THE UNIVERSITY OF AALBORG

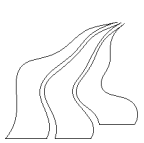
Experiences with HyperBase - a multi-user back-end for hypertext applications with emphasis on collaboration support

by

Uffe Kock Wiil & Kaster Østerbye

R 90-38 October 1990

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



Experiences with HyperBase

– a multi-user back-end for hypertext applications with emphasis on collaboration support

Uffe Kock Wiil **Kasper Østerbye**

October 1990

Department of Mathematics and Computer Science
Institute for Electronic Systems
The University of Aalborg
Fredrik Bajers Vej 7E
DK-9220 Aalborg Ø
Denmark

e-mail: kock@iesd.auc.dk

phone (switch-board): + 45 98 15 85 22
or direct: + 45 98 15 42 11 – 50 51
fax: + 45 98 15 40 08

Address all correspondence to:
Uffe Kock Wiil
on the above mail or e-mail address

Experiences with HyperBase

– a multi-user back-end for hypertext applications with emphasis on collaboration support

Uffe Kock Wiil **Kasper Østerbye**

October 1990

Abstract

This paper describes the architecture of and experiences with HyperBase, a simple multi-user back-end for hypertext systems. HyperBase is built on the client-server model. It has been designed especially to support interfaces for collaboration. The experiences from several experiments using HyperBase are reported.

1 Introduction

Hypertext has become quite popular and widespread since the introduction of NoteCards [11]. To many people, professionals as well as laymen, hypertext has become *the* way of organizing documents. This has to a high degree been possible due to the rather advanced user interfaces developed for hypertext.

For an introduction to the concept and examples of hypertext systems see Conklin [8], Nielsen [18], and the special issues on hypertext in the CACM [2], ACM TOIS [3], and BYTE [5].

Anyone who has used hypertext systems in an organizational setting will however soon recognize the need for hypertext systems that allow multiple persons working together on the same document at the same time, preferably using multiple machines connected through a network.

"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" [12].

The current generation of hypermedia does not provide support for collaboration beyond the simple mechanism of sharing data. Halasz maintains in [12] – 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."

This calls for a system where there is a clear separation between the user interface, and the storage of nodes and links. The hypertext back-end "HyperBase" developed here at The University of Aalborg, provides such a stor-

age medium, with the low-level facilities necessary to realize a hypertext system for multiple users.

This paper describes a project that has been carried out at The University of Aalborg by 6 masters students. The purpose of the project was to design and test a hypertext back-end. The project lasted two semesters. During the first semester the hypertext back-end was designed and implemented, and in the next semester the students divided into smaller groups and made different applications to test out HyperBase. One group focused on the data model and a total environment. Another group was concerned with problems concerning interface and collaboration, and the last group used HyperBase as a storage medium for C++ program fragments. Furthermore, another project group used it as a storage medium for Smalltalk-80 objects. The back-end has therefore been tested fairly well, and the diversity of the applications has given us a lot of knowledge about what is good and bad about the design of HyperBase.

HyperBase is still in use, and some of the problems that were found in its design and implementation are being worked on.

The paper is organized in two major sections. First, a description of the design and implementation of HyperBase, without any reflections and alternative solutions. In the next section problems are established, improvements are suggested, and positive things are described; in other words, we report our experiences with HyperBase.

2 HyperBase

This section gives a short general view of the architecture of HyperBase. For detailed information about HyperBase see the report *Design and Implementation of a HyperBase* [25].

The intentions with the design of HyperBase were to construct a general tool suitable as back-end for different hypermedia[1] applications. HyperBase should capture the general idea of hypermedia, provide operations on the basic entities (links and nodes), and be easy to extend and adapt to a certain field. In other words, HyperBase is thought of as a configurationable file system for hypermedia applications.

2.1 HyperBase architecture

HyperBase is designed as a stratified system (see figure 1) 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.
- a multi-user part – including locking and event handling.

The two basic levels of HyperBase could operate alone with just one user, or in other words, the multi-user part provides all the facilities necessary for the multi-user environment e.g. locking and event handling.

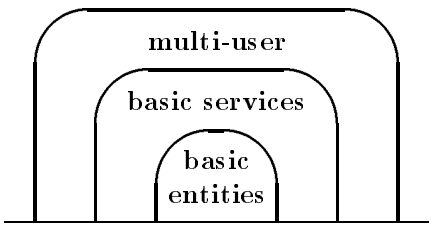


Figure 1: *The different layers of HyperBase.*

In the following sections a more detailed description of the three levels of HyperBase will be presented.

