Experiences with HyperBase: A Hypertext Database Supporting Collaborative Work

Uffe Kock Wiil
Department of Computer Science
Aalborg University, Denmark
Email: kock@iesd.auc.dk

Abstract

This paper describes the architecture and experiences with a hyperbase (hypertext database). HyperBase is based on the client-server model and has been designed especially to support collaboration. HyperBase has been used in a number of (hypertext) applications in our lab and is currently being used in research projects around the world to provide database support to all kinds of applications. One application from our lab is a multiuser hypertext system for collaboration which deals with three fundamental issues in simultaneous sharing: access contention, real-time monitoring and real-time communication. Major experiences with HyperBase (collaboration support, data modeling and performance) gained from use both in our lab and in different projects at other research sites are reported. One major lesson learned is that HyperBase can provide powerful support for data sharing among multiple users simultaneously sharing the same environment.

Keywords: Experience, hypertext database, collaboration, data modeling, performance.

1 Introduction

In recent years hypertext (see Conklin [5] and Nielsen [9] for an introduction) has become quite popular and widespread. 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 (the term hypertext will be used to cover both hypertext and hypermedia).

Anyone who has used hypertext systems in an organizational setting will, however, soon recognize the need for hypertext systems that allow several people to work on the same document at the same time, preferably using multiple machines connected through a network. This calls for a system where there is a clear separation between the user interface and the storage of nodes and links. The hyperbase (hypertext database) presented here provides such a storage medium with the low-level facilities necessary to realize a hypertext system for multiple users.

This paper describes the results of the HyperBase project. The focus in the project is on multiuser and collaboration aspects. HyperBase provides some basic mechanisms for collaboration not found in other same generation hyperbases such as HAM [3], GMD-IPSI's HyperBase [13] and HRL's HB1 [11]. HyperBase has been used in a variety of applications. One focused on data modeling aspects [7], another addressed multiuser and collaboration issues [19], another used HyperBase as a storage medium for C++ program fragments [15] and yet another used HyperBase to extend Smalltalk with persistent and shareable objects [1]. Based on first experiences, HyperBase has been developed further to improve the design and deal with some of the shortcomings. The first reliable version was released as free software in July 1991. As of April 1993, more than 300 sites around the world have down-loaded the software; some just to look at it, others to try the provided collaborative hypertext system [19] and still others to use HyperBase to provide database support to different applications.

HyperBase has been well-tested and is in use in our lab. The diversity of applications combined with feedback from other people currently using HyperBase has given valuable knowledge about development of hyperbases. Using HyperBase has shown that it can provide efficient and powerful support for controlled data sharing among people working on a joint project. Although intended to support hypertext applications, HyperBase has proven to be useful as a database for other kinds of applications as well. The HyperBase project is finished and a new hypertext platform, Hyperform [20], is currently being developed based on the experiences with HyperBase. In addition to multiuser and collaboration aspects, Hyperform also addresses other important issues of hyperbases such as data model evolution, access control, version control, query and search support and transaction management, which are necessary to provide the effective database support needed for the next generation hypertext system.

We continue in Section 2 with a description of HyperBase. Section 3 describes two applications using HyperBase. In Section 4, we report our experiences with HyperBase and in Section 5, we show how this approach relates to other research appearing in the hyperbase literature. Section 6 summarizes major points of this paper.

2 HyperBase

The intentions with the design of HyperBase are to construct a general tool suitable as a database for different hypertext (text and other media) applications. HyperBase captures the general idea of hypertext, provides operations on basic entities (nodes and links) and is easy to extend and adapt to a certain field. HyperBase is de-

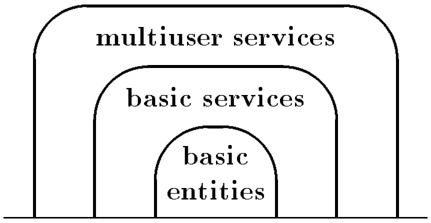

Figure 1: *The different layers of HyperBase.*

signed as a layered system with the following layers (Figure 1): basic entities (file system storing and retrieval), basic services (operations on entities) and multiuser services (locking and event handling).

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: links can only refer to one node, but several nodes can share the same link object.
- Nodes can be the source for many links.
- Both nodes and links can have a set of *predefined* key/value pairs associated with them.
- Versioning on nodes, not on links.

