

For:

***Electronic Publishing –
Origination, Dissemination and Design***

Send correspondence to:

John L. Schnase, PhD
Advanced Technology Group
School of Medicine Library and
Biomedical Communications Center
Washington University School of Medicine
660 South Euclid Avenue (Box 8132)
St. Louis, Missouri 63110

Phone: 314-362-3117

FAX: 314-362-0190

Email: schnase@medicine.wustl.edu

Design and Implementation of the HB1 Hyperbase Management System

*John L. Schnase, John J. Leggett, David L. Hicks,
Peter J. Nuernberg, and J. Alfredo Sánchez*

Design and Implementation of the HB1 Hyperbase Management System

JOHN L. SCHNASE

*Advanced Technology Group
School of Medicine Library and
Biomedical Communications Center
Washington University School of Medicine
660 South Euclid Avenue (Box 8132)
St. Louis, Missouri 63110
USA*

JOHN J. LEGGETT, DAVID L. HICKS, PETER J.
NUERNBERG, AND J. ALFREDO SÁNCHEZ

*Hypermedia Research Laboratory
Department of Computer Science
Texas A&M University
College Station, Texas 77843
USA*

SUMMARY

Hypermedia systems manage interconnected information residing within a potentially wide range of data types, including text, graphics, animations, and digitized sound and images. Effective database support for hypermedia-based computing environments is essential. In order to be effective, this support must provide a variety of capabilities that are not offered by the current generation of database management systems. We report on a prototypic system called HB1 that has been designed to meet the storage needs of advanced hypermedia system architectures. HB1 is referred to as a hyperbase management system (HBMS) because it stores and manipulates information and the connectivity data that link information together to form hypermedia.

HB1 is composed of three subsystems: the Object Manager (OM), Association Set Manager (ASM), and Storage Manager (SM). OM and ASM are both server processes accessible to distributed client processes via IPC interfaces. OM is an object server. ASM manages structural data applicable to the objects within OM's repository that are involved in hypermedia connections. Physical storage is managed by SM which, in this implementation, is a semantic network database management system. HB1 instantiates a conceptual model of hypermedia that is distinctly computational, has a strong notion of anchor and link, and abstracts information, behavior, and structure from hypermedia. It has been used as a backend for an open, object-based hypermedia system that implements distributed, inter-application linking. HB1 is proving to be an effective vehicle for research on HBMS organization.

KEY WORDS

Hyperbase management system

Hypermedia

Hypertext

Open hypermedia system architecture

Inter-application linking

Semantic object-oriented database management system

INTRODUCTION

The sophistication and effectiveness of hypermedia-based computing environments has steadily increased over the past several years. Given the anticipated improvements in hardware and software technologies, it is likely that this trend will continue. We expect the next generation of hypermedia system architectures to be network- and object-based. They will accommodate multiple media types on high-performance hardware and reflect an open and extensible design philosophy. Advanced hypermedia environments will allow multiple users to work cooperatively with extremely large volumes of richly textured, complex, dynamic, and arbitrarily structured information. Hypermedia will undoubtedly play an important role in a variety of information-intensive settings, such as electronic publishing, personal knowledge management, and digital libraries.

For this future to be realized, however, effective database support for advanced hypermedia environments is essential. In order to be effective, this support must provide a variety of capabilities that are not offered by the current generation of database management systems. In addition to the usual requirement for permanence of data, controlled sharing, and backup and recovery, data management facilities must be capable of modeling complex interrelationships and providing direct support for novel data types. They must also be able to handle lengthy transactions, transactions that are unique to hypermedia, massive distribution of functionality, extensibility, and versioning of both the data and structure of hypermedia.

HB1 is a prototype database management system (DBMS) designed to meet the storage needs of advanced hypermedia system architectures. We refer to HB1 as a *hyperbase management system* (HBMS) because it supports not only the storage and manipulation of information, but the storage and manipulation of connectivity data as well. HB1 was designed with a very specific goal in mind. It is intended to support an open, object-based system architecture for distributed, inter-application linking. A hypermedia system, called System Prototype 1 (SP1), based on such an architecture has been developed in the Hypermedia Research Laboratory at Texas A&M University, and HB1 is its backend.

