THE UNIVERSITY OF AALBORG

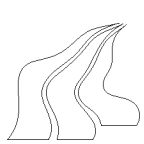
Using Events as Support for Data Sharing in Collaborative Work

by

Uffe Kock Wiil

R 91-14 April 1991

Institute for Electronic Systems
Department of Mathematics and Computer Science
Fredrik Bajers Vej 7 — DK 9220 Aalborg Ø — Denmark
Tel.: +45 98 15 85 22 — TELEX 69 790 aub dk



Using Events as Support for Data Sharing in Collaborative Work

Uffe Kock Wiil
Department of Mathematics and Computer Science
The University of Aalborg
Fredrik Bajers Vej 7E
DK-9220 Aalborg Ø, Denmark
E-mail: kock@iesd.auc.dk

Abstract

Data sharing is fundamental to computer-supported cooperative work. This paper describes how the use of events and fine grained locks, especially events, can provide powerful support for simultaneous sharing. The description is based on experience with a prototype hypertext system – Emacs HyperText System – and on observations from the literature. Three issues that comes up with simultaneous sharing: *Access contention*, *Real-time monitoring*, and *Real-time communication*, are given special interest in the paper. It is described how events and fine grained locks can remedy these issues in general and in practice – examples from the use of events and fine grained locks in EHTS are given.

This paper also appears in *Proceedings of the International Workshop on CSCW*, pages 162–176, edited by K. Gorling and C. Sattler, Berlin, April 9-11 1991.

1 Introduction

The goal of this paper is to articulate one of the major lessons learned so far from the design and use of Emacs HyperText System (EHTS) [20], a prototype hypertext system for reading, writing, and collaboration, using HyperBase [19, 21] as storage medium. The idea of using events and fine grained locks to support the collaborative situation, where several authors are working on the same document, turns out to be very powerful, especially with respect to the mechanism of sharing data.

"Hypermedia is a natural medium for supporting collaborative work. Creating annotations, maintaining multiple organizations of a single set of materials, and transferring messages between asynchronous users are the kinds of activities that form the basis of any collaborative effort. These are also activities for which hypermedia systems are ideally suited" [11]. The current generation of hypermedia does not provide support for collaboration beyond the simple mechanism of sharing data. Halasz establishes in [11] – Issue 6: Support for Collaborative work – that: "The next generation of hypermedia needs to focus equally on improvements in the technologies for shared databases and on improvements in the support for the social interactions involved in the collaborative use of a shared network. Both are necessary if hypermedia is to realize its potential as an ideal vehicle for supporting collaborative work." For an introduction to the concept and examples of hypertext systems see Conklin [8], Nielsen [14], and the special issues on hypertext in the CACM [3], ACM TOIS [4], and BYTE [6]. An introduction to different aspects of computer-supported cooperative work (CSCW) can be found in the special issues on CSCW in ACM TOOIS [1, 2].

By using events and fine grained locks in collaboration, we go beyond the simple mechanism of sharing data, and we are to some extent able to support the social interactions involved in collaboration. More specifically, it is showed how the three issues that Trigg et al. [18] talks about in connection with simultaneous sharing: *Access contention*, *Real-time monitoring*, and *Real-time communication* can be remedied by the use of the provided event and lock mechanisms.

The paper is organized in three sections. First, a description of the approach to collaboration in EHTS. Then, a description of the event and lock mechanisms. Finally, a description of the usability of events and fine grained locks in support for data sharing based on experiences from the use in EHTS and on observations from the literature.

2 The EHTS approach

This section describes our approach to collaboration in the development of EHTS.

Many programming environments are intended for a single user located at a

single machine, as is the case with the Smalltalk–80 environment [10]. In the development of EHTS [20] we aimed at a programming environment, which combines the concepts *integration*, *incrementality*, and *interactiveness* (I^3) [15] with the concepts *multi-user* and *multi-machine* (M^2). This combination is called a *multi programming environment.*

The development of EHTS has focused on two new aspects that comes up with respect to the I^3–M^2 combination. First, interaction is no longer restricted to be man-machine. When several persons are sharing the same environment, there is going to be a need for a man-man interaction through the machine. For example, how should the actions performed by one user influence on the other users in the environment? Secondly, integration of tools running on many different machines requires some rethinking of what constitutes a good architecture for the environment. This paper is only concerned with the experiences concerning the man-man interaction in EHTS.