The system keys in links consist of the destination node and various system information. For nodes, the interesting keys consist of a list of outgoing links and the *data field*. Unlike the other keys, the link list and data field are of variable size. HyperBase does not place any restrictions on the data field; it is solely up to the applications to decide 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 cannot predict what kind of information actually will be stored.

Before compiling the HyperBase server the user can specify names of additional keys and the *fixed* maximum size of their values (number of bytes). These keys will be included in all nodes (or links). Examples of predefined keys could be: *owner*, *date*, *type*, *name*, etc.; i.e., static keys which the application programmer can predict. In addition to predefined keys, HyperBase supports runtime attachment of variable size attribute/value pairs to nodes.

Version management in HyperBase is inspired by the RCS model [16]. The RCS model is extended with notions of global and local versions:

- a global version is visible to all users, while
- a local version enables users to work in privacy until the work is ready to be passed on to other users.

Global versions are static, while local versions of a node are deleted when the most recent local version is checked in as a global version. Before a node is created, users have to decide whether or not different versions of the node should be maintained.

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

1. Creating and maintaining structures: Operations to create and delete entities and maintain references between nodes and links.
2. Reading and writing: Operations to read and write predefined keys in entities. HyperBase also provides a function capable of returning a list of all nodes or links.
3. Attribute management: Operations to create, delete, retrieve and modify attributes in nodes.
4. Version management: Operations to create, delete and manipulate global and local versions of nodes.
5. Networking: Connection and disconnection of HyperBase sessions, and a toggle-read-write-buffering function used to speed network communication.

Multiuser services: HyperBase is 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 is implemented as a server (based on the client-server model) allowing only one client at a time to access the shared resource, the hyperdocument. To support collaborative work, two mechanisms are included in the design, the lock and event mechanisms.

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`(all,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).

It is possible for a user to lock an entity while the content is being updated. The lock 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. This enables 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. This fine grained lock 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 contents of locked keys of nodes and links and, in addition, readers can get the login (user) name of the person having locked a specific key in an entity.

2.1 Implementation

In order to simplify the implementation, test and maintenance, HyperBase is separated into functional blocks each performing specified tasks (Figure 2). Each block is

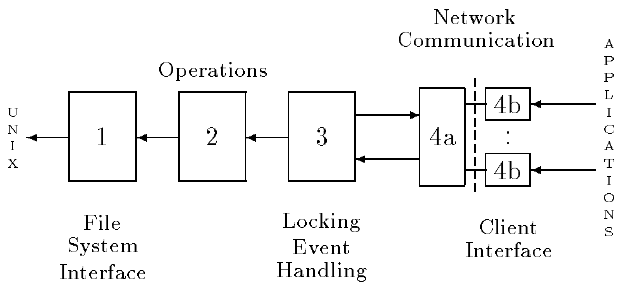

Figure 2: *The four functional blocks of HyperBase.*

implemented as a C++ class. Block 1 through 4a make up the HyperBase server, which is a single process running on the server machine. Block 4b is the client library, which is part of an application program running on each client machine. The server communicates with the client applications through a byte stream protocol (TCP/IP), although it may alternatively be statically linked with an application. Block 1 provides storing and retrieval of the basic entities on the host file system, block 2 provides the basic services, while block 3 and 4a are the multiuser part of HyperBase (Figure 1). The dotted line between block 4a and 4b illustrates the physical network, e.g. an Ethernet.

File system interface: An obvious way to store links and nodes would be to store one entity (node or link) in a file of its own. A preliminary HyperBase experiment [4] showed that opening and closing files are very time-consuming in Unix. Therefore, this solution would place a bottleneck at the 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 nodes and links are deleted.

We have chosen a file representation where nodes and links are stored in three files. One file for nodes, one for links and one for data fields and dynamic allocated attributes of nodes. The reason for keeping the variable size data fields and dynamic allocated attributes separate from the node (Figure 3) is to make all nodes the same size. Data fields and dynamic allocated attributes are

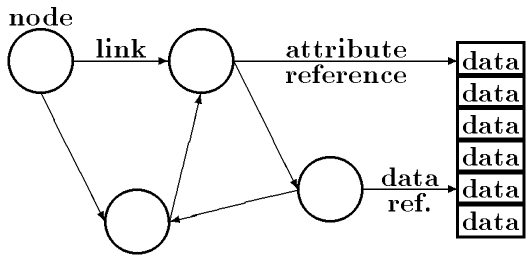

Figure 3: *An illustration of the implementation of the data model, consisting of nodes and links. The contents of data fields and dynamic allocated attributes are stored in a separate file. Nodes contain references to the location of its data field and attributes on the data file.*

divided into blocks of 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 data access and management simple and very efficient.