As shown in Figure 1, HB1 is composed of three subsystems: the Object Manager (OM), the Association Set Manager (ASM), and the Storage Manager (SM). OM manages a shared object repository and ASM manages hypermedia connections that exist among OM objects. SM maps HB1's data model into physical storage. In the current implementation, SM is a semantic network database management system. Heterogeneous distribution of OM and ASM functionality is achieved by a client/server model using X Window System's interprocess communication facilities.

In this paper, we report on the design and implementation of the HB1 hyperbase management system and describe its operation within the SP1 framework. We also summarize our experiences using these systems and identify the important issues that must be addressed by future research in this emerging area.

HYPERMEDIA CONCEPTS

The concepts underlying hypermedia have been described in several places [1–7]. Simply put, hypermedia systems allow networks of linked information to be created and manipulated. For example, in Figure 2, a portion of the digitized image of a bird in a multimedia document has been linked to a paragraph of related information in a text object. A typical system would provide the necessary mechanisms to display and edit images and text, author links, and highlight information that is linked. Subsequent access to connected information is then obtained through an active process of navigation. With the aid of a high-resolution, bit-mapped display and a pointing device, users move through hypermedia by traversing links, or browsing, in addition to traditional query mechanisms. It is this ability to *author* and *browse* arbitrary connections among information that distinguishes hypermedia systems from other types of multimedia information systems.

Beyond these basic notions, the abstractions, interfaces, functionality, and terminology presented by the current generation of hypermedia systems differ widely. Despite the variation in existing hypermedia systems, most have one critical architectural feature in common: they are monolithic, stand-alone applications. The behaviors that make them distinctive are encapsulated within each system. The hypermedia structures that they manipulate cannot be shared, and the information they manage is relatively isolated from other applications. In short, most existing hypermedia systems lack the architectural framework required for integration into computing environments of the future.

CONCEPTUAL MODEL UNDERLYING HB1

Our work focuses on software architectures to support advanced hypermedia functionality. In particular, we are interested in approaches that accommodate openness, extensibility, distribution, and inter-application linking. Over the past several years, a perspective on hypermedia has evolved that fulfills these requirements.

Figure 3 shows the conceptual model of hypermedia upon which HB1 is based. The model consists of six principle elements, the first three of which are: applications, components, and persistent selections. *Applications* are simply programs. *Components* are the data or information manipulated by applications. For example, text editor applications typically manipulate ASCII components. *Persistent selections* are selections within components that, in contrast to the usual implementation of selection, persist between application sessions and can be accessed at a later time. Applications implement this functionality by maintaining persistent selection data structures for components (Figure 4). Within this conceptual framework, applications, components, and persistent selections embody the information managed by a system, and the network structure of hypermedia results from the connections forged among persistent selections.

The remaining three components of the model, anchors, links, and associations, are essential for hypermedia functionality. *Anchors* and *links* are processes in the operating systems sense of the word. They are programs or program components that can be independently scheduled by the underlying operating system in order to accomplish some task. In this case, they implement the behaviors that characterize hypermedia, such as customized views or various traversal behaviors. As shown in Figure 3, anchors are associated with persistent selections, and links are associated with anchors, thereby completing connections among persistent selections. The relationships between these elements, depicted as arcs in Figure 3, are *associations*. Associations, in contrast to anchors and links, do not implement behavior. They are structural entities – collections of identifiers that tie elements together.

A general hypermedia system architecture implemented along these lines allows the integration of diverse applications, anchors, and links under a common hypermedia model. Specifically, non-monolithic, inter-application linking can be realized by moving hypermedia connectivity data (associations) and hypermedia functionality (anchors and links) into a distinct "link services" subsystem of a computing environment (Figure 3). Hypermedia services can then be provided to participating applications through interprocess communication (IPC).

Inter-application linking is a powerful means of achieving application-level extensibility within a hypermedia system. However, HB1's data model accommodates extensibility in other ways as well. New hypermedia functionality – that is, new anchor and link processes – can be incorporated at any time without affecting the underlying structure of hypermedia. In addition, massive distribution of functionality is accommodated since application, anchor, and link processes, as well as processes affecting associations, can conceptually reside anywhere within the IPC domain of the architecture.

