THE UNIVERSITY OF AALBORG

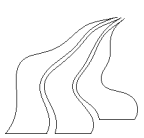
Issues in the Design of EHTS: A Multiuser Hypertext System for Collaboration

by

Uffe Kock Wiil

R 91-24 June 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



Issues in the Design of EHTS: A Multiuser Hypertext System for Collaboration

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

phone: + 45 98 15 42 11 – tone – 50 51
fax: + 45 98 15 81 29

Abstract

This paper describes the architecture, issues from the design, and experiences from the use of EHTS (Emacs HyperText System), an experimental multiuser hypertext system for collaboration. EHTS consists of a text editor, a graphical browser, and an active hypertext backend named HyperBase.
HyperBase is built on the client-server model and has been designed especially to support collaboration among its users, by providing an event (notification) mechanism and a fine-grained lock mechanism. HyperBase is still experimental: The present version only provides a simple node and link based data model and very limited query facilities.
We distribute part of the information in HyperBase at application level by maintaining caches in the editor and browser. Events from HyperBase are used to notify event handlers in the two hypertext tools about important actions on the shared data in HyperBase, enabling them to monitor changes.
Four categories of issues from the design and experiences from the use of EHTS are reported: architecture, collaboration, user interface, and data model.
Based on our experiences with the client-server model, we suggest a new and improved architecture for the event driven multiuser hypertext system.
One major lesson learned is that events and fine-grained locks can provide powerful support for data sharing among multiple users simultaneously working in the same environment.
The need for a flexible data model is another lesson learned. It is difficult to predict what data model objects actually will be needed, when the data model is designed before the user interface, as is the case in this experiment. The data model of HyperBase forces us to make some less successful decisions in the design of the graphical browser.

1 Introduction

The EHTS experiment [22] is a preliminary study in the "Hyperstructure Programming Environment" project [23] at our lab. One of the problems with most programming environments is that they are intended for a single user located at a single machine, as is the case with the Smalltalk–80 environment [9] and the Interlisp environment [18].

This is in contrast to how things happen in the real world, where programs often are developed by a large team of programmers, who could be located at each there workstation connected through a local area network. In the development of EHTS we aimed at a programming environment, which combines the concepts *integration*, *incrementality*, and *interactiveness* [15] with the concepts *multiuser* and *multimachine*.

The hypertext concept (see Conklin [5] and Nielsen [14] for an introduction) is used as a framework for development of the multiuser programming environment. First, we designed and implemented a hypertext database, HyperBase [21]. Then, we developed a multiuser multimachine hypertext system (EHTS) consisting of two tools: a multiple window text editor and a graphical browser using HyperBase as storage medium.

The focus in the EHTS experiment is on multiuser and collaboration issues in and the architecture of hypertext systems. We are less concerned with other important issues of hypertext systems, like data model and user interface issues.

The goal of this paper is to articulate the major lessons learned so far from the design and use of EHTS. We start with an overview of EHTS in section 2, before we discuss different issues in the design and report our experiences with the use of EHTS in section 3. The discussion of design issues are divided into four: architecture issues, collaboration issues, user interface issues, and data model issues.

2 Overview of EHTS

In this section we present the current stage of development of EHTS. The frontend of EHTS is developed using prototyping. We have been through many design–experiment iterations before reaching a stage in the development where we find the system ready to use in different experiments in our lab. At this stage the system is still experimental and not a finished product.

EHTS is like most other hypertext systems designed as a layered architecture (figure 3). The bottom level is the hypertext database server named HyperBase. Additional layers of functionality are built on top of HyperBase. In EHTS, one application layer is built on top of HyperBase and a user interface layer is built on top of the application layer.

The overview of the system is divided into two parts: a description of HyperBase and a description of the frontend layers. We will describe the current status of the system and save reflections on and experiences with the system for the next section on Design Issues.

2.1 HyperBase

The description is based on the existing implementation of HyperBase [21], including the in [24] mentioned improvements.

The intentions with the design of HyperBase were to construct a general tool suitable as backend for different hypermedia applications. HyperBase should capture the general idea of hypermedia, provide operations on the basic entities (nodes and links) and be easy to extend and adapt to a certain field.

2.1.1 HyperBase Architecture

HyperBase is designed as a stratified system

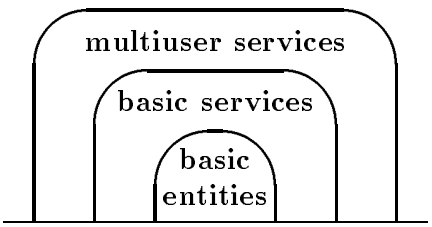


Figure 1: *The different layers of HyperBase.*

with the following layers:

- basic entities – nodes and links (file system storing and retrieval).
- basic services – operations for building graphs, read and write commands etc.
- multiuser services – locking and event handling.

The two basic layers of HyperBase could operate alone with just one user. The multiuser part provides all the facilities necessary for the multiuser environment.

2.1.2 Basic Entities

This layer provides the interface to the underlying file system. The interface is a simple data model of nodes and links:

- Links are separate objects and can only refer to nodes.
- Links are unidirectional and many-to-one, that is, links can only refer to one node, but several nodes can be the source for one link object.
- Nodes can be the source for many links.
- Both nodes and links have a set of *predefined* key/value pairs associated with them.

For links, the predefined keys are the destination node and various system information.

For nodes, the interesting keys are a list of outgoing links and the *data field*. Unlike the other keys, the outgoing links and the data field are of variable size. HyperBase does not place any restrictions on the data field, it is solely up to the applications to find out how to use this field. The reason for this is that we wanted HyperBase to be able to store all kinds of binary data, and therefore we can not predict what kind of information actually will be stored. This is also the reason why we do not have any notion of *anchors*: we do not know what kind of information is stored, and the implementation of anchors depends on the type of the stored information.

Before compiling the HyperBase server the user can specify names of additional keys and the *fixed* maximum size of their values. These keys will be included in all nodes (or links). Examples of user defined keys could be: *owner*, *date*, *version*, *type*, *name* etc. i.e. static keys of which the application programmer can predict the use. Run-time allocation of key/value pairs is not supported.