2.2 Basic entities

This level provides interface to the underlying file system UNIX[2]. The interface provided by this layer is a simple model of links and nodes:

1. Links can only refer to nodes.
2. Links are unidirectional.

[1] The term *hypermedia* is used instead of *hypertext* in this section, to emphasize that the storage support in HyperBase is not limited to pure textual applications.

[2] UNIX is a registered trademark of AT&T.

3. Links are many-to-one, that is, links can only refer to one node, but

4. Nodes can share links.

5. Nodes can be the source for many links.

6. Because of the flat link and node hierarchy, links can point to any node in the hyperdocument.

7. Both links and nodes can have a *predefined*[3] set of 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 client programs 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 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 a number of additional keys that 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. Unlike nodes, links have no variable sized keys.

2.3 Basic services

The basic services provided by HyperBase can be divided into three different categories of operations. Operations concerning:

1. Creating and maintaining structures in HyperBase. The operations are:

   - `createnode`,
   - `createlink` (between two nodes)[4],
   - `delete` (link/node),
   - `movelink` (move reference in link to refer to another node),
   - `removelink` (remove link reference in node).

2. Reading and writing in HyperBase. The operations are:

   - `write` (node/link, key, value),
   - `read` (node/link, key, value).

   HyperBase also provides a function capable of returning a list of all existing links or nodes in the hyperdocument:

   - `enumerate` (node/link).

3. Connection and disconnection of sessions to HyperBase. The operations are:

   - `connect` (server, user),
   - `disconnect`.

   "Server" is the name of the machine on which the HyperBase server is running. "User" is the name a user will be known under when the server distributes events to other users.

All operations return an integer, which reveals whether the operation succeeded or failed. If the operation failed the value of the integer describes what went wrong.

2.4 Multi-user aspects

HyperBase has been designed with special support for collaborative work in a multi-user 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 (see figure 2).

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

[3] Before compiling the HyperBase server the user can specify names of keys and the *fixed* maximum size of their values - see later.

[4] It is possible for two nodes to share a link, that is, a node can use an already existing link to refer to another node.

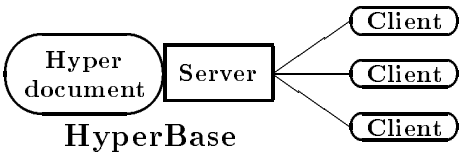


Figure 2: *The client-server model. The HyperBase server has direct access to the shared resource (the hyperdocument). All clients must access the hyperdocument through the HyperBase server.*

2.4.1 Locking and event handling

It is possible for a user to lock an entity (either a node or a link) while the content is being updated. This way all other users are denied read and write access to the locked entity. The following locking operations are provided by HyperBase:

- `lock` (node/link),
- `unlock` (node/link)
- `showlock`, returns the user name of the client that has locked a specific entity.

The event mechanism provides the users with the possibility of being notified when other users perform important actions in the database, or when certain critical events occur on nodes or links of interest to them. Users can subscribe to any operation (including the lock and unlock operations) on any key in both nodes and links (0 means all):

event(0,write,data): tell me whenever the data field of any node is written (only nodes have data fields).

event(0,write,0): tell me whenever any key in any entity is written.

event(0,0,0): event on all operations on all keys on all entities (system logger).

The following event operations are provided by HyperBase:

- `event`, described above.
- `unevent` (node/link, operation, key) – unsubscribe to previous subscribed event.
- `showevent`, returns the names of the clients that have 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`, wait for an event to arrive from the server. A time limit can be set from zero to infinity.
- `readevent`, read an event, if any, and return immediately.

2.5 Implementation

In order to simplify the implementation, the test, and the maintenance, HyperBase is separated into functional blocks each performing specified tasks. The functional blocks of HyperBase are shown i figure 3.

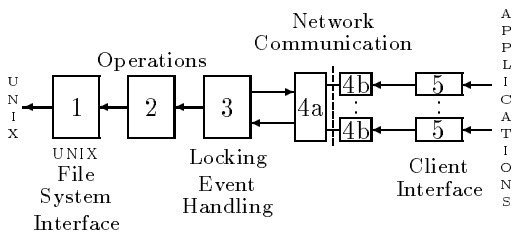


Figure 3: *HyperBase divided into five functional blocks.*

Each block is implemented as a C++ [22] class. Block 1 through 4a make up the HyperBase server, which is a single process running at the server machine, while block 4b and 5 are the client library, which is part of an application program (e.g. a hypermedia application) running at each client machine.

