graphic and for structure objects (hierarchical tree structures). Group-aware user interfaces are available for the editing of hierarchical document structure, for the editing of graphic objects and for the editing of subtrees of a document structure with the linearized text objects. The GNU-Emacs editor was enhanced to serve as a group-aware user interface. Shells were written for standard text editors and for FrameMaker. Import- and export tools are available for LaTeX and SGML files.

The system has been implemented for HP-UX 9.0x and for SOLARIS 2.4 systems. See http://www11.informatik.tu-muenchen.de/local/proj/iris/ for present information and for information on the availability of the software to the public.

5. Conclusion

Until now the need of support for collaborative editing among distributed groups has been identified but not satisfied.

In this article I have collected user requirements for a group editor and presented a new model for such an application that supports all the requirements. The proposed model has several advantages:

- It can be used in wide area networks.
- It supports disconnected workstations as well as mobile workstations with low bandwidth communication links. (The user or the application can determine when to distribute updates and when to receive updates.)
- It models as a minimum the basic semantics of single user editors.
- Existing single user editors can easily be converted to user interfaces of the system
- It supports the integration of other CSCW products and the integration of the editor itself into other systems.

Plans for the future

The model and the editing environment now need evaluation. We are completing our system by enhancing user interfaces for structure, text and graphic. Editors for other medium types and for several medium types together are planned.

The following aspects are considered as interesting and will be faced in the future:

- resolution of conflicts: how to help the user in this task (for example Intelligent ‘diffing’ (Neuwirth et al., 1992))

- user interface: how to present the information (differences between announced disconnection of mobile workstations and disconnections due to network failure); how to use the model of service degradation for information display
- user interface: multi-user compound document architecture
- access layer: evaluation of the possibilities of constraints on the transfer of passive copies
- integration of annotations
- integration of security concepts

References

Adler, R. M. (1995): Emerging Standards for Component Software. *IEEE Computer*, **28**(3), March 1995, pp. 68–77.

Baecker, R. M., Nastos, D., Posner, I. R. and Mawby, K. L. (1993): The User-Centred Iterative Design of Collaborative Writing Software. *Proceedings of ACM Conference on Human Factors in Computing Systems (INTERCHI'93)* (Amsterdam, The Netherlands), April 1993, SIGCHI, Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E. and White, T. (eds). ACM Press, New York, NY. pp. 399–405.

Beaudouin-Lafon, M. (1990): Collaborative Development of Software. *Proceedings of IFIP Conference WG 8.4 Conference on Multi-User Interfaces and Applications* (Heraklion, Greece), September 1990, Gibbs, S. and Verrijn-Stuart, A. A. (eds). North-Holland, Amsterdam. pp. 103–114.

Beaudouin-Lafon, M. and Karsenty, A. (1992): Transparency and Awareness in a Real-Time Groupware System. *Proceedings of 5th ACM Symposium on User Interface Software and Technology* (Monterey, CA), November 1992, SIGGRAPH/SIGCHI. ACM Press, New York, NY. pp. 171–180.

Beck, E. E. (1993): 6: A Survey of Experiences of Collaborative Writing. In Sharples, M. (ed.), *Computer Supported Collaborative Writing*. Springer Verlag, Berlin. pp. 87–112.

Beck, E. E. and Bellotti, V. M. E. (1993): Informed Opportunism as Strategy: Supporting Coordination in Distributed Collaborative Writing. *Proceedings of 3rd European Conference on Computer Supported Cooperative Work* (Milan, Italy), September 1993, de Michelis, G., Simone, C. and Schmidt, K. (eds). Kluwer Academic Publishers, Dordrecht. pp. 233–248.

Berliner, B. (1990): CVS II: Parallelizing Software Development. *Proceedings of USENIX Conference Winter 1990* (Washington, DC), January 1990. USENIX Association, Berkeley. pp. 341–352.

Borghoff, U. M. and Schlichter, J. H. (1995): *Rechnergestützte Gruppenarbeit — Eine Einführung in Verteilte Anwendungen*, Springer Lehrbuch. Springer Verlag, Berlin, 1995.

Chamberlin, D. D. (1990): Managing Properties in a System of Cooperating Editors. *Proceedings of International Conference on Electronic Publishing, Document Manipulation & Typography* (Gaithersburg, MD), September 1990, Furuta, R. (ed.). Cambridge University Press, Cambridge. pp. 31–46.

Coombs, J. H., Renear, A. H. and DeRose, S. J. (1987): Markup systems and the future of scholarly text processing. *Communications of the ACM*, **30**(11), November 1987, pp. 933–947.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D. and Terry, D. (1988): Epidemic Algorithms for Replicated Database Maintenance. *Operating Systems Review*, **22**(1), January 1988, pp. 8–32.

Dillon, A. (1993): 5: How Collaborative is Collaborative Writing? An Analysis of the Production of Two Technical Reports. In Sharples, M. (ed.), *Computer Supported Collaborative Writing.* Springer Verlag, Berlin. pp. 69–86.