2.1.3 Basic Services

The basic services provided by HyperBase can be divided into two categories of operations:

1. Creating and maintaining structures.
2. Reading and writing.

HyperBase also provides a function capable of returning a list of all existing nodes or links in the hyperdocument. See appendix A for a complete list of HyperBase operations.

2.1.4 Multiuser Services

HyperBase has been designed with special support for collaborative work in a multiuser environment. To maintain a single updated set of data accessible by multiple asynchronous users, HyperBase was implemented as a server, based on the client-server model, allowing only one client at a time to access the shared resource, the hyperdocument (figure 3).

The HyperBase server communicates with the client applications through a byte stream protocol although it may alternatively be statically linked with an application.

To support collaborative work two mechanisms were included in the design – the lock and event mechanisms.

Event Mechanism

The event mechanism enables users to be notified when other users perform important actions in HyperBase, or when certain critical events occur on nodes or links of interest to them. Users can subscribe to any operation on any key in both nodes and links: `event(entity,operation,key)`. The following is a list of examples of how the event mechanism can be used:

- `event`(all,write,data): tell me whenever the data field of any node is written (only nodes have a data field).
- `event`(lock,all): tell me whenever any key of any entity is locked.
- `event`(all,all,all): event on all operations on all keys of all entities (system logger).

Naturally, it is possible to unsubscribe to previously subscribed events, and in addition a list of all users having subscribed to a specific event can be obtained.

Lock Mechanism

It is possible for a user to lock an entity while the content is being updated. The locking mechanism is capable of locking (and unlocking) whole nodes and links in one operation, or single keys in nodes and links one at a time, enabling one person to modify the data field of the node, other persons to modify other keys 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 keys 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 keys of nodes and links, including the data field of nodes, and in addition readers can get the login (user) name of the person having locked a specific key in an entity.

2.2 Frontend Layers

EHTS supports the concept of multiperson distributed editing, pioneered by Augment [7], which allows joint authorship of papers, programs, etc. Several persons can access the hyperdocument simultaneously over a local area network from a number of Sun Sparcstations.

Figure 2 shows the two hypertext tools in EHTS. The editor supports multiple overlapping windows on the workstation screen, one for each open hypertext node. Each window displays the name of the node at the top of the window and in the status line at the bottom. Standard window operations are supported: Windows can be repositioned, resized, closed, and put aside as icons.

Fonts in and size of node windows are saved as attributes to the node, giving windows the same look next time they are accessed. Therefore, different nodes can have different window size depending on the amount of data in the node. We also provide scrolling, because there is no restrictions on the data size in nodes. Nodes can be opened either as editor windows or as read-only browser windows.

The graphical browser shows a picture of part of the underlying hyperdocument as a graph, providing an important measure of contextual and spatial cues to supplement the users model of which nodes he is viewing and how they are related to each other and their neighbors in the graph. The browser window can like editor windows be repositioned, resized, closed, and put aside as an icon. The browser view can be split into a number of vertical views each showing different parts of the graph. Both vertical and horizontal scrolling is provided. The layout in the browser and the size of the browser window can be saved in a separate user specific layout file.

The browser provides a context-sensitive cursor: Operations in the browser can be invoked with a single point-and-click. The context of the cursor (for instance, whether it is in empty space or inside a node icon) determines which operations are currently available via the mouse buttons.

The browser supports user defined (hierarchical) views on the hyperdocument. Each user can arrange the shared network of nodes and links in the his own (hierarchical) manner. User defined views can be saved in the user specific layout file.

Integration of Tools

Both hypertext tools can be used independently of each other, but they are designed to be complementary to one another and to be used at the same time when editing. The general idea with having two separate 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.

It is possible to integrate an editor and a browser by creating a connection between them. There is no restriction on where in the local area network the two tools should run. The tools can run on different machines in the network and display their windows on different workstation screens. But naturally each editor/browser tool pair should display their windows on the same screen to get the best use of the system, though it is possible that one user is in control of the browser, while another user is in control of the editor in an editor/browser pair.

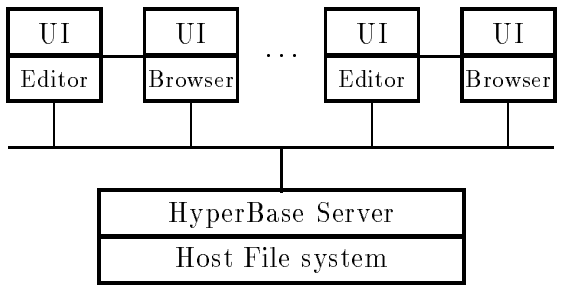


Figure 3: *The layered architecture of EHTS. Both tools are clients connected to the shared HyperBase server. Each Editor/Browser pair normally display their windows on the same screen. Editors and browsers are integrated via a direct communication channel.*

When a connection is created, it is possible to set a couple of options (figure 2) in the browser specifying that the user wants node and attribute windows to appear as editor windows when he invokes view commands in the browser in stead of being displayed as read-only browser windows.

Navigation

Navigation is supported in both the editor and the browser.