The hypertext concept is used as a framework for development of the multi programming environment. A prototype hypertext system consisting of a multiple window text editor and a graphical browser using HyperBase [19, 21] as storage medium, was developed – see figure 1. The general idea with the two tools in EHTS is: If a user only wants to browse through (read) the text[1] of the system, he uses the browser only. If the user wants to create, modify or delete text and hyperstructure, he uses the editor also.

A major decision, concerning the man-man interaction in the system, was that, *all* actions on the shared data performed by one user should automatically be visible to all users immediately. This way users can profit by the work of other users, right after the work is carried out. How the data sharing in EHTS is supported, is shown in examples in section 4.

3 Events and locks

The description of the event and lock mechanisms in this section is based on the existing implementation of these in HyperBase [19], including the in [21] mentioned improvements.

3.1 Events

The event mechanism provides users with possibility of being notified when other users perform important actions in the database, or when certain critical events occur on data model objects (nodes or links) of interest to them.

[1] The prototype of EHTS supports text only, although HyperBase can store any kind of binary data.

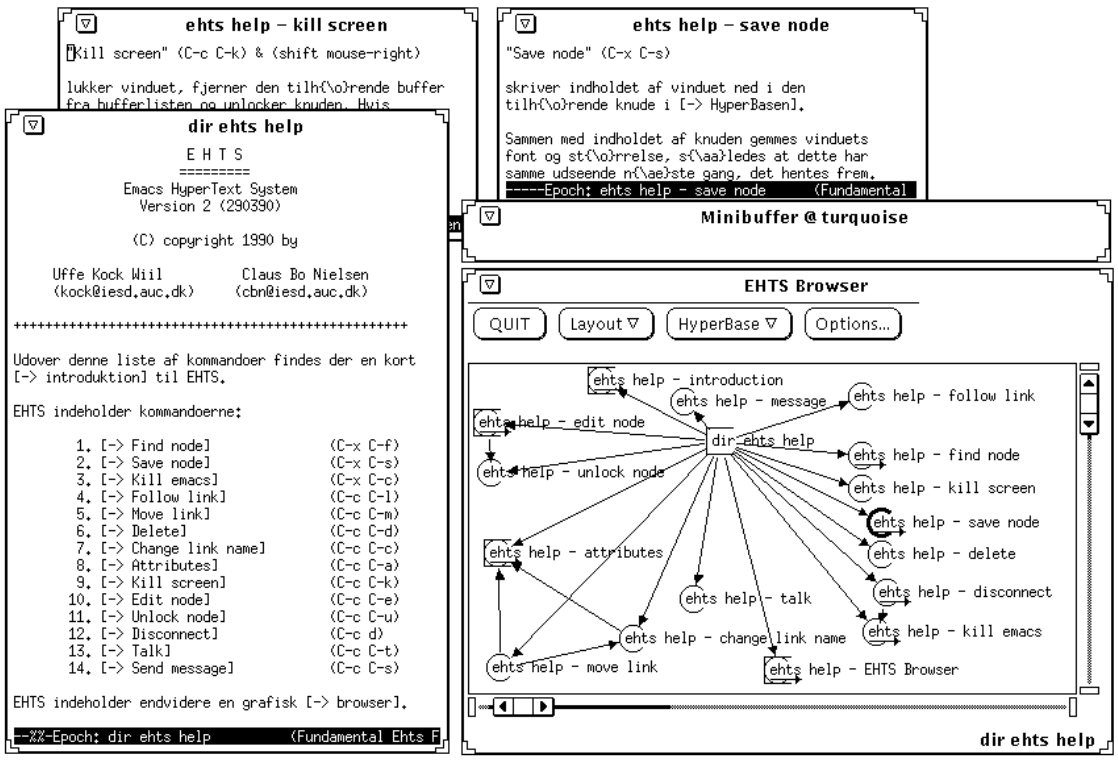


Figure 1: An illustration of the two tools in EHTS, the editor and the browser. The editor can have multiple windows opened, each containing the data attribute of a node. The minibuffer is the man-machine interaction buffer. Questions and messages from and answers to EHTS goes through this buffer. The different icons in the graphical browser represents different types of nodes.

Users of HyperBase can subscribe to any operation on any attribute in both nodes and links: `event`(object, operation, attribute). The following is a list of examples of, how the event mechanism can be used:

- **event(any,write,data):** tell me whenever the data attribute of any node is written (only nodes have a data attribute).
- **event(any,lock,any):** tell me whenever any attribute in any object is locked.
- **event(any,any,any):** event on all operations on all attributes of all objects (system logger).