Before the server is compiled, different parameters can be specified in a *configuration file*, for instance: predefined keys for nodes and links and the block size in the file for the data fields and dynamically allocated attributes. The configuration program determines the actual size of the entities in the three files.

3 Using HyperBase

A variety of applications in our lab have used HyperBase to provide database support [1, 2, 7, 15, 19]. One such application, Emacs HyperText System (EHTS), a multiuser hypertext system for collaboration, is described in Section 3.1. After the release of HyperBase, the system has been used in a number of research projects around the world to provide database support for different application areas such as software engineering systems, collaborative systems, personal information managers and hypertext systems. One of these projects, the EGRET project at the University of Hawaii is briefly described in Section 3.2.

3.1 EHTS

EHTS [19] is a collaborative hypertext authoring system based on HyperBase. EHTS consists of two tools: a multiple window text editor and a graphical browser. EHTS enables a group to collaborate on a shared task. Changes made on shared data by one user are immediately visible to all other members of the group. Group members can communicate in real-time and send asynchronous messages within the EHTS environment, enabling collaboration among members separated by time as well as space.

Trigg et al. [17] describe three fundamental issues in simultaneous sharing: access contention, real-time monitoring and real-time communication. EHTS uses events and fine grained locks to deal with these issues and to provide controlled data sharing among collaborating group members [18]. EHTS also provides contention resolution at the level of attributes in nodes and links and allows any number of users to simultaneously read and display the data field of a given node in a window on the screen. EHTS allows locked objects to be read by any number of users, however permission to make modifications to the data field are restricted to one user at a time. Locks are allocated when the user invokes the editor lock command indicating a change in mode from browse to edit. Locks are deallocated when either the editor unlock command is invoked or the window is closed. All readers are notified as soon as possible that a data field they are accessing may be changed. Readers are provided with four types of modification notices:

Intention. All readers are notified when one person signals intention to modify the data field of the node by obtaining a lock. The node icon in the browser is shown **bold faced**. The readers also get the name of the person, enabling contact through use of the internal talk mechanism (real-time communication). Readers can then subscribe to the event corresponding to when the writer unlocks the data field.

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

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

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

3.2 The EGRET project

The EGRET (Exploratory GRoup work EnvironmenT) project [8] at the University of Hawaii is pursuing a research program designed to investigate evolution in collaborative systems. EGRET is a framework for supporting a fundamental characteristic of exploratory group work: the dynamic, emergent and evolving nature of the structure of information and artifacts produced by exploratory collaborative activities. HyperBase is used as a database server for multiuser support in EGRET.

4 Experiences with HyperBase

This Section describes the major experiences with HyperBase obtained by its extensive use in different applications in our lab and at other research sites. The experiences are grouped into three categories: collaboration support, data modeling and performance. We also describe how HyperBase has been improved in various ways based on early experiences.

4.1 Collaboration support

HyperBase is designed to support collaborative work in a multiuser environment. Collaboration is supported by the event and lock mechanisms:

- The event mechanism enable applications to monitor changes in the shared network of nodes and links.
- The fine grained lock mechanism changes the nature of access contentions, moving the contention from the level of whole entities to the level of single keys in entities.

As described, EHTS uses events and locks to help users solve access contentions. Users are notified when other users perform lock operations on nodes opened in windows on the screen, and when locked nodes of interest to them become unlocked. EHTS uses HyperBase events to monitor changes in the shared network of nodes and links in both the author and browser tool (real-time monitoring). In this way users can profit by the work of other users immediately after the work is carried out. Events and locks are also used to provide real-time communication and support messages between users working within the environment, enabling collaboration between users separated by time as well as space.

It is our experience that these two mechanisms together provide powerful support for controlled data sharing between multiple users working in the same environment. 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 described by Halasz [6].