Block 1 provides storing and retrieval of the basic entities on the host file system (UNIX), block 2 provides the basic services, while block 3 and 4a are the multi-user part of HyperBase (see figure 1). The dotted line between block 4a and 4b illustrates the physical network, e.g. an Ethernet.

2.5.1 File interface

An obvious way to store links and nodes would be to store one entity (link or node) in a file of its own. Unfortunately, opening and closing files is very time consuming in UNIX. Therefore, this solution would place a bottleneck at the very bottom of HyperBase.

An alternative solution would be to store all information in a single file, which could be kept open at all times. Here we would run into problems of storage management, e.g. how to reclaim storage when links and nodes are deleted.

We have chosen a file representation where the links and nodes are stored in three files. One file for links, one for nodes and one for the data fields of the nodes. The reason for keeping the data field separate from the node (see figure 4) is to make all nodes the same size. The data fields are divided into blocks of

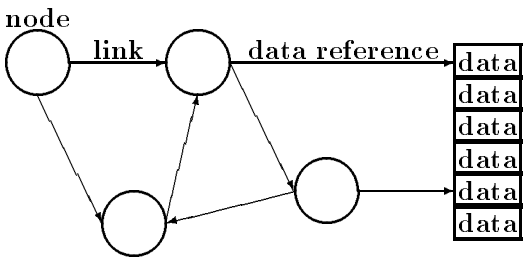


Figure 4: *An illustration of the implementation of the simple HyperBase data model, consisting of links and nodes and the separation of the data field from the nodes (all nodes can have a data reference).*

equal size and organized similar to *i-nodes* in UNIX. Thus the size of the basic entities in all three files is the same, which makes access and management very simple and efficient[5]

Before the server is compiled, a number of different things can be specified in a simple *configuration file*. For instance user defined keys for links and nodes, and the block size for the data fields. The configuration file determines the actual size of the entities in the three different files.

[5]In timing of HyperBase, we have not been able to measure any difference in storage into single nodes or to UNIX files - only networking seems to be the limit.

2.5.2 Networking

The HyperBase server communicates through TCP/IP protocols (sockets), so clients can connect to a server anywhere in the network. However, there is only one HyperBase server for each HyperBase – in other words, no distributed database management, where several HyperBase servers on different machines maintain the same HyperBase. The number of clients that can be connected to a server depends on the number of available sockets.

3 Experiences with HyperBase

This section describes the hands on experiences with HyperBase, which are obtained by its extensive use in different projects. See appendix A for descriptions of the projects in which HyperBase is used as a storage medium. First, a description of some problems that were discovered, and improvements that were made during the use. Then the positive experiences with the design and the implementation are described.

Along with the description of problems, improvements, and positive experiences, a comparison is made of the features in HyperBase with features in two other multi-user back-ends for hypertext systems. While HAM [6] (Hypertext Abstract machine) is a transaction-based server for a hypertext storage system, KMS [4] is a total distributed hypertext environment for managing knowledge in organizations. Therefore KMS cannot be compared directly with HAM and HyperBase, but some of the features of the KMS system are logically provided by a storage subsystem, which makes a comparison possible. However, it should be kept in mind that HAM is intended as a general purpose hypertext back-end, while what we call the storage subsystem of KMS is intended for use in KMS only.

The items of the following sections are listed randomly, that is, not in line of increasing or decreasing importance.

3.1 Problems and improvements

Links

Most of the projects discovered the need for system supported bidirectional links. Bidirectional links could be provided by the use of a static user defined key in the link keeping track of the source node, or by the use of two of the provided unidirectional links, but both these solutions require extra communication between the clients and HyperBase. The communication between clients and HyperBase should be minimized to keep a reasonable load on the HyperBase server.

The provided many-to-one link feature was never used in any of the projects. A one-to-one link possibility (one source node and one destination node) has shown to be sufficient.

In KMS, links are components of nodes and not separate objects like in HAM and HyperBase. HAM provides bidirectional links unlike both KMS and HyperBase. KMS links can be used to invoke programs.

Based on our experience we believe that links should be one-to-one, bidirectional, and separate objects with a possibility to attach attributes to them. Furthermore, like in KMS it should be possible to invoke programs when following a link. This feature is also called procedural attachment [8].

Attributes

Attributes (keys) in links and nodes are static except the data field in nodes. The need for run time binding (allocation) of new attributes to both links and nodes is a very important conclusion in most of the projects using HyperBase.