Naturally, it is possible to unsubscribe to previous subscribed events, and in addition a list of all the users, that have subscribed to a specific event, can be obtained.

3.2 Locks

The locking mechanism is capable of locking (and unlocking) whole nodes and links in one operation, or single attributes in nodes and links one at a time, enabling one person to modify the data attribute of the node, other persons to modify other attributes of the node, while yet another person is annotating it by creating links. In other words, this fine grained locking mechanism enables several persons to share data and attributes within a node (or link), instead of sharing a network of nodes and links, with only one person operating on a node or a link at a time.

It is possible to read the contents of locked attributes of nodes and links, including the data attribute of nodes, and in addition readers can get the login (user) name of the writer, who has locked a specific attribute.

4 Data sharing

This section describes the possibilities, that emerges when events and fine grained locks are introduced in systems – especially hypertext systems – as support for data sharing in collaborative work. In other words, in systems where multiple users depend on simultaneous access to shared data in a common database.

Even without the use of events and locks, HyperBase provides a simple mechanism of sharing data between simultaneous accessing users:

> In order to maintain a single updated set of data accessible by multiple asynchronous users, HyperBase is a (non-distributed) single machine server, based on the client-server model, allowing only one client at a time to access the shared database [21].

Trigg et al. [18] talks about three issues, that comes up with simultaneous sharing:

1. Access contention.
2. Real-time monitoring.
3. Real-time communication.

How each of these issues can be remedied in general by the use of events and fine grained locks will be showed in the following sections, and examples from the use of events and fine grained locks in EHTS will be given.

4.1 Access contention

As mentioned, in the description of the lock mechanism (section 3.2), different users can operate on different attributes of nodes at the same time, by the use of fine grained locks, while other users annotate the node by creating links. Thus, the

fine grained lock mechanism changes the nature of access contention, moving the contention from the level of whole nodes (and links) to the level of single attributes in nodes (and links).

Situations with conflicts on modifying shared data (access/lock collisions) cannot be avoided, but, by moving the contention to a finer grained level, these situations will occur less frequently. Most likely contentions will occur in connection with data attributes, as this is the mostly accessed attribute. The event mechanism can help collaborators to solve conflicts on modifying shared data:

- Notification when a user signals an intention to modify an attribute in a node or link of interest (by obtaining a lock). Thereby other users are told, that they cannot modify the same attribute also. They can obtain the name of the user locking the attribute, enabling them to contact him and discuss the changes to make, for instance by using an internal talk mechanism (see section 4.3). By internal, we mean inside the hypertext environment.

- Likewise, notification when the user is finished modifying the attribute of interest, and therefore is unlocking it. Thereby interested users can be notified, exactly when they are able to modify the attribute, enabling them to do something else in the meantime.

How the event and lock mechanisms helps to deal with access contentions in EHTS is explained in the following section.

EHTS – Access contention

EHTS supports the possibility for different persons to modify different attributes at the same time. The data attribute of a node can be opened and modified in a text editor window on the screen. Likewise, the other attributes of a node can be opened in a common attribute window on the screen. The attributes in the attribute window can be modified independently of each other and the data attribute by different persons. An example of an attribute window is shown in figure 2.

The prototype does not support the possibility for one person to modify the data attribute of a node, while other persons are annotating it by creating links. This has to do with the way links are designed and used in EHTS. Links are shown as a text string: $[->linkname]$ in the data attribute. Therefore, users have to be in control of the data attribute to create outgoing links, although HyperBase places no such limitations. In HyperBase the outgoing links of a node are saved independently of all other attributes, including the data attribute.

To summarize, one person can modify the data attribute of a node and/or annotate it, while other persons modifies other attributes at the same time.

Three different features are provided in EHTS to help collaborators to solve conflicts on modifying shared data.