Locks: In its original form, the lock mechanism was only capable of locking whole links and 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 former problem means, for instance, that when one user is updating 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 nodes or links are locked. Therefore, in order to make better use of the lock facility, it was changed to its present form.

Events: The event mechanism is a very important part of the collaboration support in HyperBase, and to make it even more useful, a more sophisticated way of subscribing to events could be provided. A client should 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 about all operations, except lock operations, performed on all keys in nodes and links. 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.

Hyperform: Both the event and lock mechanism has survived the move into the next generation hyperbase at our lab. In Hyperform these mechanisms are similar to the original ones, since these have been proven to be necessary and sufficient to provide support for collaborative work. In addition, Hyperform supports the event pattern matching feature.

4.2 Data modeling

HyperBase provides a simple data model consisting of nodes and links. It is possible to specify (predefined) keys in both nodes and links before compilation and attach additional attributes/value pairs to nodes at run-time.

More powerful data models can be maintained with these basic constructs. One application [7] introduced an extended data model at the application level based upon the 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 unidirectional links of the underlying data model. Nodes contain a predefined key *type* specifying 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 Unix.

Some applications using HyperBase did not require data model support beyond the basic node-link model [2, 15], while others found the data model in HyperBase too simple [1, 7, 19] and had to combine some of the basic constructs to provide the necessary data model support. It was mainly in two areas that the data model was extended: to provide a higher level of abstraction (a hierarchy of nodes) and to provide bidirectional links.

Node hierarchy: Node hierarchies in HyperBase can be maintained at the application level by the use of a predefined key in the node specifying the node type, e.g. data (text, graphics, pictures, ...) or directory. Discussions continue in the hypertext literature about augmenting the basic node and link based data model with a composite data model object [6]. HAM [3] supports composites (called contexts), which partition the nodes and links in a hypertext graph. GMD-IPSI's HyperBase [13] supports a similar construct. In experiments performed in our lab, the directory abstraction known from file systems was able to support the needs for node hierarchies and, in some cases, the basic node-link model was sufficient.

Links: As mentioned, some of the applications discovered the need for bidirectional links. Bidirectional links can be provided by use of a predefined key in the link keeping track of the source node or by use of two of the provided unidirectional links, but both of 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 applications. A one-to-one link possibility (one source node and one destination node) has shown to be sufficient in most cases. It is also our experience that it should be possible to attach attribute/value pairs to link objects, as can be done to node objects in HyperBase, and it should be possible to invoke programs when following a link.

Query and search: HyperBase has very limited query and search facilities. Clients can retrieve a list of all nodes and links and how they are interconnected. But, for instance, it is not possible to get a list of the nodes and links that are locked by other users already connected to the hypertext system.

We discovered that it is crucial to have a query mechanism which can return a fully updated state of the underlying hyperdocument. When you start-up a graphical browser in a multiuser environment, it is nice if you can see which nodes are locked by other users already connected to the system [19]. Many other query and search possibilities should also be present in a hyperbase:

- Fast keyword, attribute and string search.
- Extraction of specific nodes and links that match a specific description: nodes or links of a certain type, with certain attributes, locked nodes or links, all nodes or links, etc.
- Extraction of specific subgraphs; for instance, nodes of a certain type using links of certain types (pattern matching).

Halasz [6] divides search and query facilities into two broad classes: *content search* and *structure search*. In content search, all nodes and links in the network are considered to be independent entities and are examined individually for a match to the given query. Structure search involves a query language geared toward describing hypertext network structure, and a search engine capable of satisfying the queries expressible in the hypertext query language.

Version management The version management facilities of HyperBase were not present in the first version. It was added later [2] to enable HyperBase to support applications needing to maintain multiple versions of nodes.

This feature is very useful in collaborative working sessions such as software development projects where different programmers work on different parts of the project. Local versions makes it possible to always keep a running version (the global version) of the program and, at the same time, enable programmers to further develop different parts of the program in privacy and commit the changes to the running version whenever the parts are tested and ready.

Hyperform: Hyperform provides object-oriented data modeling facilities. Classes providing basic features (concurrency control, notification control, access control, versioning control and query and search) can be specialized using multiple inheritance to provide database configurations supporting all kinds of data model objects and operations. This way the data model does not place any restrictions on facilities that can be provided by the application.