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

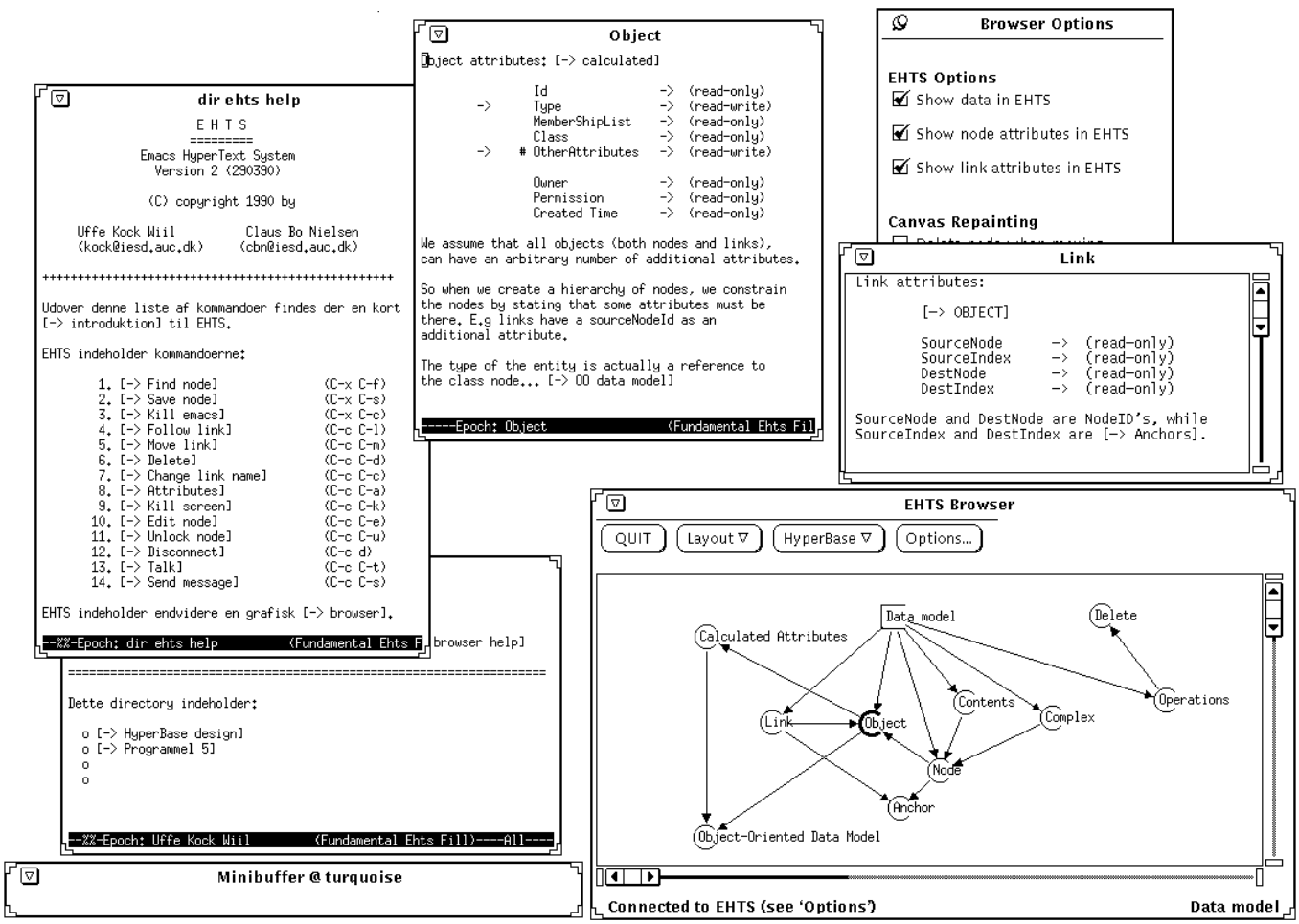


Figure 2: *An illustration of the two tools in EHTS, the editor and browser.*
The editor can have multiple windows opened, each containing the data field of a node. The minibuffer is the man-machine interaction buffer. Questions and messages from and answers to EHTS goes through this buffer.
The different icons in the graphical browser represents different types of nodes: circles are normal nodes, while squares are directory nodes. Bold faced icons indicates a lock on the data field of the node.
Browser options determine whether windows opened in the browser appears as editor windows or as read-only browser windows. Browser windows have a vertical scroll bar at the right hand side.

In the browser users navigate using the mouse. Node icons in the browser contains the name of the nodes. A click with the middle mouse button on a directory node brings the user one level deeper in the hierarchy. A click with the right mouse button in the canvas brings up a menu that allows the user to go either to the previous level, to the top level, or to a specific level in the hierarchy. A click with the right mouse button on a node brings up a menu that allows the user to view the contents of the node in a window, to view the attributes of the node in a window, or to view the attributes of outgoing links in windows. Depending on the options in the browser these window appears either as editor or read-only browser windows.

In the editor the database can be browsed by following links and opening windows successively to examine their contents. It is also possible to display a specific node by invoking the `find node` command. There is completion on node names of all nodes in the database. The editor commands are normally invoked using the keyboard, but it is also possible to follow links and to close windows using the mouse, which helps the user to a faster navigation.

Neither the browser nor the editor supports keyword, attribute, and string search in the hyperdocument.

Editing Nodes

A user can open nodes as editor windows on the screen in three different ways. Either by following a link to the node from another editor window, by invoking the `find node` command in the editor, or by selecting the *display node contents* item in the node menu in the browser with the *display windows in editor* option set.

In order to be able to change the contents of a node in an editor window, the user needs to get a lock on the data field of the node. Therefore, the editor provides `lock` and `unlock` commands. To help solve conflicts on updating shared data in HyperBase (access contentions) the users are provided with two real-time communication mechanisms, a talk and a message/mail mechanism. Collaboration issues are discussed in section 3.2.

Creating a new node is done, either by creating a link to a non-existing node or by using the `find node` command and specify a new node name. In either cases an empty window appears on the screen with the font and size attributes set to a default value.

It is also possible to change some of the attributes of nodes (name, window size, window font) and links (name, destination node) in special attribute windows on the screen or by invoking special editor commands, like `move link` and `change link name`. Attribute windows can be opened either from the browser or by the `attributes` command in the editor.

In addition to the special hypertext editor commands mentioned, all standard editor operations, like mouse/keyboard cut and paste and string search within node contents, are supported.

Appendix B contains a description of the hypertext operations in the editor and the operations provided in the graphical browser.

Monitoring

Both tools provide real-time monitoring of visible changes made by other users of the system. Visible changes (figure 2) on nodes opened in editor windows on the screen are:

- renaming nodes.
- resizing node windows.
- changing fonts in windows.
- changing the contents of nodes.
- deleting nodes.

Visible changes in the graphical browser are:

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

Furthermore, locking and unlocking of nodes are monitored in the browser. Icons of locked nodes are shown bold faced (figure 2). Collaboration issues are discussed in section 3.2.

3 Design issues

In this section we examine some of the issues in the design of EHTS, and we reflect on some of the design decisions made. Similar issues have been discussed in Conklins excellent summary of the hypertext field [5], and in papers describing specific systems, such as Intermedia [8], NoteCards [10], Neptune/HAM [6, 4], KMS [1], and Hoopertext [2].

We cannot describe all the steps in the prototyping design, but we will try to capture the most important empirical steps. We emphasize issues that characterize differences between EHTS and other hypertext systems. The issues are organized into four categories: architecture, collaboration, user interface, and data model. Experiences with HyperBase, obtained by its extensive use in different projects, are reported in a separate paper by Wiil and Østerbye [24].