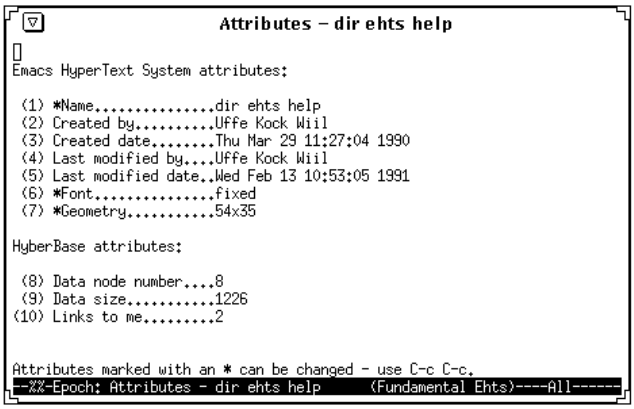
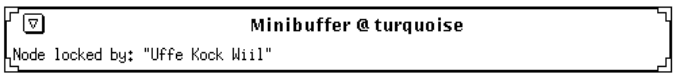


Figure 2: Illustration of an attribute window in the text editor. Attributes marked with an asterisk (*) can be modified by different persons independently of the data attribute of the node.

First, when a data attribute of a node gets locked, both hypertext tools are notified. In the browser, node icons are shown **bold faced**, when the data attribute of the node is locked – the data attribute of node "`ehts help - save node`" in figure 1 is locked. In the editor, readers having the data attribute of the node opened in a text editor window on the screen gets notified: (beep)

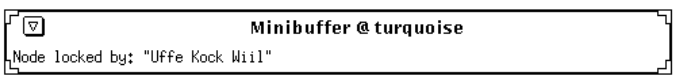
enabling them to contact the writer, for instance by the use of the internal talk mechanism in EHTS, and discuss the changes to make. This mechanism is described in section 4.3.

Secondly, a reader can open a locked data attribute of a node in a text editor window on the screen, but he gets notified: (beep)

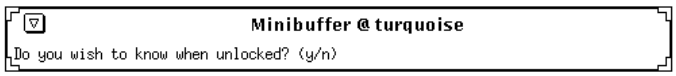


If the reader keeps the node open he can spy on the changes made by the writer, see section 4.2 Real-time monitoring.

Finally, when a user wants to modify the data attribute of a node, he tries to lock it. Either, he obtains a lock on the data attribute and is able to modify the contents, or he gets notified: (beep)

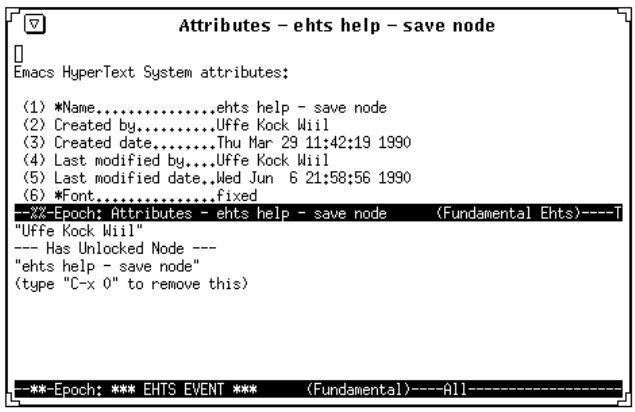


Shortly after, he is asked:



enabling him to edit the contents.

Then later, when the data attribute in question is unlocked, the user gets notified in his current working window (if his answer was *yes*): (beep)



To summarize, an unfortunate user (a victim of collision) can do one of three things:

1. He can contact the writer and discuss the changes to make.
2. He can subscribe to the event on when the writer is finished modifying and wait for this to happen. While he is waiting, he can keep the window opened and follow the changes made by the writer.
3. He can close the window and try to modify the data attribute of the node later.

4.2 Real-time monitoring

Each collaborator usually does a substantial amount of work before passing his work to the other collaborators [18]. During this period, the other collaborators cannot review the ongoing work, and hence cannot affect its real-time progress.

Events enables users to maintain a fully updated local representation of the shared hyperdocument. This is what Trigg et al. [18] calls real-time monitoring. Real-time monitoring is for instance useful in a graphical browser tool, and can be obtained by subscribing to changes in the state of the hyperstructure, like the use of the operations `createnode`, `createlink`, `lock`, `unlock`, `movelink`, `delete`, etc. When the tool is notified about changes in the hyperdocument, an internal event handler can automatically read the changes in the shared database and update the local representation of the hyperdocument.

The implementation of the tools in EHTS can exemplify how real-time monitoring can be supported in practice.

EHTS – Real-time monitoring

As mentioned, both tools use the event mechanism to maintain a fully updated local representation of the parts of the underlying shared database, that are relevant to the tool. Event handlers inside the tools takes care of the automatic updating.