4.3 Performance

HyperBase is quite efficient. EHTS is capable of following a link and displaying the node in a new window in less than a second on average. The start-up time for EHTS tools differs from 10-30 seconds depending on the load on the network and HyperBase (with several thousands nodes and links stored), when using the buffering feature in the server (described below). Both EHTS tools cache a part of the information in HyperBase to speed operations and monitor changes on shared data.

Networking: In situations where clients made extensive use of the read and write operations (for instance when EHTS tools start-up) communication times turned out to be a problem in the first version of HyperBase. The efficiency of the network protocols was improved by a factor of at least 10 in these situations by introducing a *toggle-read-write-buffering* function in HyperBase.

Before sending several read (and write) requests, the client can tell the server that it does not want any return values yet by sending the toggle command to the server. When this option is set, the server buffers all return values until the client sends the toggle command again. This extension of HyperBase enables clients to send large numbers of read (and write) operations to the server in one package. After sending the last request, clients use the toggle command again and receive all return values in one package.

For instance, EHTS Editor reads the names of all nodes into a cache when it starts-up (to provide completion on node names in its commands). First, it retrieves a list of all node numbers from HyperBase. Then, to reduce network traffic, it sends one large package consisting of the toggle command followed by all the read requests for node names (one for each node present in HyperBase) followed by the toggle command again. Finally, the editor receives all return values in one large package. Thus, with the toggle-read-write-buffering function network communication is reduced to two packages instead of several thousands, depending on the number of nodes in HyperBase.

5 Related work

Other approaches to hyperbase systems include Tektronix' Hypertext Abstract Machine (HAM) [3], GMD-IPSI's HyperBase [13] and Cooperative Hypermedia Servers (CHS) [12], HRL's HB1 [11] and HB2 [10], UNC's Distributed Graph Service (DGS) [14] and RMIT's Hyperion [21]. These hyperbase approaches differ in their intended application areas, but have one thing in common: they are all general-purpose hyperbase servers intended to provide application independent hypertext support. HAM is designed to support hypertext based CAD and CASE applications and the GMD-IPSI approaches are designed to support a hypertext based authoring system. HRL's HB1 and HB2 feature open, extensible architectures focusing on support for inter-application linking, UNC's DGS services a hypertext system for structuring ideas in collaborative working sessions and RMIT's Hyperion concentrates on efficient management (storage and retrieval) of large volumes of text.

HyperBase is designed to support collaboration among its users and was the first hyperbase system to include a general event mechanism and provide user-controlled locks. Other hyperbase systems appearing at the same time as HyperBase such as HAM, GMD-IPSI's HyperBase, Hyperion, and HB1 does not offers support for collaboration beyond the simple mechanism of data sharing known from client-server architectures. These approaches does not provide general event and lock mechanisms to support CSCW applications.

The successors of HB1 (HB2) and GMD-IPSI's HyperBase (CHS) provide event and lock mechanisms operating at the object-level. The event and lock mechanisms of HyperBase and its successor Hyperform provide attribute-level event and lock mechanisms enabling a finer grained collaboration.

6 Summary

In this paper, we have described the architecture and experiences with HyperBase, a hypertext database supporting collaborative work. The extensive use of HyperBase in different applications has shown that some applications need data model support beyond support directly available in HyperBase and have to combine basic constructs into more powerful data models. Other applications need more powerful query and search facilities than provided by HyperBase.

More important, using HyperBase has also proven that it can provide efficient and powerful support for controlled data sharing among people working on a joint project. Although intended for hypertext applications, HyperBase has proven to be useful as a database for other kinds of applications as well.

The emphasis in this (our first) generation hyperbase system was on collaboration support. The event and lock mechanisms have proven to be necessary and sufficient to provide support for collaborative work. This work is the first step towards providing support for the social interactions involved in collaborative use of a shared network, as described by Halasz [6]. Experiences gained from this work have been used to identify several important issues to be addressed in future hyperbase research.

Present status: The HyperBase project is finished. HyperBase development has taken approximately 2.5 man years of effort (400 KByte source code). EHTS development has taken approximately 1.5 man years (200 KByte source code). HyperBase and EHTS are available via anonymous ftp from iesd.auc.dk in the hypertext directory.