The reflections on the design decisions are based on our experiences with the use of EHTS in different situations. As an example has EHTS been 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 student collaborated on a design exercise in groups of 2 to 3 participants. The design documentation was written in EHTS, which has given us a lot of experience with the use of EHTS in general and in collaborative situations.

3.1 Architecture Issues

One of the major goals of the EHTS experiment was to find *the* appropriate architecture of the hypertext based multiuser programming environment.

One of the first questions in the design of EHTS was whether the information in the system should be distributed or not. Distribution of information can take place at different levels in a system: database level and application level.

Distribution of information is mostly a matter of efficiency (speed), but also safety if the data is replicated at database level. To avoid some of the hard problems of distributed (hypertext) databases and get on with the collaboration experiments, we decided to use a centralized database server (figure 3) and distribute some of the information at application level instead, to gain speed.

One of the advantages of the client-server architecture, which is also used in the Neptune/HAM [6, 4] system, is that it supports shared, concurrent, and distributed access from the client applications to the centralized information base.

Distribution of information at application level in a system with a centralized database server can be based on the use of an event mechanism in the server, like we do it in EHTS. This methods is based on a cache in each of the tools keeping a fully updated copy of some of the information in HyperBase. When the information in HyperBase is changed, the tools receive an event. Internal event handlers in the tools can then update or discard the information in the cache depending on the chosen notification strategy. There are 4 possible notification strategies [8]:

1. Immediate update (real-time monitoring).
2. Immediate notification (real-time notification).
3. Passive notification ("mail-box" notification).
4. No notification.

We have chosen the first strategy in EHTS for collaborative reasons (see section 3.2), which means that we read in the information when the tools start up, and update the information in the cache and on the screen when it becomes inconsistent with the persistent copy in HyperBase.

The degree of distributed information is also an interesting issue. The extremities is no information cached and all information cached. If all the information in the database is cached in the tools, the centralized database server will be reduced to a centralized event and lock server keeping a persistent copy of the information. We have chosen a solution somewhere between these extremes in EHTS.

In the editor we maintain a list of all nodes (name, Unique ID) in HyperBase to provide fast completion on nodes names, and a list of outgoing links (name, UID) of all nodes opened in windows on the screen to be able to follow links faster.

In the browser we maintain a list of all nodes (name, UID) and links (name, UID, destination node ID) to monitor the changes in the graph and to support user defined (hierarchical) views on the hyperdocument.

Caching a too large portion of the hyperdocument at application level would make HyperBase drown in event generation, increase the network communication to a level of many collisions, keep the applications busy updating internal structures (and windows on the screen), and increase the consumed amount of memory by the caches, which could mean increased swapping and decreased machine performance. Caching also increases the start up time of tools and the load on both the network and HyperBase in start up periods.

We found a distribution degree that enables real-time monitoring, speeds up some of the operations, and has a reasonable start up time.

Measurements have shown that EHTS is capable of following a link and display the node in a new window in less than a second on average compared

to less than half a second on average in KMS [1] to follow a link and display the frame within the same window. KMS uses a distributed database, but do not replicate the data.

As a result of the EHTS experiment an improved architecture (figure 4) is proposed for the multiuser programming environment that should increase the efficiency (speed/memory) of the system by decreasing the load on the network and HyperBase, and reduce the amount of memory spend on caching part of the hyperdocument in the tools.

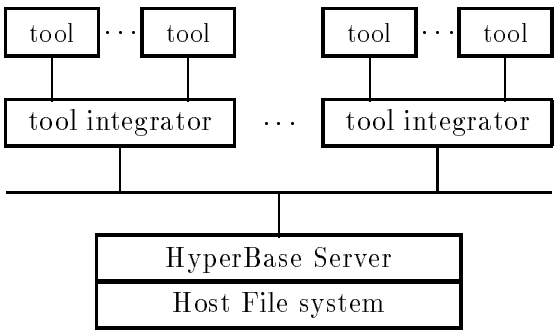


Figure 4: *The improved architecture of the hypertext based multiuser programming environment with a centralized HyperBase.*

The architecture is still based on the client-server model. A new component, a tool integrator, is introduced on the client side. The integrator could be thought of as an advanced extensible client library providing both a client-client and a client-server communication protocol.

An integrator is located on each client workstation in the network capable of running the hypertext system. Tools connect to the integrator on the workstation on which they are running instead of HyperBase. Caching is done in the integrator and the internal event handler in the integrator maintains the cache based on events from HyperBase and generates update messages to the tools.

This way the HyperBase only have to generate one event for each integrator instead of one event for each tool, and we only need to cache and maintain the basic information once (in the integrators) for each user instead of in each tool.

An open issue in connection with this architecture is whether integrators should be connected to HyperBase all the time or just when tools are connected to them. If the integrators are connected at all times, the start up time for tools would be very short, but on the other hand integrators would have to maintain their cache, putting load on the network and HyperBase, even when no tools are connected to them.

3.2 Collaboration Issues

When designing a collaborative multiuser hypertext system like EHTS a number of issues needs to be addressed [2]:

- What happens when two users want to change the same node? Is there a way to queue the requests or to be notified when the node becomes free?
- Should it be possible for instance in a graphical browser to see where the other users of the system currently works?
- When will one users presentation be updated to reflect database changes made by another user?
- Should the users attention be drawn to changed objects?

Contention handling issues in multiuser hypertext systems are similar to concurrency control issues in normal (relational, object-oriented, network, ...) database systems. Concurrency control is normally provided by the use of a locking mechanism [3]. Typically these systems provide both read and write locks and transactions to make the database look like a single user database to all users.

The following questions needs to be addressed if locks are used:

- Can one user view the current contents of a node while it is locked for editing by another?
- At what level (node, attribute, sentence, word, letter) should the contents of nodes and links be locked?
- Should it be possible to kill locks in case a user has forgotten to unlock an important node and is gone home?
- When should locks be allocated? When the user opens a node, or when he specifically tells the system that he wants to change mode from browse mode to edit mode?
- Should locks be deallocated when the node window is closed?