Editor

The internal event handler of the editor automatically update any visible change to nodes opened in editor and attribute windows. Visible changes are:

- renaming nodes,
- resizing node windows[2],
- changing fonts in windows[2],
- changing the contents of the data attribute.

Naturally, changes are not visible to other users, before they are saved in the HyperBase.

The first three categories of changes are saved and therefore visible to other users immediately after the change. Concerning the latter category, at least three different monitoring approaches could have been used: saving after every keystroke, saving after a certain time interval, or saving on the users command. We have chosen the latter approach, that is leaving it up to the user to decide when to save changes to data attributes in nodes, although it goes against the overall design decision, mentioned in section 2, saying that all actions on shared data performed by one person should immediately after be visible to all users. The main reason being our concern for system performance[3], but also to allow moments of privacy, for instance while a participant works out the spelling of a word or a turning of phrase. The consequence is an unknown (user dependent) delay in update time for data attribute windows.

When a node is deleted, displays containing the node disappears immediately after from the screens of all readers. This has to do with the above mentioned overall design decision for EHTS.

[2]The size of and the font in node windows are saved in attributes of nodes, so that the windows have the same look next time they are opened.

[3]Small grain-size transmission of data allows small grain-size collaboration but is computationally expensive.

Browser

The graphical browser of EHTS keeps a picture of the hyperdocument, that is, how nodes are connected by links. It also shows, which nodes have the data attribute locked (the icon is bold faced).

When changes are made on the hyperstructure, and when data attributes of nodes get locked or unlocked, the browser gets notified by the event mechanism, the internal event handler reads the change, and updates the picture. Changes on the hyperstructure can be:

- creating new nodes,
- creating links between nodes,
- deleting nodes,
- deleting links,
- moving the destination of a link from one node to another.

Furthermore, is changes in node names updated on the browser picture.

One aspect of real-time monitoring is WYSIWIS (What You See Is What I See) [16, 17]. WYSIWIS is a foundational abstraction for multi-user interfaces that expresses many of the characteristics of a chalkboard in face-to-face meetings. In its strictest interpretation, it means that everyone sees exactly the same image of the meeting information and also can see where everyone else is pointing. The term was defined in connection with the Colab at Xerox PARC, which is an experimental meeting room that provides computational support for collaboration in face-to-face meetings. The Colab is equipped with a large touch-sensitive screen, a stand-up keyboard, and a number of workstations connected over a local-area network which supports a distributed database. The Colab uses a relaxed version of WYSIWIS allowing the participants to have private windows, to let public windows appear at different places on different screens, and to make pointers visible only on request. Relaxations to WYSIWIS is categorized as constraints on space, time, population, and congruence. Therefore, the relaxed version of WYSIWIS used in the Colab and the multi-user interface provided in EHTS have some properties in common, like no specific display space for windows on the screen, a delay in update time for window contents, and no strict requirements for a congruent view across screens.

4.3 Real-time communication

Communication between collaborators usually occur across time and space with consequent loss of mutual context, timeliness, etc [18]. Simultaneous information

space sharing allows for the possibility of remote real-time communication between collaborators, thereby changing the nature of their communications.

Events and locks can be used to built internal talk and message mechanisms, as can be seen in the descriptions of these in EHTS.

EHTS – Talk mechanism

Every user has his own *talk node* in connection with his login (root) node. He has automatically subscribed to the operations `lock` and `write` on the data attribute of his talk node. The following is an example of a talk session between two users, **A** and **B**:

> **Example:** Talk session
>
> User **A** wants to talk to user **B** and uses the `talk` function. The talk function locks **B**'s talk node and opens the talk nodes of both users in text editor windows on **A**'s screen. **B** gets notified about the fact, that his talk node has been locked – in other words, that somebody wants to talk to him. The internal event handler in **B**'s text editor locks **A**'s talk node and opens both talk nodes in windows on **B**'s screen. **A** writes "Hey **B**, this is **A** – I wondered if you would do me a favor...?" in **B**'s talk node, and saves the contents of the node, which causes, that **B**'s local version of his talk node automatically gets updated. **B** reads the message, writes an answer in **A**'s talk node, and saves the contents. **A** reads the answer from **B**, and the talk goes on in a similar way. When the talk is over, both users close the talk windows, and both nodes automatically gets unlocked and ready for future talk sessions.