A user of HyperBase, as it is now, must know all the necessary keys before he starts (configures and compiles) the server. This makes it difficult to add new tools to the hypermedia environment, because most likely new tools need to add new keys to links and nodes.

The variable size data field in nodes could be used for storing dynamic attributes in addition to data, but this requires an interpretation of the data field at application level. This would have the unfortunate consequence that one would not be able to make an event mechanism that distinguishes between attributes. The use of the data field for attributes would only be possible for nodes, so we still have the problem with attributes in links, as links do not have a data field.

HAM supports the use of attributes in links, nodes, and contexts. A context is a directory-like collection (a partition) of links and nodes. KMS does not support attachment of attributes to its objects (called frames).

Our experience is that links and nodes should have run-time allocation of attributes and data, and this should be managed by HyperBase rather than by the clients.

Locking

During the use of HyperBase it became clear that the locking mechanism was not as usable as first thought. In its original form, the locking mechanism was only capable of locking whole links and nodes (hard locking), that is, all keys including the data field in nodes, and not specific keys in entities one at a time. Furthermore, it was not possible for users to read keys in locked entities.

The first problem means, that when one user is updating for instance the data field (the node is locked), other users cannot update other keys in the node. The latter problem makes it impossible to navigate through a hyperdocument when some of the links or nodes are locked.

It became clear, that in order to use the locking facility, it was necessary to change it, so that it is possible to lock specific keys in entities one at a time, and to lock whole entities in one operation, if necessary. Besides that, it is now possible to read keys[6] in locked links and nodes.

Events

The event mechanism is a very important part of the collaboration support in HyperBase, and to make it even more useful, some (small) improvements can be made.

A more sophisticated way of subscribing to events should be provided. A client should

[6] The data field in a node is a key, so it is now also possible to read data when the data field is locked.

be able to set up a pattern, matching his exact needs for event subscription. He should be able to specify (in one subscription): tell me whenever any client, except me, creates a new node, or: tell me whenever any operation, except lock operations, is performed on any key in links and nodes.

The latter event pattern matching can also be set up with the existing event mechanism, but it requires one subscription per operation, minus the lock operation naturally. The event pattern matching feature is therefore mostly a matter of efficiency, but it also remedies a small event problem discovered in some of the projects mentioned in appendix A – it should be optional for a client to receive events on his own actions performed in HyperBase.

Query and search

A major conclusion of the projects using HyperBase is, that it is lacking a query and search mechanism. This feature was intentionally left out in the design of HyperBase, because it was to be constructed in a one semester project.

The only information about the hyperdocument that can be obtained, is a list of all links or nodes[7]. With this function (`enumerate`) clients are able to get started with knowledge of (part of) the hyperdocument, but a lot of other query and search possibilities should (could) also be present in a hypertext backend:

- fast keyword, attribute and string search in the hyperdocument (a must),
- extraction of specific links and/or nodes from the hyperdocument, which match a specific description: links or nodes of a certain type, with certain attributes, locked links or nodes, all links or nodes etc. (a must),
- extraction of specific subgraphs of the hyperdocument, for instance nodes of a certain type using links of certain types (pattern matching).

[7] We never had more than a few dozens of thousands of nodes, so we did not have problems with this list being too big.

Because of the lacking query and search facilities, it is impossible to get a completely updated picture of (part of) the hyperdocument. In a graphical browser for a multi-user environment, it is nice if one can see which nodes are locked by other users [24]. With the enumerate and read functions, you can get a list of all links and nodes, and how they are interconnected, but you cannot get a list of the links and nodes that are locked by other users already connected to the hypertext system. It is crucial to have a query mechanism which can return a fully updated state (picture) of the underlying hyperdocument.

With respect to search and query we mention that KMS only supports text string search within a hierarchy of frames, while HAM provides a filtering mechanism that allows subsets of HAM objects to be extracted from large graphs. HAM filters allow the user to specify visibility predicates, which are expressions relating attributes and their values. Filtering of the following items is possible in HAM: Contexts in a graph, links and nodes in a context, instances (earlier versions) of links and nodes in specified contexts, and a set of links and nodes in specified contexts based on a specific link ordering.

Halasz [12] divides the search and query facilities into two broad classes: *content search* and *structure search*. In content search, all links and nodes in the network are considered to be independent entities and are examined individually for a match to the given query. Structure search involves some kind of query language geared toward describing hypertext network structure, and a search engine capable of satisfying the queries expressible in the hypertext query language. Schütt and Streitz are working on a Hypertext Query Language (HTQL) for their hypermedia engine, which is also called *HyperBase* [20]. The development of HTQL will be reported in a future paper by Schütt [21] some time in 1991.