Based on the experiences with the HyperBase project, an approach to flexible hyperbase support predicated on the notion of extensibility is currently being developed [20]. The extensible hypertext platform (Hyperform) implements basic hyperbase services that can be tailored to provide specialized hyperbase support. Hyperform is based on an internal computational engine that provides an object-oriented extension language which allows new data model objects and operations to be added at runtime. Hyperform has a number of built-in classes to provide basic hyperbase features such as concurrency control, notification control (events), access control, version control and search and query. Each of these classes can be specialized using multiple inheritance to form virtually any type of hyperbase support needed in next generation hypertext systems. The first version of Hyperform is implemented and operational in Unix environments. Early experiences indicate that the Hyperform approach greatly reduces the effort required to provide high quality customized hyperbase support to dynamic, open and distributed hypertext applications.

Acknowledgments: I wish to acknowledge Claus Bo Nielsen, who took part in both the development of EHTS and HyperBase, and Carsten Ruseng Jakobsen, Finn Sølvsten, Per Magnus Petersen, Hans Mejdahl Jeppesen, Poul Larsen, Morten Tolbøl and Levi Christiansen, who also were part of the HyperBase team.

References

[1] P. Abrahamsen et al. Introduction of graph-grammars in programming environments. Master's project, Aalborg University, June 1990.

[2] T. Andersen et al. HT-KSVK: A programming environment with support for version control and configuration management. Master's project, Aalborg University, Dec. 1990. (in Danish).

[3] B. Campbell and J. M. Goodman. HAM: A general purpose hypertext abstract machine. *Communications of the ACM*, 31(7):856–861, July 1988.

[4] J. P. Christensen. HyperBase: A server for hypertext applications. Master's thesis, Aalborg University, Aug. 1988.

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

[6] 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.

[7] C. R. Jacobsen, H. M. Jeppesen, and P. M. Petersen. Urd: Using the hypertext concept to project support. Master's thesis, Aalborg University, June 1990.

[8] P. Johnson. Supporting exploratory CSCW with the EGRET framework. In *ACM CSCW '92 Proceedings*, pages 298–305, Toronto, Canada, Nov. 1992.

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

[10] J. L. Schnase. *HB2: A hyperbase management system for open, distributed hypermedia system architectures*. PhD thesis, Texas A&M University, 1992.

[11] J. L. Schnase, J. J. Leggett, D. L. Hicks, P. J. Nürnberg, and J. A. Sanchez. Design and implementation of the HB1 hyperbase management system. *EP-odd*, 6(2), 1993.

[12] H. A. Schütt and J. M. Haake. Server support for cooperative hypermedia systems. In *Proceedings of Hypermedia '93*, Zurich, Switzerland, Mar. 1993.

[13] H. A. Schütt and N. A. Streitz. HyperBase: A hypermedia engine based on a relational database management system. In *ECHT '90 Proceedings*, pages 95–108, Versailles, France, Nov. 1990.

[14] J. B. Smith and F. D. Smith. ABC: A hypermedia system for artifact-based collaboration. In *Hypertext '91 Proceedings*, pages 179–192, San Antonio, Texas, Dec. 1991.

[15] F. Sølvsten. A visual interconnection language. Master's thesis, Aalborg University, June 1990.

[16] W. F. Tichy. RCS: A system for version control. *Software - Practice and Experience*, 15(7):637–654, July 1985.

[17] R. H. Trigg, L. A. Suchman, and F. G. Halasz. Supporting collaboration in NoteCards. In *CSCW '86 Proceedings*, pages 147–153, Austin, Texas, Dec. 1986.

[18] U. K. Wiil. Using events as support for data sharing in collaborative work. In *International Workshop on CSCW*, pages 162–176, Berlin, Germany, Apr. 1991.

[19] U. K. Wiil. Issues in the design of EHTS: A multiuser hypertext system for collaboration. In *HICSS-25 Proceedings*, pages 629–639, Kauai, Hawaii, Jan. 1992.

[20] U. K. Wiil and J. J. Leggett. Hyperform: Using extensibility to develop dynamic, open and distributed hypertext systems. In *ACM ECHT '92 Proceedings*, pages 251–261, Milano, Italy, Dec. 1992.

[21] J. Zobel, R. Wilkinson, J. Thom, E. Mackie, R. Sacks-Davis, A. Kent, and M. Fuller. An architecture for hyperbase systems. In *Proceedings of the 1st Australian Multi-media Communications, Applications & Technology Workshop*, pages 152–161, 1991.