In a system for collaboration, like EHTS, we want the users to be aware of the multiuser situation, in other words, we want the multiuser situation to be visible and the database to be "active". Therefore, the database of a collaborative system should support a single lock type only. This lock is an exclusive

write permission allowing locked (parts of) objects to be read by the other users. Instead of transactions[2] the database should provide an event mechanism enabling users to be notified about any action on any (part of an) object taken by any user, including lock and unlock operations.

An alternative to locks is to let a version control mechanism solve the problems in connection with update contentions. This way, a user would just create a new version (a variant) of the node instead of being told that the node is currently locked and therefore write inaccessible. This solution opens the question: Should the version control mechanism support a merge between different versions of nodes?

Trigg et al. [19] talks about three fundamental issues in simultaneous sharing: access contention, real-time monitoring, and real-time communication. In [20] Wiil describes how the attribute-level event and locking mechanisms of HyperBase are used in EHTS to deal with access contentions and to provide real-time monitoring and real-time communication.

EHTS provides contention resolution at the level of attributes in nodes and links. EHTS allows any number of users simultaneously to read and display the data field of a given node in a separate window on the screen. However, permissions to make modifications to the data field is restricted to one user at a time.

Locks are allocated when the user invokes the editor lock command indicating that he wants to change mode from browse to edit mode. Locks are deallocated either when the editor unlock command is invoked or when the window is closed – therefore, we do not need a kill lock command.

All readers of the data field are notified when certain modification events occur. Following Stefik et al. [17] readers are notified as soon as possible that a data field they are accessing may be changed.

Readers are provided with four levels of modification notices:

1. All readers are notified when one person signals his intention to modify the data field of the node by obtaining a lock – and the node icon in the browser is shown **bold faced** (like `Object` on figure 2). The readers also get the name of the person, enabling them to contact him by the use of the internal talk mechanism (real-time communication). Readers can then subscribe to the event on when the writer is finished modifying and unlocks the data field.

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

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

4. When a node is deleted, the display of the node is removed from the screens of all readers of the node.

To sum up, EHTS provides help in dealing with access collisions at 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 mechanisms (talk and message/mail). Real-time monitoring enables the unfortunate persons to follow (spy on) the writers changes. Notification when the writer unlocks the data field is also a useful feature.

Our experiences have shown that the provided attribute-level event and lock mechanisms can facilitate the distribution of relevant information among participants, allow people to follow the overall progress, and allow the participants to safely edit and change the hyperstructure. Thus, the provided support goes beyond the simple mechanism of sharing data, and meets the challenges of support for the social interactions involved in sharing data, as they are stated by Halasz in [10] Issue 6: Support for Collaborative Work.

3.3 User Interface Issues

Even though the user interface design had a low priority in the EHTS experiment, we addressed a number of issues:

- Which multiuser window system should EHTS support? EHTS should run on our local area network of Sun Sparcstations.

- Should the user interface look and feel be based on well known standards, or should we create our own standard?

[2] Transactions should only be used to provide crash recovery, like HAM [4] does.

- Should we extend an already existing editor with hypertext functionality or should we create a new hypertext editor from scratch?
- Should each node be displayed in a separate window?
- How should anchors (link sources) be presented on the display?

A number of "fast" solutions are made to provide a reasonable user interface enabling us to experiment with multiuser and collaboration issues.

EHTS is developed to run under X Windows (X11.4), as this window system was the most widespread network transparent multiuser window system in our department at the development time. Because of the one-way compatibility between X Windows and Open Windows, EHTS also runs under Open Windows, which lately has become the most widespread window system in our department.

The user interface of the graphical browser is based on XView [11], an OPEN LOOK toolkit for X11. This way menus, panels, buttons, icons, scrollbars, etc. in the browser observes the OPEN LOOK standard, which a lot of users already are familiar with.

The editor is based on Epoch [12] a multiple window extension to GNU Emacs [13]. The GNU Emacs editor was chosen because it is the single most used editor in our department and it is easy to extend using the provided extension language GNU Emacs Lisp. This way we get all the normal editing facilities for free and only have to provide the special hypertext editor commands. It also means that users familiar with GNU Emacs only have to learn a dozen new editor commands in stead of learning to use a whole new editor in order to use the hypertext system.

The first version of EHTS used GNU Emacs as editor which only supports one window on the screen, like KMS [1]. When a link was followed, a new node was created, or the `find node` command was invoked the contents of the accessed node was displayed within the same window. We kept a list of previous visited nodes enabling fast backtracking, like in KMS.

Later we ported the system to Epoch, which supports multiple overlapping windows enabling a one-to-one correspondence between windows on the screen and nodes in the hypertext database, as suggested by Conklin [5] and supported in most hypertext systems [16]. It is our experience that in both editing and browsing situations it is much nicer to be able to have more than one node opened at the same time in windows on the screen.

Extending Epoch with hypertext functionality raises the question:

- How should the hypertext functionality be integrated with the existing editor functions?

Epoch functions are usually invoked by a key sequence on the keyboard. A few functions like cut and paste can also be invoked using the mouse. The provided hypertext functions can only be invoked in Epoch buffers with EHTS minor mode on. In EHTS minor mode some of the normal Epoch functions (Ctrl-x prefix): `find-file`, `save-buffer`, and `save-buffers-kill-emacs`, are redefined to operate on HyperBase nodes instead of files. The rest of the hypertext editor functions: `follow link`, `move link`, etc. are provided on the key sequences for user defined operations (Ctrl-c prefix), see appendix B.

The solution with a minor hypertext mode in Epoch makes it easy :

- to import text from and export text to the normal file system with the Epoch functions `insert-file` and `write-file` / `write-region`, which still operates on normal files.
- to switch to other editor modes, which means that the same editor can be used to provide both hypertext and all normal file editing facilities at the same time. Key bindings even have the same logical meaning in different buffer windows providing different facilities.

Having chosen Epoch as editor, most of the user interface look and feel of the editor is fixed, but we still had to come up with a solution on how to display anchors (link sources) in node windows.

We decided to use "to" links in EHTS, like in KMS [1] and NoteCards [10], as this is the only directly supported link form in HyperBase. "Into" and "Back" links, known from Intermedia [8] and Neptune [6], could have been provided by including and maintaining static keys in HyperBase links. Source anchors in EHTS are special strings: [-> *linkname*] (embedded in the text) and destination anchors are nodes.