It should be noted that many of the central concepts found in current monolithic hypermedia systems do not carry over to a non-monolithic, link services world. For example, the concept of node essentially goes away, and the semantics of authoring and browsing become the shared responsibility of applications and the link services subsystem. Connections, in a sense, exist only from information to information. The traditional notion of anchor corresponds most closely to our persistent selection, which is managed entirely by applications rather than link services.

HB1 DESIGN AND IMPLEMENTATION

As indicated earlier, the overall goal for HB1 is to provide effective data management support for advanced hypermedia environments. In attaining this goal, HB1 follows the approach of providing a lightweight, broadly applicable hyperbase service within an open, distributed architectural setting [8]. The basic assumption is that carefully defined generality at the hyperbase level can facilitate flexibility, tailorability, and extensibility system wide. HB1's conceptual design and its prototypic implementation are an attempt to produce a framework that allows this hypothesis to be examined.

HB1's architecture is specifically tuned to the conceptual model for hypermedia described in the previous section. This model leads rather directly to a functional decomposition that distinguishes between structure and object management and between management at an abstract, or logical, level and management at a physical level. In order to support these distinctions, HB1 is organized around three domains: an abstract object domain, an abstract structure domain, and a physical domain. It is the responsibility of the object domain and the structure domain to implement HB1's data model; the physical domain is responsible for managing physical storage. Clients interface directly with the object and structure domains, while these two domains interface internally with the physical domain.

This philosophical organization is embodied in the HB1 system architecture shown in Figure 1. HB1 is a centralized, single-user HBMS consisting of three subsystems: the Object Manager (OM), Association Set Manager (ASM), and the Storage Manager (SM). OM implements the notion of a large, shared repository of simple, unstructured objects. ASM provides persistent and sharable storage for the connectivity data that link information together to form hypermedia. Together OM and ASM implement HB1's data model, which abstracts inter-object connectivity, behaviors, and information from hypermedia. SM maps HB1's data model into physical storage. Distribution of OM and ASM functionality across a range of platforms is achieved by a client/server model using interprocess communication facilities. We begin a more detailed discussion of HB1's architecture by looking at the Storage Manager.

Storage Manager (SM)

HB1's Storage Manager is a prototypic semantic network database management system called Saberel¹, now under development by IBM. Saberel was chosen because it allows unstructured objects to be stored, manipulated, and accessed by way of conveniently defined inter-object relationships. Traditionally, Saberel has been used for CAD and VLSI design applications. We have extended and modified this storage system to better support distribution, large object size, and hypermedia transactions. We elaborate on our experiences with the semantic modeling of hypermedia associations in [9]. Before describing Saberel further, we quickly review the major distinctions between semantic and traditional database approaches.

Semantic Database Models. Traditional database management approaches tend to emphasize information structures that promote efficient storage and retrieval of fixed-size information. For example, the relational, hierarchical, and network models use a simple record-based format. These approaches generally lack direct support for relationships, data abstraction, inheritance, constraints, and unstructured objects [10]. In the past several years, however, greater consideration has been given to the user's perception of data rather than its physical representation. This has resulted in a requirement for richer, more expressive data structuring capabilities and has led to a renewed interest in semantic data models. Semantic models attempt to provide more powerful abstractions for structuring complex objects than are supported by traditional models. Although a detailed consideration of semantic modeling is beyond the scope of this paper, excellent surveys can be found in [11, 12].

Saberel. The Saberel semantic network database management system is representative of a family of "binary" semantic models, all of which represent data using two primary constructions: entity sets and binary relationships. As we will show, schemas of these models generally consist of labeled nodes for entity sets and labeled arcs corresponding to binary relationships between them.

Saberel supports atomic objects, called *subjects*. Subjects can carry a printable name and a string of data of arbitrary size, referred to as a *user managed string* or *UMS*. Virtually all of Saberel's higher-level abstractions are derived from subjects. For example, entity sets, or *categories*, are themselves subjects, as are relationship names. As a result, the semantic distinction between objects, types, and attributes is not directly enforced by Saberel's modeling primitives. Each binary relation is viewed as an inverse pair of single-valued functions. Saberel subjects are stored in a *repository* and partitioned among any number of lockable *areas*. Repositories and areas are

¹ Saberel is not a product.

implemented as UNIX files, and, while they assist the user in organizing a Saberel database, they do not participate as abstractions in Saberel's semantic model.