Networking

In situations where clients make extensive use of the read and write operations, communication times turned out to be a problem. The efficiency of the network protocols was improved by a factor of 10, by introducing a *toggle-read-*

write-buffering function in HyperBase.

Before sending a bunch of read and/or write requests, the client tells the server that he does not want any return values in the near future, by sending a special command to the server. When this option is set, the server buffers all return values, until the client uses the special toggle command again.

This extension of HyperBase enables clients to send large numbers of read and/or write operations to the server in one bunch. After sending the last request, the clients use the toggle command again, and receive all the return values in one bunch.

Data model

Two of the projects [24, 15] found the data model[8] (only links and nodes – see figure 4) in HyperBase too simple. The projects discovered the need for a higher abstraction level than the flat hierarchy of links and nodes.

One of the projects [24] carried out a small experiment, with a directory-like hierarchy introduced at application level. Nodes could be perceived as either directory nodes or data nodes in the graphical browser, showing one level of the hierarchy. The experiment revealed the need for some form of abstraction mechanism in HyperBase, to support users navigating through hyperspace. When there is more than a few dozens of interconnected nodes, it gets **very** difficult to find the desired information in the network of links and nodes.

Another project [15] introduced an extended data model at application level, based upon the somewhat simple data model in HyperBase. The model, called the structure-atom model, divides nodes into two types: those containing data (atom nodes) and those creating the structure (structure nodes). Links in the structure-atom model are bidirectional, and are constructed of two of the simple unidirectional links of the underlying data model. Nodes contain a static user defined key *type*, which specifies the type of the node. Structure nodes cannot contain data and can be compared with directories, and atom nodes can be compared with files in a normal file system e.g. UNIX.

In KMS, the data model consists of a set of interlinked, screen-sized workspaces called frames. Frames are organized in a tree hierarchy. The HAM data model also provides a hierarchy of objects within a graph (contexts, nodes, links, attributes). Both HAM and KMS support different versions of links and nodes.

The conclusion concerning data models and HyperBase must be, that it is necessary to have a higher level of abstraction in HyperBase, than the simple link and node based data model. Our guess is, that this abstraction mechanism should be somewhat like the directory abstraction in normal file systems (hierarchy), combined with some kind of partitioning concept to support multiple authors working together on a document – contexts – as suggested by Delisle and Schwartz [9] and implemented in HAM [6]. Halasz [12] also states the need for augmenting the basic link and node model with a composite, and Schütt and Streitz [20] have introduced a composite, called *HB_Complex_Objects*, in the data model for their hypermedia engine.

When introducing a higher level of abstraction to the simple HyperBase data model, a need for types of links (organizational/structure and cross-reference) like those provided by both HAM and KMS arises. Furthermore, like in both KMS and HAM it ought to be possible to retrieve old versions of links and nodes.

3.2 Positive experiences

Collaboration support

As mentioned earlier, HyperBase was especially designed to support collaborative work in a multi-user environment. The collaborative situation is supported by the event and locking mechanisms:

- The event mechanism can notify interested parties, when certain critical events occur in the shared network of links and nodes, that is, when important events have occurred on links and nodes of interest to them.

[8] Seen from the users point of view, the data model consists only of links and nodes – it is transparent to the users that the data field is separated from nodes.

- The improved locking mechanism is capable of locking single keys one at a time, making it possible for one person to update a key in the node, another person to update the data of the node, while yet another person is annotating it by creating links. In other words, this 'soft' locking mechanism makes it possible for several persons to share data within a node, instead of sharing a network of links and nodes, with only one person operation on a link or a node at a time. It is also possible to read the contents of locked keys of links and nodes, including the data field in nodes.

Together, these two mechanisms can support multiple users working in the same environment, as have been the case with all the projects using HyperBase. The mechanisms 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 [12].

Two of the projects [24, 15] ended up with a prototype of a hypertext environment, in which multiple users located on different machines were able to collaborate on writing source code and documentation. These environments were heavily based on the use of the event and locking mechanisms. The locking mechanism, to ensure exclusion of other users from data and keys being updated by one user. The event mechanism, to tell when to update contents in text and browser windows with all users[9], and, in case of conflicts, to tell users who have tried to access locked data and/or keys, when these become unlocked and therefore accessible to the interested user.

In other words, in the experiments the event mechanism is used to keep an updated picture of (part of) the hyperdocument with all the users, and to keep users informed about important actions. The locking mechanism is used to avoid conflicts on updating shared data.