The link presentation (anchors) are tied to the link representation through a buffer local list containing link names and the belonging link ID's. This fast solution only works if the users let the provided editor functions maintain link destinations and link names (anchors). If the users decide to update or delete anchors with normal editing commands, we do not get the internal link list updated, and we are in trouble.

This problem could be solved by using the provided buttons in Epoch. Buttons can be created as read-only (version 3.2), can have attributes attached to them specifying the presentation style on the screen, and data specifying the link(s) anchored in the button. Button objects in Epoch do not have a fixed position (start and end point) in the buffer, they stay in the context (relative position) where they were created. This way it would be possible to restrict anchor update to a few hypertext commands in the editor.

When we implemented EHTS, buttons in the available Epoch version (3.1) could not be specified as read-only. Therefore, at that time the button solution was less obvious than it is today.

3.4 Data Model Issues

The simple node and link based data model of HyperBase with a predefined number of keys in nodes and links turned out to be a problem in the prototyping design.

Before starting the prototyping design we had to decide which keys to include in nodes (figure 5) and links (figure 6). Having taken this decision, the data model is totally static.

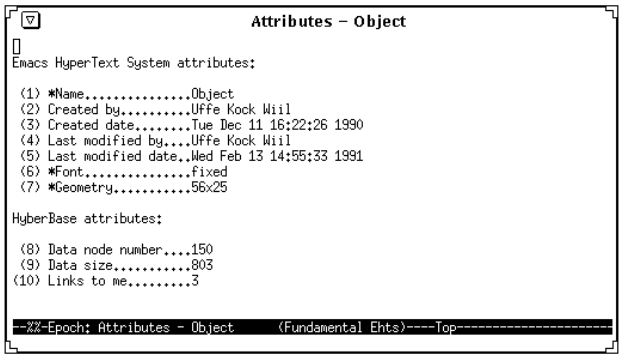


Figure 5: *Keys attached to nodes.*

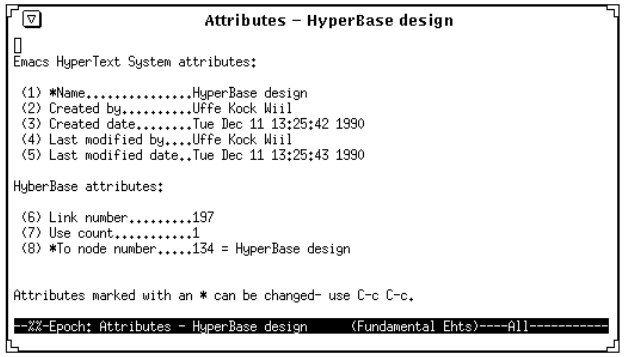


Figure 6: *Keys attached to links.*

The essential idea of a prototyping design is learning from experience along the way. This was not possible in connection with the data model. We could have used a more flexible data model providing run time attachment of attributes to both nodes and links, and a composite object. In stead we had to make some less successful decisions in the design of the graphical browser, which will be explained in the following:

Like other hypertext designers before us, we ran into the "getting lost in hyperspace" or the disorientation problem [5] in the browser (figure 7). Our solution to this problem is to provide user defined hierarchical views on the hyperdocument in the graphical browser.

A composite data model object could have been used to provide the multilevel hierarchy, like contexts partition nodes and links of a graph in a tree structure in HAM [4]. KMS [1] and NoteCards [10] also supports a hierarchical structuring of their hyperdocuments, but do not provide composites.

As HyperBase does not provide composites, we introduced different types of nodes (directory and data) to support the multilevel hierarchy, like it is done in NoteCards (fileboxes and notecards). As HyperBase does not support run time attachment of new keys, we store node types in the user specific layout file of the browser. This solution raises the issue:

- Where should new nodes be presented in the browser?

We present new nodes at top level, as we let the users decide where in the hierarchy new nodes belong. This solution implies that users have to move every single node created by both themselves and all other users to the appropriate level in the hierarchy, if they do not want the top level to be overcrowded with nodes.

Experiences have shown that users gets annoyed, when they have to move to many nodes around in the browser, and when they cant find their newly created nodes in their working directory, because they are shown at top level. We also realized it would be a good idea to provide autosave of the user defined layout in the browser. Users also gets annoyed if they after having moved a lot of nodes (forget to save the changes and) quit the browser and finds their view unchanged next time they enter the browser.

We still believe the idea of user defined hierarchical views is good, but it should be combined with a more flexible data model, like the one suggested in [25], to solve the mentioned problems. Composites would enable user defined views, as nodes can be part of more

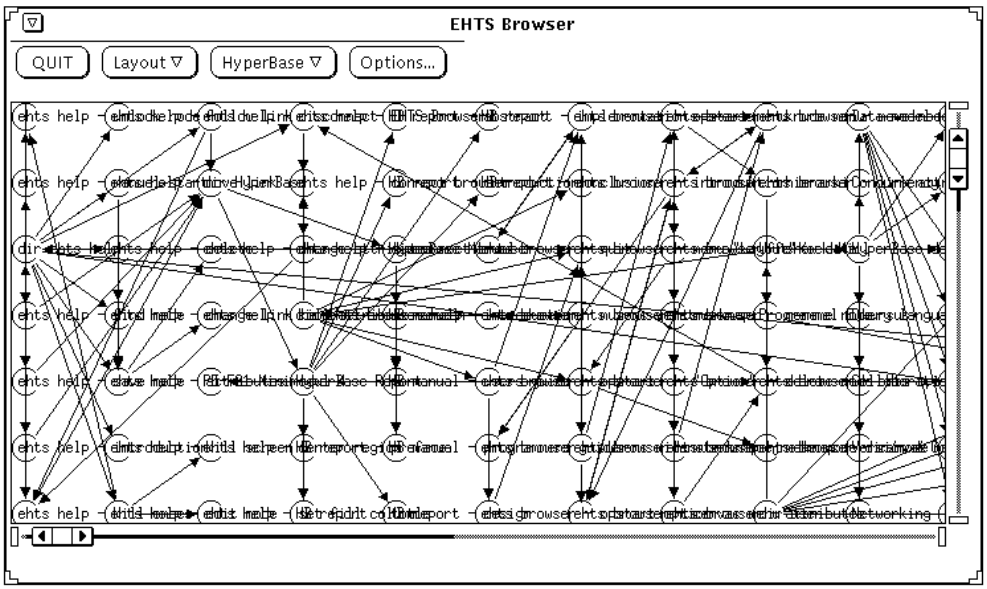


Figure 7: *The simple layout algorithm of the browser creates a disorientation problem even with a few hundred nodes in HyperBase.*

than one composite, and at the same time solve the problem with where to present new nodes, as nodes could be placed in the current workspace (composite) of the user creating the node. Different layouts of the nodes in a composite could be saved in different run time attached attributes of the composite, enabling users to have both separate and shared views, if the views of other users also could be loaded.