Dorner, J. (1992): Authors and Information Technology: New Challenges in Publishing. In Sharples, M. (ed.), *Computers and Writing: Issues and Implementations.* Kluwer Academic Publishers, Dordrecht.

Dourish, P. and Bellotti, V. (1992): Awareness and Coordination in Shared Workspaces. *Proceedings of 4th International Conference on Computer Supported Cooperative Work* (Toronto, Canada), October 1992, Turner, J. and Kraut, R. E. (eds). ACM Press, New York, NY. pp. 107–114.

Ellis, C. A. and Gibbs, S. J. (1989): Concurrency Control in Groupware Systems. *Proceedings of ACM SIGMOD International Conference on Management of Data* (Portland, OR), June 1989. Published as Clifford, J., Lindsay, B. and Maier, D. (eds), *SIGMOD Record*, **18**(2). ACM Press, New York, NY. pp. 399–407.

Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1990): Design and Use of a Group Editor. *Engineering for Human Computer Interaction*, 1990, Cockton, G. (ed.). North-Holland, Amsterdam. pp. 13–25.

Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991): Groupware – Some Issues and Experiences. *Communications of the ACM*, **34**(1), January 1991, pp. 38–58.

Fish, R. S., Kraut, R. E. and Leland, M. D. P. (1988): Quilt: A Collaborative Tool for Cooperative Writing. *Proceedings of ACM SIGOIS/IEEE TC-OA Conference on Office Information Systems* (Palo Alto, CA), March 1988. Published as Allen, R. B. (ed.), *SIGOIS Bulletin*, **9**(2&3). pp. 30–37.

Flower, L. S. and Hayes, J. R. (1981): A Cognitive Process Theory of Writing. *College Composition and Communication*, **32**(4), 1981, pp. 365–387.

Gaines, B. R. and Malcolm, N. (1993): Supporting Collaboration in Digital Journal Production. *Journal of Organizational Computing*, **3**(2), 1993, pp. 195–213.

Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992): Human and technical factors of distributed group drawing tools. *Interacting with Computers*, **4**(3), 1992, pp. 364–392.

Greif, I., Seliger, R. and Weihl, W. (1986): Atomic Data Abstractions in a Distributed Collaborative Editing System (Extended Abstract). *Proceedings of 13th ACM SIGACT/SIGPLAN Annual Symposium on Principles of Programming Languages* (St. Petersburg Beach, FL), January 1986. ACM Press, New York, NY. pp. 160–172.

Grudin, J. (1990): Why Groupware Applications Fail. In Laurel, B. (ed.), *The Art of Human-Computer Interaction.* John Wiley, New York.

Irish, P. M. and Trigg, R. H. (1989): Supporting Collaboration in Hypermedia: Issues and Experiences. *The Society of Text – Hypertext, Hypermedia and the Social Construction of Information*, 1989, Barrett, E. (ed.). MIT Press, Cambridge. pp. 90–106.

Jones, S. (1993): 10: MILO: A Computer-Based Tool for (Co-)Authoring Structured Documents. In Sharples, M. (ed.), *Computer Supported Collaborative Writing.* Springer Verlag, Berlin. pp. 185–202.

Kistler, J. J. and Satyanarayanan, M. (1992): Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, **10**(1), February 1992, pp. 3–25.

Lay, M. M. and Karis, W. M. (eds) (1991): *Collaborative Writing in Industry: Investigations in Theory and Practice*, Lay, M. M. and Karis, W. M. (eds). Baywood Publishing Company, Amityville, 1991.

Leland, M. D. P., Fish, R. S. and Kraut, R. E. (1988): Collaborative Document Production Using Quilt. *Proceedings of 2nd International Conference on Computer Supported Cooperative Work* (Portland, OR), September 1988. ACM Press, New York, NY. pp. 206–215.

Lewis, B. T. and Hodges, J. D. (1988): Shared Books: Collaborative Publication Management for an Office Information System. *Proceedings of ACM SIGOIS/IEEE TC-OA Conference on Office Information Systems* (Palo Alto, CA), March 1988. Published as Allen, R. B. (ed.), *SIGOIS Bulletin*, **9**(2&3). ACM Press, New York, NY. pp. 197–204.

Lu, I. M. and Mantei, M. M. (1991): Idea Management in a Shared Drawing Tool. *Proceedings of 2nd European Conference on Computer Supported Cooperative Work* (Amsterdam, Netherlands), September 1991, Bannon, L., Robinson, M. and Schmidt, K. (eds). pp. 97–112.

Malcolm, N. and Gaines, B. R. (1991): A Minimalist Approach to the Development of a Word Processor Supporting Group Writing Activities. *Proceedings of ACM SIGOIS Conference on Organizational Computing Systems* (Atlanta, GA), 1991, SIGOIS. ACM Press, New York, NY. pp. 147–152.

McAlpine, K. and Golder, P. (1994): A New Architecture for a Collaborative Authoring System: Collaborwriter. *Computer Supported Cooperative Work (CSCW)*, **2**(3). Kluwer Academic Publishers, Dordrecht, 1994, pp. 159–174.