In the internal talk sessions the users must remember to save the contents of the opponents talk node, when he is finished writing his message. Otherwise, the other part will not get the message! This is in contrast to the talk mechanism in for instance UNIX^{TM}, where you can see every character, that the opponent writes, immediately. EHTS could update talk windows after every keystroke. It is due to performance and privacy reasons, we chose to let the user decide when to save changes to data attributes in nodes (see section 4.2).

EHTS – Message mechanism

An internal real-time message mechanism can also be based on the use of locks and events, because messages can be perceived as a specialization of talk, where the users send a message and do not expect an answer immediately, if ever. Seen from that point of view, the principles of the talk mechanism can be used for the message mechanism also, which is the case in EHTS.

User-to-user messages must be written in the message node of the destination user, causing the message node to pop-up on the screen of the destination user.

Likewise, global messages must be put in the global message node of the system, which causes that all users gets notified.

A non real-time message mechanism (a mail mechanism) should also include a mechanism for keeping more than one message (mail), for instance in a list of mail nodes. Such a mechanism is not incorporated in EHTS yet.

The use of a set of semistructured message templates can be helpful in designing computer-based communication and coordination systems [12, 13]. Semistructured messages can help provide automatic aids for (1) composing messages to be sent, (2) selecting, sorting, and prioritizing messages that are received, (3) responding automatically to some messages, and (4) suggesting likely responses to other messages.

5 Other aspects of data sharing

Delisle and Schwartz [9] have introduced an abstraction to the simple node and link based hypertext data model, called contexts, geared towards multiple users. Contexts, as implemented in HAM [7], provides a way for teams of authors to work together in independent hypertext partitions without the risk of interfering with each other. The independent partitions can to be joined (merged) at carefully controlled intervals.

Akscyn et al. [5] use versioning to avoid loss of data in data sharing. Data sharing between multiple users in KMS takes form of an *optimistic concurrency control*, based on the assumption that conflicts are rare in a system with relatively few users and a great number of frames (nodes). There is no locking mechanism in KMS like in the earlier version ZOG. When a conflict arises a new version of the frame is created, to avoid destruction of changes to the frame made by other users.

6 Related Work

The Xanadu/Server^{TM} [22] is a hypermedia engine – a tool for storage, manipulation, organization, and retrieval of information, being developed at present by Xanadu Operating Company. The Xanadu/Server is designed to operate in a distributed, multi-user environment. Therefore, the ability to monitor the dynamic state of shared information and collaborate with other users has been a high priority. Xanadu *Document Sensors* offer a way of monitoring and interacting with the information base and other users. Sensors can be attached to various documents (or specific parts of documents) and watch for specific events such as the attachment of a link, or the modification of the document. When these events occur, they send a signal to the application.

7 Summary

That events and fine grained locks provide powerful support for data sharing has been shown through examples of the use in EHTS.

To sum up, EHTS provides contention resolution at the level of single attributes in nodes and links. EHTS allows any number of users simultaneously to read and display the data attribute of a given node. However, permissions to make modifications to the data attribute is restricted to one user at a time. All readers of the node (data attribute) are notified when certain modification events occur. Following Stefik et al. [17] readers are notified as soon as possible that a data attribute they are accessing may be changed. Readers are provided with four levels of modification notices:

- when one person signals his intention to modify the data attribute of the node by obtaining a lock, all readers are notified that someone has declared an intention to modify the data attribute of the node. The readers also get the name of the person, enabling them to contact him by the use of an internal talk mechanism (real-time communication). Readers can then subscribe to the event on when the writer is finished modifying and unlocks the data attribute.

- when the writer actually writes the modified data attribute of the node onto the shared database, all readers of the data attribute automatically gets the contents in the data attribute display updated with the modifications made by the writer (real-time monitoring).

- when the writer is finished modifying the data attribute of a node, users, having subscribed to this event, gets notified that the data attribute of the node has been unlocked and therefore is write accessible.

- when a node is deleted, the display of the node is removed from the screens of all readers of the node.

Thus, EHTS provides some help in dealing with access collisions at the attribute level in nodes: notification of the name of the modifying person (the writer), which is helpful in connection with the provided real-time communication mechanism. Real-time monitoring enables the unfortunate persons to follow (spy on) the writers changes. Notification when the writer unlocks the data attribute, is also a useful feature. Help beyond this indirect access collision help is not provided; locks (exclusive write permission) on attributes are allocated in an unlimited period of time on a first-come first-served basis.