It is our experience that it is impossible to predict the exact outcome of a prototyping user interface design, and therefore also impossible to determine the necessary data model support in advance.

A flexible data model could have helped us in the design of the user interface by providing composites and run-time attachment of attributes in the first place or by providing an extension possibility.

4 Summary

In this paper we have given an overview, described issues in the design, and experiences with the use of EHTS, a multiuser hypertext system for collaboration.

We focused on multiuser and collaboration issues in and the architecture of hypertext systems in the EHTS experiment and were less concerned with other important issues of hypertext systems, like user interface and data model issues.

A new and improved client-server architecture for event driven multiuser hypertext systems has been presented, and it is described how attribute-level events and locks can provide powerful support for simultaneous sharing.

We described how buttons can improve the link presentation, and how a more flexible data model can solve the problems in the graphical browser.

Our experience tells us that the design of the data model has great influence on the design of the user interface. The developers of KMS [1] had a similar experience.

Ongoing Work

Based on the experiences with HyperBase and EHTS and on literature studies, a new and improved version of HyperBase is currently being developed. This version [25] is based on an object-oriented data model and an internal engine providing an extension language in which new data model objects can be derived (specialized) from the basic type hierarchy of objects. Functions can be derived (inherit functionality) from existing functions and new functions can be added to deal with specialized objects and to extend the functionality of HyperBase. The engine will also provide a special hypertext query and search language. Access control, notification, versioning control, and concurrency control mechanisms will also be incorporated in the new version.

Acknowledgments

I wish to acknowledge Claus Bo Nielsen who took part in both the development of HyperBase and

EHTS, and Carsten Ruseng Jakobsen, Hans Mejdahl Jeppesen, Per Magnus Petersen, Finn Sølvsten, and Poul Larsen who also took part in the development of HyperBase.

References

[1] 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.

[2] L. M. Berlin and V. L. O'Day. Platform and Application Issues in Multi-User Hypertext. Technical Report HPL-90-05, Software Technology Laboratory, Hewlett Packard, Feb. 1990.

[3] P. A. Bernstein and N. Goodman. Concurrency Control in Distributed Database Systems. *ACM Computing Surveys*, 13(2):185–221, June 1981.

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

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

[6] N. M. Delisle and M. D. Schwartz. Neptune: a Hypertext System for CAD Applications. In C. Zaniolo, editor, *Proceedings of ACM SIGMOD'86 – International Conference on Management of Data*, pages 132–143, Washington, D.C., May 28–30 1986.

[7] D. C. Engelbart and W. K. Englich. A Research Center for Augmenting Human Intellect. In *AFIPS Confenrence Proceedings, Vol. 33, Part 1*, pages 395–410, Washington, D.C., 1968. The Thompsons Book Company.

[8] L. N. Garrett, K. E. Smith, and N. Meyrowitz. Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 163–174, Austin, Texas, December 3–5 1986.

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

[10] 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.

[11] D. Heller. *XView Programming Manual, An OPEN LOOK Toolkit for X11.* O'Reilly & Associates Inc., 1990.

[12] S. M. Kaplan, A. M. Carroll, C. Love, and D. M. LaLiberte. *Epoch – GNU Emacs for the X Windowing System*, Oct. 1990. Release 3.2 – Based on GNU Emacs 18.55.

[13] B. Lewis, D. LaLiberte, and the GNU Manual Group. *The GNU Emacs Lisp Reference Manual*, edition 0.1 beta (version 18) edition, June 1989.

[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] J. L. Schnase, J. Leggett, C. J. Kacmar, and C. Boyle. A Comparison of Hypertext Systems. Technical Report TAMU 88-017, Hypertext Research Lab, Texas A&M University, Sept. 1988.

[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] W. Teitelman and L. Masinter. The Interlisp Programming Environment. *Computer*, 14(4):25–34, Apr. 1981.

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

[20] U. K. Wiil. Using Events as Support for Data Sharing in Collaborative Work. In K. Gorling and C. Sattler, editors, *Proceedings of the International Workshop on CSCW*, pages 162–176, Berlin, April 9–11 1991.

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

[22] 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.

[23] U. K. Wiil, K. Nørmark, and K. Østerbye. Aspects of Hyperstructure Programming Environments. Internal working note, The University of Aalborg, Apr. 1991.

[24] 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.

[25] K. Østerbye and U. K. Wiil. A Flexible data model for Hypertext-Systems. Extended abstract, The University of Aalborg, Apr. 1991.

A HyperBase Operations

The notation for describing operations is of the form:

operation (in_1,$\cdots$,in_n) $\longrightarrow$ (out_1,$\cdots$,out_n)

For all of the HyperBase operations out_1 is an integer revealing whether the operation succeeded of failed. If the operation failed the value of the integer will describe what went wrong.

A.1 Structure Operations

Links are created in two operations in order to provide many-to-one links. The first operations creates the link object with a reference to a specific node and the second operation creates a reference from a node to a link object.

CreateNode () $\longrightarrow$ (Entity)
Creates a node and returns the unique entity ID.

CreateLink (ToNode) $\longrightarrow$ (Entity)
Creates a link with a reference to ToNode and returns the unique entity ID.

UseLink (FromNode,Link) $\longrightarrow$ ()
Creates a reference from a node to a link.

MoveLink (Link,NewNode) $\longrightarrow$ ()
Moves the reference in a link to a new node.

RemoveLink (FromNode,ToLink) $\longrightarrow$ ()
Removes the reference from a node to a link object.

Delete (Entity) $\longrightarrow$ ()
Deletes an entity.