Saberel does not support type constructors, ISA relationships, or derived schema components and enforces few combining restrictions or integrity constraints. The philosophical approach of Saberel, and similar models, is to provide a small, universal set of constructs that can be used to build more powerful structures. These models tend to be "minimalist" in the sense that they require the database designer to understand fewer constructs. One apparent benefit to such an approach is the relative ease with which additional behavioral layers can be applied to a structural database of information.

Saberel has two programming interfaces: a low-level procedural interface to the library routines that implement a Saberel database and a declarative, high-level language that can be precompiled into C applications. Existing Saberel applications have required little sharing of data. Consequently, the system has not supported distribution or transaction management at the database level. One of our objectives was to extend Saberel functionality by creating a network-accessible server interface to the system. HB1's Object Manager and Association Set Manager subsystems are the first two such interfaces we have developed.

Object Manager (OM)

HB1's Object Manager is a server process that provides persistent and sharable storage for applications. As shown in Figure 1, OM is built on top of HB1's Storage Manager and provides an IPC interface to clients. OM, in effect, manages a Saberel repository of unstructured objects.

OM Data Model. HB1 objects are simple. They are arbitrary size byte strings having unique identity and optional type and name attributes. We avoided implementing objects with execution semantics because we wanted the server to be lightweight and general [8]. HB1 objects have no class structure, methods, or inheritance, although these mechanisms could readily be built on top of OM. This simple view of object is one that is shared by a class of data managers referred to as *persistent object stores* [10]. As described later, persistent object stores are sometimes used for storage management within more complete object-oriented database systems and, in many respects, can be thought of as extensions of virtual memory.

Each object in the database has a unique 32-bit identifier (OID). OIDs are sequentially allocated by the server upon request from clients creating objects. The OIDs of deleted objects are not reused since references to the objects may remain in the database. Since HB1 is a centralized server, many of the naming problems that arise in decentralized distributed systems are avoided. As described below, most client interactions with OM are based on OIDs. OM allows clients to have direct access to OIDs. The OIDs presented to clients are the same OIDs used internally by OM and persist across sessions; they are not special "handles" that cannot be examined further or stored outside the database. This makes it possible to extend OM's policies and mechanisms through additional layers interspersed between HB1 and the client space.

Applications use objects by copying them from the server into the virtual memory space of their own processes where they are manipulated locally and written back to the server. OM implements the following basic backend operations: *Create*, *Delete*, *Retrieve*, and *Store*. The *Create* operation generates a new OID. *Delete*, *Retrieve*, and *Store* operations are based on input OIDs. OM also implements attribute operations such as *Name* and *Type* and a *Resolve* operation that allows searches of an OM repository based on object attributes.

Semantic Modeling of OM Data. Figure 5 shows how Object Manager data are modeled by the Storage Manager. Three subject categories are explicitly represented in the database: OBJECT, NAME, and TYPE. The chunks of memory that constitute objects in our system are members of the OBJECT category. Ovals in Figure 5 represent Saberel subjects, and, in this instance schema, we show the binary relationships that exist between example subjects in each category.

Association Set Manager (ASM)

The Association Set Manager is a server process that provides persistent and sharable storage for associations and manages their run-time representation in support of a link services subsystem. To clarify what is meant by this, we must first explain ASM's data model. ASM manages the structural or connectivity data applicable to those objects within OM's repository that are involved in hypermedia associations. In a sense, it manages the static, backing store representation of hypermedia. ASM is built on top of HB1's Storage Manager and provides its own IPC interface to clients (Figure 1). For more details, see [13].

ASM Data Model. In Figure 6, we present a simple example of an association. Here, a hypermedia connection exists between persistent selections in two different components, each of which is handled by a different application. The persistent selections, components, applications,

anchors, and links all have unique IDs. PSIDs are generated and managed by applications, and the scope of their uniqueness extends only to the components in which they reside. CompIDs, AppIDs, AnchorIDs, and LinkIDs correspond to the OIDs for these objects on backing store. These identifiers are generated and maintained by HB1's Object Manager, and the objects are stored in the Storage Manager's repository. As a result, these identifiers are unique across the world served by HB1.