8 Conclusion

The introduced event and lock mechanisms facilitate the distribution of relevant information among collaborators, and allow collaborators to follow the overall progress and to safely edit and change the hyperstructure. The three issues of simultaneous sharing: access contention, real-time monitoring, and real-time communication, mentioned by Trigg et al. [18], can be remedied by the use of events and fine grained locks. Thus, the provided support goes beyond the simple mechanism of sharing data, and meets the challenges of support for the social interactions involved in collaboration, as they are stated by Halasz [11].

The use of events and fine grained locks, especially events, in collaboration is not fully investigated yet. In the future we may uncover a lot of problems, which can be solved by the use of events, as they are known from the current implementation in HyperBase [21] or in a future version.

Ongoing Work

At present EHTS is used in a series of lectures on hypertext given by the author at The University of Aalborg to a class of masters students. In connection with the lectures, the students collaborate on a design exercise in groups of 2 to 4 participants. The design documentation is written in EHTS, which gives us a lot of experiences with the use of EHTS in collaborative situations.

Acknowledgments

Comments from Bent Bruun Kristensen and Kasper Østerbye on earlier versions of this paper have improved the presentation.

References

[1] Special Issue on Computer-Supported Cooperative Work. *ACM Transactions on Office Information Systems*, 5(2), Apr. 1987.

[2] Selected Papers from the Conference on Computer-Supported Cooperative Work (CSCW '88). *ACM Transactions on Office Information Systems*, 6(4), Oct. 1988.

[3] Special Issue on Hypertext. *Communications of the ACM*, 31(7):816–895, July 1988.

[4] Special Issue on Hypertext. *ACM Transactions on Information Systems*, 7(1), Jan. 1989.

[5] R. M. Akscyn, D. L. McCracken, and E. A. Yoder. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM*, 31(7):820–835, July 1988.

[6] Special Issue on Hypertext. *BYTE*, 13(10):234–269, Oct. 1988.

[7] B. Campbell and J. M. Goodman. HAM: A General Purpose Hypertext Abstract Machine. *Communications of the ACM*, 31(7):856–861, July 1988.

[8] J. Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41, Sept. 1987.

[9] N. M. Delisle and M. D. Schwartz. Contexts – A Partitioning Concept for Hypertext. *ACM Transactions on Office Information Systems*, 5(2):168–186, Apr. 1987.

[10] A. Goldberg and D. Robson. *Smalltalk–80, The Language*. Addison–Wesley, 1989.

[11] F. G. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[12] K.-Y. Lai, T. M. Malone, and K.-C. Yu. Object Lens: A 'Spreadsheet' for Cooperative Work. *ACM Transactions on Office Information Systems*, 6(4):332–353, Oct. 1988.

[13] T. W. Malone, K.-Y. L. Kenneth R. Grant, R. Rao, and D. Rosenblitt. Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination. *ACM Transactions on Office Information Systems*, 5(2):115–131, Apr. 1987.

[14] J. Nielsen. *Hypertext & Hypermedia*. Academic Press Inc., 1990.

[15] K. Nørmark. Programming Environments – Concepts, Architectures and Tools. Technical Report R-89-05, The University of Aalborg, Apr. 1989.

[16] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Transactions on Office Information Systems*, 5(2):147–167, Apr. 1987.

[17] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. *Communications of the ACM*, 30(1):32–47, Jan. 1987.

[18] R. H. Trigg, L. A. Suchman, and F. G. Halasz. Supporting Collaboration in NoteCards. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'86)*, pages 147–153, Austin, Texas, December 3–5 1986.

[19] U. K. Wiil et al. Design and Implementation of a HyperBase. Internal Report IR 90-03, The University of Aalborg, Sept. 1990.

[20] U. K. Wiil and C. B. Nielsen. Integration of Tools in a Hypertext Environment (in Danish). Internal Report IR 90-02, The University of Aalborg, Sept. 1990.

[21] U. K. Wiil and K. Østerbye. Experiences with HyperBase – a multi-user back-end for hypertext applications with emphasis on collaboration support. Technical Report R 90-38, The University of Aalborg, Oct. 1990.

[22] Xanadu Operating Company, Autodesk, Inc. *Xanadu/Server – Network-Based Software for Hypermedia Information Management*, Sept. 1990.