One of the projects [15] used the locking and event mechanisms to build a *talk* mechanism inside the prototype hypertext environment.

The lock and event mechanisms as they are designed in HyperBase are more or less unknown features in both HAM and KMS – naturally both systems support sharing of data between multiple users. HAM has little or no direct collaboration support, but security for data contained in a graph is provided through its access control list (ACL) mechanism. Attaching an ACL to an object is optional. An ACL entry consists of a user or group name and a set of permissions. A user is anyone who has access to the graph, a group is a list of users, and the available permissions are *access*, *annotate*, *update*, and *destroy*. Collaboration support 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. There is no locking mechanism in KMS like in the earlier version ZOG. When a conflict arises a new frame is created, to avoid destruction of changes to the frame made by other users. Like HAM, KMS supports protection of ownership.

We never discovered the need for a protection mechanism in HyperBase since all the applications using HyperBase were used by a small group of people knowing and trusting each other. But should the need arise, a protection mechanism can be built into HyperBase at application level, based on a number of user defined keys: owner, group, and permissions.

Efficiency

That HyperBase is quite efficient was proven in the projects. For instance, the prototype of the hypertext environment in the project [24] is able to follow links and open new windows in less than a second, even when there is a reasonable load on the network.

In the design of KMS the goals of the system response when following a link was to be able to access and display a random frame across a wide-area network, in less than 0.25 seconds

[9] When a user saves the contents of a text window corresponding to a node in the HyperBase the content is updated with all other users having this node opened in a text window on the screen.

on average.

The efficiency of HyperBase could probably be increased by distributing the nodes and links like it is done in KMS. But the complexity of the system would also increase, and new problems would arise. Like how to represent link destinations when links are pointing from one part of the HyperBase to another. Like HyperBase, HAM is a single server hypertext back-end.

Design

The division of HyperBase into functional blocks (see figure 3) each corresponding to (a part of) a layer in the stratified system model of figure 1, was a good basis for the implementation of the system. It was easy for different persons in the group to implement different blocks because of the specified block interfaces. Besides that, HyperBase was easy to debug, maintain and extend with new functions because of the good understanding of the bounded functionality in the blocks.

Communication protocol

The communication between the HyperBase server and the clients is based on TCP/IP protocols (sockets). This has proved to be a good idea, because sockets are very well-known and used foundations for communication between workstations in LAN's. Therefore, a lot of different programming languages support the use of sockets.

In the different projects the following programming languages were used on the client side of the network: GNU Emacs Lisp [17], Smalltalk [10], C [16], and C++ [22] though the client library is implemented in C and C++ only. In other words, in some of the projects, applications were built that communicated directly with HyperBase through sockets without the use of the implemented client libraries.

4 Related Work

A lot of different hypertext and hypermedia systems have been developed until now, but only a few independent hypertext engines or back-ends have been developed.

Many of the features of the Hypertext Abstract Machine (HAM) [6] have already been discussed in different parts of this paper: HAM is a general-purpose, transaction-based, multi-user server for a hypertext storage system. The data model in HAM is based on five different types of objects: graphs, contexts, nodes, links, and attributes. Ham provides an automatic version history mechanism, an access control list (ACL) mechanism, and a filtering mechanism which allows subsets of HAM objects to be extracted from large graphs.

The hypermedia engine approach (also called *HyperBase*) of Schütt and Streitz [20] is based on an existing relational database management system – Sybase. The data model is built on top of Sybase, and provides three different types of first class objects: *HB_Nodes*, *HB_Links*, and a composite *HB_Complex_Objects*, while our approach is built from scratch. The hypermedia engine approach of Schütt and Streitz focuses on the development of a data model and a query language with respect to the data model, while our approach is more concerned with collaboration issues, and therefore provides a simple data model and no search and query support.

Yet another approach to hypertext back-ends is "The Toolkit Approach to Hypermedia" by Puttress and Guimaraes [19]. This paper describes a toolkit that can be used by developers to add hypermedia functionality to their tools, independently of their particular application or environment. The toolkit provides a simple hypermedia data model and an object-oriented user interface. Eggs, the storage system interface, is a simplified version of HAM using roughly the same data model objects as HAM.

5 Conclusion

The development of HyperBase was a one semester student project, and because of the short development period HyperBase has some shortcomings: a too simple data model and the lack of search and query facilities. The emphasis was on collaboration support, and some new mechanisms were introduced.