Murray, D. and Hewitt, B. (1994): 3: Capturing Interactions: Requirements for CSCW. In Rosenberg, D. and Hutchison, C. (eds), *Design Issues in CSCW.* Springer Verlag, Berlin. pp. 27–59.

Nelson, C. (1995): OpenDoc and its Architecture. *The X Resource*, (Issue 13). O'Reilly, Sebastopol, CA, January 1995, pp. 107–126.

Neuwirth, C. M., Chandhok, R., Kaufer, D. S., Erion, P., Morris, J. H. and Miller, D. (1992): Flexible Diff-ing in a Collaborative Writing System. *Proceedings of 4th International Conference on Computer Supported Cooperative Work* (Toronto, Canada), October 1992, Turner, J. and Kraut, R. E. (eds). ACM Press, New York, NY. pp. 147–154.

Neuwirth, C. M., Kaufer, D. S., Chandhok, R. and Morris, J. H. (1990): Issues in the Design of Computer Support for Co-authoring and Commenting. *Proceedings of International Conference on Computer Supported Cooperative Work* (Los Angeles, CA), October 1990. ACM Press, New York, NY. pp. 183–195.

Neuwirth, C. M., Kaufer, D. S., Chandhok, R. and Morris, J. H. (1994): Computer Support for Distributed Collaborative Writing: Defining Parameters of Interaction. *Proceedings of International Conference on Computer Supported Cooperative Work* (Chapel Hill, NC), October 1994, Furuta, R. and Neuwirth, C. M. (eds). ACM Press, New York, NY. pp. 145–152.

Newman, R. and Newman, J. (1993): 3: Social Writing: Premises and Practices in Computerized Contexts. In Sharples, M. (ed.), *Computer Supported Collaborative Writing.* Springer Verlag, Berlin. pp. 29–40.

Newman-Wolfe, R. E. and Pelimuhandiram, H. K. (1991): MACE: A Fine Grained Concurrent Editor. *Proceedings of ACM SIGOIS Conference on Organizational Computing Systems* (Atlanta, GA), 1991, SIGOIS. ACM Press, New York, NY. pp. 240–254.

Pacull, F., Sandoz, A. and Schiper, A. (1994): Duplex: A Distributed Collaborative Editing Environment in Large Scale. *Proceedings of International Conference on Computer Supported Cooperative Work* (Chapel Hill, NC), October 1994, Furuta, R. and Neuwirth, C. M. (eds). ACM Press, New York, NY. pp. 165–173.

Sasse, M. A., Handley, M. J. and Chuang, S. C. (1993): Support for Collaborative Authoring via Email: The MESSIE Environment. *Proceedings of 3rd European Conference on Computer Supported Cooperative Work* (Milan, Italy), September 1993, de Michelis, G., Simone, C. and Schmidt, K. (eds). Kluwer Academic Publishers, Dordrecht. pp. 249–264.

Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H. and Steere, D. C. (1990): Coda: A highly available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, **c–39**(4), April 1990, pp. 447–459.

Schlichter, J. H. and Borghoff, U. M. (1992): Concurrency Control for Multiuser Editors. *Proceedings of ACM Workshop on Tools and Technologies for CSCW (CSCW'92)* (Toronto, Canada), November 1992. Published as *SIGOIS Bulletin*, **13**(4). ACM Press, New York, NY. p. 11.

Sharples, M. and O'Malley, C. (1988): 17: A Framework for the Design of a Writer's Assistant. In Self, J. (ed.), *Artificial Intelligence and Human Learning.* London: Chapman and Hall. pp. 276–290.

Sharples, M., Goodlet, J. S., Beck, E. E., Wood, C. C., Easterbrook, S. M. and Plowman, L. (1993): 2: Research Issues in the Study of Computer Supported Collaborative Writing. In Sharples, M. (ed.), *Computer Supported Collaborative Writing.* Springer Verlag, Berlin. pp. 9–28.

Sohlenkamp, M. and Chwelos, G. (1994): Integrating Communication, Cooperation, and Awareness: The DIVA Virtual Office Environment. *Proceedings of International Conference on*

Computer Supported Cooperative Work (Chapel Hill, NC), October 1994, Furuta, R. and Neuwirth, C. M. (eds). ACM Press, New York, NY. pp. 331–343.

Tatar, D. G., Foster, G. and Bobrow, D. G. (1991): Design for Conversation: Lessons from Cognoter. *International Journal on Man-Machine Studies*, **34**(2), 1991, pp. 185–210, Republished in Greenberg, 1991.

Teege, G. and Koch, M. (1994): Integrating Access and Collaboration for Multimedia Applications. *Proceedings of International Conference on Multimedia, Hypermedia and Virtual Reality* (Moscow, Russia), September 1994, Brusilowsky, P. (ed.). pp. 170–176.

Tichy, W. F. (1982): Design, Implementation, and Evaluation of a Revision Control System. *Proceedings of 6th International Conference on Software Engineering* (Tokyo, Japan), September 1982. pp. 58–67.