A.2 Read and Write Operations

Read (Entity,Key) $\longrightarrow$ (Value)
Reads a key of an entity and returns the value.

Write (Entity,Key,Value) $\longrightarrow$ ()
Writes the value into a key of an entity.

Enumerate (EntityType) $\longrightarrow$ (EntityList)
Returns a list of either all existing nodes or links.

A.3 Multiuser Operations

Lock (Entity,Key) $\longrightarrow$ ()
Locks a key in an entity.

Unlock (Entity,Key) $\longrightarrow$ ()
Unlocks a key in an entity.

ShowLock (Entity,Key) $\longrightarrow$ (User)
Returns the user name of the lock owner on a key in an entity.

Event (Entity,Operation,Key) $\longrightarrow$ ()
Subscribes to an operation on a key in an entity.

UnEvent (Entity,Operation,Key) $\longrightarrow$ ()
Unsubscribes to an operation on a key in an entity.

ShowEvent (Entity,Operation,Key) $\longrightarrow$ (Users)
Returns a list of users having subscribed to a specific event.

In addition to the above mentioned event operations provided by the server, there are two event handling commands included in the client library, enabling clients to receive events from the server. These are:

GetEvent (Timelimit) $\longrightarrow$ (event)
Wait for an event to arrive from the server. A time limit from zero to infinity can be set.

ReadEvent () $\longrightarrow$ (event)
Read an event, if any, and return immediately.

A.4 Other Operations

Connect (Server,UserID) $\longrightarrow$ ()

Disconnect () $\longrightarrow$ ()

B EHTS Operations

The description of operations in EHTS in divided into two sections: Editor and browser operations.

B.1 Editor Operations

Kill emacs (C-x C-c)
Disconnects from HyperBase and EHTS Browser before calling the normal save-buffers-kill-emacs command.

Find node (C-x C-f)
Opens a node window on the screen containing the data field of the chosen node in HyperBase. Space bar completion on existing node names. Typing in a new node name creates a new empty node. Notification if the node is locked by another user.

Save node (C-x C-s)
Writes the contents of the node window to the data field of the node in HyperBase, and the font and window size to node attributes. When saving the contents other users having the node opened in a window on the screen gets their display updated.

Attributes (C-c C-a)
Opens an attribute window on the screen with either the attributes of the current working node or one of the links in the node. Space bar completion on the node/link choice and on link names in the buffer window. It is possible to change some of the node (name, font, size) or link (name, destination) attributes.

Connect browser (C-c C-b)
Connect to an EHTS Browser. A machine name is required.

Disconnect browser (C-c b)
Disconnect from the EHTS Browser.

Change link name (C-c C-c)
Changes the name of a link both in the node window and in HyperBase. Space bar completion on link names. A new link name is required.

Center region (C-c c)
Center a chosen region in a node window. Used when editing.

Delete (C-c C-d)
Deletes either a node or one of the links in the current working node (and the link string in the node window). Space bar completion on the node/link choice, on node names, and on link names in the node window. Deleting a node causes the node window to disappear from the screen of all users of EHTS. A node can only be deleted if it is not referenced by other nodes.

Disconnect (C-c d)
Disconnects from HyperBase.

Edit node (C-c C-e)
Changes the node window from read-only to modify mode, by obtaining a lock on the data field of the node. Notification if the node is locked by another user. Possible notification when the other user unlocks.

Set fill column (C-c C-f)
Sets the fill column of the node window to screen width minus 3. Used when editing.

Kill screen (C-c C-k) & (shift-mouse-right)
Closes (and unlocks) node and attribute windows.

Follow link (C-c C-l) & (shift-mouse-left)
Follows a link reference from the source node causing a node window to appear on the screen. Space bar completion on link names. Typing in a new link name creates a new link to either an existing or a new node. Typing in a new node name creates a new empty node. Space bar completion on existing node names.

Move link (C-c C-m)
Moves a link destination to either an existing or a new node. Space bar completion on link names. Typing in a new node name creates a new empty node. Space bar completion on existing node names.

Send message (C-c C-s)
Send either a personal or a global message. Space bar completion on possible users of the system (not on connected users). Opens a message window on the screen. The message is send, causing the message window to appear on the screen(s) of the receiver(s), when the contents is saved. Notification if the message node is locked by another user.

Talk (C-c C-t)
Talk to another user of EHTS. Space bar completion on possible users of the system (not on connected users). Opens two talk windows (own and opponents) on the screen of both users. Notification if one of the talk nodes is locked by another user. Talk windows gets updated when the contents is saved.

Unlock node (C-c C-u)
Unlocks a node and changes the node window from modify to read-only mode.

B.2 Browser Operations

Operations in the browser are invoked using the mouse. The menu buttons in the browser head contains (figure 2):

QUIT – and disconnect from HyperBase.

Layout – user defined.

Load layout

Save layout

HyperBase

Connect – to a HyperBase.

Disconnect – from a HyperBase.

Machine – pull-right menu. Give machine name for a running HyperBase.

Options

EHTS Options – integrate editor and browser pair.

Canvas Repainting – how to repaint the canvas, when nodes are moved or deleted.

The cursor in the browser is context-sensitive:

The left mouse button is used to move (point, click, and drag) nodes icons around in the canvas.

A click with the middle mouse button on a directory node icon brings the user one level deeper in the hierarchy.

A click in the canvas with the right mouse button brings up a menu enabling users to select and display any given level in the user defined hierarchy:

Previous level – select and display previous visited level. Visited levels are kept in a stack.

Top level – select and display top level.

Goto level – select and display a specific level in the pull-right menu.

A click with the right mouse button on a node icon brings up a menu enabling the user to:

Show data either in an editor or a read-only browser window depending on the options in the browser.

Rename node in HyperBase and in the canvas.

Delete node in HyperBase and in the canvas.

Make node DIR / NODE – Change node from normal to directory node or from directory to normal.

Attributes – display node attributes, either in an editor or a read-only browser window depending on the options in the browser.

Move node to top level – used to create or rearrange the user defined hierarchy.

Move node to ... – pull-right menu. Used to create or rearrange the user defined hierarchy.

Link attributes – pull-right menu. Display link attributes for one of the links (if any) in the node, either in an editor or a read-only browser window depending on the options in the browser.