In spite of the lacks and possibilities for improvements, HyperBase has proven to be easy to use as back-end in different hypertext applications, especially applications concerned with multiple users working in the same environment, or in other words, multiple users working on the same project. Furthermore, HyperBase has proven easy to maintain, because of the designed system model.

The developed locking and event mechanisms, especially the introduction of the event mechanism, are the first steps towards a solution of some of the problems with collaboration support that Halasz establishes in [12] – Issue 6: Support for Collaborative Work. We hope that other people will start where this work ended, and further develop the event (and locking) mechanism in the future.

Future Work

At the moment two groups of master students are working on different extensions of HyperBase, respectively version and configuration control (VCC)[10], and an extension language in which users can specify changes to the HyperBase and add new functions at run-time.

In the next year or so a new and improved version of HyperBase partly based on the results in the student projects will be developed at The University of Aalborg. We have developed a new general model for integration of tools[11]. Based on this model HyperBase, will be redesigned and (most of) the mentioned lacks and improvements will be incorporated in the new version.

The new HyperBase (version 2) will be transaction-based, and in long terms we might develop a distributed version of HyperBase (version 3). The developed model does not limit HyperBase to be placed on one machine.

[10] It is the plan that a VCC layer will be added on top of the other layers of HyperBase.

[11] We consider HyperBase, hypertext editors, hypertext browsers etc. to be tools just like normal editors, compilers, debuggers etc.

Acknowledgments

Comments from Carsten Ruseng Jacobsen and Bent Bruun Kristensen on earlier versions of this paper have improved the representation. Bent Bruun Kristensen and Kurt Nørmark both contributed with descriptions of projects using HyperBase, which they have been supervising.

Appendix

A Using HyperBase

This appendix briefly describes the four different applications of HyperBase as a storage medium, at The University of Aalborg.

In addition to the four projects that have used HyperBase, a commercial company called RegneCentralen (RC), which develops computers and software, has ported HyperBase to a RC 900 with 3/86 architecture running UNIX V. Their intentions are to use HyperBase as a storage medium for an on-line annotation and documentation system.

A.1 Integration of Tools in a HyperText Environment

The project [24] aimed at the development of a programming environment which combines the concepts of *incrementality*, *integration*, and *interactiveness* (I^3) with the concepts *multi-user* and *multi-machine* (M^2). This combination is called a *multi programming environment*.

The project 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 using the same environment, there is going to be a need for man-man interaction through the machine. For example, how should the actions performed by one user influence on the other users of the environment? Secondly, integration of tools that run on many different machines requires some rethinking of what constitutes a good architecture for the environment.

The hypertext concept is used as a framework for development of the multi programming environment. A prototype hypertext system – Emacs HyperText System (EHTS) – consisting of a text editor and a graphical browser using HyperBase as back-end, was developed to run under X Windows. EHTS was developed with the M^2 aspects in mind, in other words EHTS has special support for collaborative work. Epoch [7] – a multiple window version of GNU Emacs [17] – was used as basis for the text editor, and the graphical browser was developed using XView [13]. These basic hypertext tools were integrated into a multi programming environment.

Based on these experiments, an architecture for hypertext systems is proposed. The model is explicitly taking care of the multi machine/user aspects, and opens up for future extension towards a distributed HyperBase.

A.2 Using HyperText for Project Support

The project [15] discusses, designs, and implements a prototype of a programming environment based on the hypertext concept. It is investigated how well this environment supports a project group in developing software and documentation.

The report focuses on the structuring aspects of hypertext, and discusses in great deal a model called the structure-atom model. The idea is that all nodes are divided into two groups: Those only containing data, and those structuring this data. The objects of the developed data model – structure nodes, atom (data) nodes, and links – are stored persistently in HyperBase.

The problem of how to store and access programs and documents using the structure-atom model is discussed, and also the topic of report generation is addressed.

A prototype of the environment was developed based on GNU Emacs using HyperBase as back-end. The environment was self applied in the sense that the whole report was written using the developed prototype.

A.3 AVIL – A Visual Interconnection Language

The purpose of the project [23] is to investigate the advantages of using a graphical interface in a programming environment supporting multiple users working on persistent information.

The project includes a design and an implementation of a visual programming language called AVIL. The purpose of AVIL is to clarify program interrelations. Relations may be viewed and edited graphically.

Programs are organized as graphs where nodes contain program segments and edges describe relations of various kinds between program segments. Hence *programming in the large* (graphical) is separated from *programming in the small* (textual).

An experimental version of AVIL is designed and implemented for the language C++ [22]. The general relations in object-oriented languages and special relations in C++ are supported. The AVIL environment is implemented as a graphical hypermedia front-end. The hypermedia back-end stores the program graphs in HyperBase. The implementation is experimental, with the purpose to investigate the capabilities of graphical interconnection languages.

The project concludes that AVIL has proven easy to work with. This means that more efficient program development and maintenance is supported by the graphical presentation of the relations between program fragments. At the technical level the approach may introduce rather complicated consistency problems. Among the possible evaluation criteria for such languages it is concluded that AVIL for a combination of application area and abstractional level is superior to the similar languages evaluated as part of the project.

A.4 An Environment based on Graph Grammars

The main theme of the project [1] is to evaluate the use of graph grammars as a means for constraining a program network. A graph grammar framework has been implemented in Smalltalk [10]. The nodes and links of graphs are stored persistently in HyperBase.

The applicability of graph grammars has been tested in two experiments. The first experiment combines graph representations of classes and methods in Smalltalk with JSD entity structure diagrams [14]. The second experiment is oriented towards the implementation of the graph grammar framework itself. Graph grammar productions can be represented as graphs, and the legal production graphs can be defined and constrained by a graph grammar. Such a graph grammar has been elaborated in the second experiment.

It is the overall experience of this project that graph grammars represent a useful formalism for constraining graph representations in the domain of programming environments. A major advantage of graph grammars, as opposed to the other constraint formalisms we have considered, is that graph grammars are constructive, and that a syntax-directed style of interaction can be built on top of them. In several situations it turned out to be useful to form abstractions of several graph productions, and to use these abstractions as "graph edit operations".

References

[1] Abrahamsen, P. et al. *Introduction of Graph-grammars in Programming Environments.* Project report from Dept of Mathematics and Computer Science, Institute for Electronic Systems, The University of Aalborg, June 1990.

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

[3] *Special Issue on Hypertext.* ACM Transactions on Office Information Systems, 7(1), January 1989.

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

[5] *Special Issue on Hypertext.* BYTE, 13(10):234-269, October 1988.

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

[7] Carroll, A. and Kaplan, S. *Epoch – GNU Emacs for the X Windowing System.* Online documentation, March 1990.

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

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

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

[11] Halasz, F., Moran, T., and Trigg, R. *NoteCards in a Nutshell.* Proceedings of the 1987 ACM Conference on Human Factors in Computer Systems (CHI+GI '87), pages 45 - 52, Toronto, Ontario, April 5 - 9, 1987.

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

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

[14] Jackson, M. *System Development.* Prentice-Hall International, 1983.

[15] Jacobsen, C. et al. *Urd – Using the Hypertext Concept to Project Support.* Master of Science Thesis, Dept of Mathematics and Computer Science, Institute for Electronic Systems, The University of Aalborg, June 1990.

[16] Kernighan, B. and Ritchie, D. *The C Programming Language.* Second Edition, Prentice Hall Software Series, 1988.

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

[18] Nielsen, J. *Hypertext & Hypermedia.* Technical University of Denmark, Academic Press Inc, 1990.

[19] Puttress, J. and Guimaraes, N. *The Toolkit Approach to Hypermedia.* To be published in the Proceedings of the 1^{st} European Conference on Hypertext (ECHT'90), Versailles (Paris), France, November 27 - 30, 1990.

[20] Schütt, H. and Streitz, N. *HyperBase: A Hypermedia Engine based on a Relational Database Management System.* Arbeitspapiere der GMD (469), GMD-IPSI Darmstadt, July 1990.
To be published in an abbreviated version in the Proceedings of the 1^{st} European Conference on Hypertext (ECHT'90), Versailles (Paris), France, November 27 - 30, 1990.

[21] Schütt, H. *HTQL – A Hypertext Query Language.* To be published.

[22] Stroustrup, B. *The C++ Programming Language.* Addison–Wesley, 1987.

[23] Sølvsten, F. *A Visual Interconnection Language*. Master of Science Thesis, The University of Aalborg, June 1990.

[24] Wiil, U. and Nielsen, C. *Integration of Tools in a Hypertext Environment* (in Danish). Technical Report IR 90-02, Dept of Mathematics and Computer Science, Institute for Electronic Systems, The University of Aalborg, September 1990.

[25] Wiil, U. et al. *Design and Implementation of a HyperBase*. Technical Report IR 90-03, Dept of Mathematics and Computer Science, Institute for Electronic Systems, The University of Aalborg, September 1990.