To represent the structural information involved in such a connection, ASM's data model defines three collections of IDs: Bridges, Sides, and Associations (see Figure 6). A *Bridge* consists of a {PSID, CompID, AppID} triple and imparts global uniqueness to a PSID. We use the term bridge because the triple spans the distinct address domains managed by HB1 and individual applications. An AnchorID associated with one or more Bridge triples forms a *Side*. Finally, the connection is completed by a LinkID being associated with two or more Sides. This is called an *Association*. The *set of associations* available at a given moment defines a context.

Note that this definition allows connections to occur at the level of anchors and at the level of links. For example, Figure 7 shows a more complex association having three sides, two of which contain multiple bridges. It is in the definition of ASM's data model that structure (Bridges, Sides, and Associations) is abstracted from the behavioral elements (Anchors and Links) and information (Persistent Selections, Components, and Applications) of hypermedia.

With respect to structure manipulation, the major abstractions presented to clients by ASM are sides and associations. The semantics of structure manipulation are provided through a set of basic operations that include: *AttachAnchor*, *AttachLink*, *DetachAnchor*, *DetachLink*, and *FollowAssociation*. *Attach-* and *DetachAnchor* allow sides to be created and deleted. Likewise, *Attach-* and *DetachLink* allow associations to be created and deleted. *FollowAssociation* is a query that returns the IDs of elements ({PSID, CompID, AppID} triples, AnchorIDs, LinkIDs) reachable in an association given one or more input IDs. This operation is primarily in support of browsing semantics and would generally compute reachability based on an input {PSID, CompID, AppID} triple that had been selected at the application level as the starting point of a navigation operation.

Semantic Modeling of ASM Data. Figure 8 is a semantic schema showing how Association Set Manager data are modeled by the Storage Manager. It is important to understand that the OM and ASM operate on the same Saberel repository, although OM objects reside in a different area than ASM's data. Figure 8 is a complete instance schema for the simple association example of Figure 6. The shaded area in Figure 8 corresponds to the OM schema in Figure 5, and the subjects and

relations lying outside the shaded region are the part of the schema manipulated by ASM. This example shows graphically how information, structure, and behavior have been separated in HB1.

The OBJECT category in Figure 8 contains applications, components, links and anchors as well as other objects handled by OM. In addition, ASM uses the categories PSID, BRIDGE, SIDE, and ASSOCIATION. ASM stores any PSIDs involved in connections in the PSID category. Subjects in the BRIDGE category tie together objects identified by {PsID, CompID, AppID} triples. Subjects in the SIDE category reference BRIDGES that have been grouped together by anchor attachment, and ASSOCIATION subjects join SIDES that have been grouped by link attachment. It is important to notice that the Storage Manager, by virtue of its underlying semantic database, directly represents associations among linked objects in our system. Or, said another way, the collection of IDs that define BRIDGE, SIDE, and ASSOCIATION entities need not be repeated in the database because these Saberel subjects have relationships that point directly to the objects involved. The connectivity information maintained by the Association Set Manager, in effect, overlays the objects maintained by the Object Manager.

HB1 / SP1 Operation

As indicated earlier, HB1 provides data management support for a hypermedia system called System Prototype 1 (SP1). SP1 implements an object-based system architecture for distributed, inter-application linking. Basically, the system allows hypermedia connections to be forged among applications that are able to participate in a distributed *link services* protocol. The functional capabilities of the system are realized by client/server relationships among several software components: Participating Applications, the Link Services Manager, and HB1's Object Manager and Association Set Manager servers. The overall architectural organization of SP1 is shown in Figure 9.

The Link Services Manager (LSM) is a server process that provides run-time support for inter-application linking. LSM coordinates the interprocess communication required to implement the hypermedia functionality of Participating Applications. These activities include the attachment and detachment of anchors and links, conveying requests to the applications that they display anchored or linked persistent selections, and browsing operations. LSM services, in a sense, instantiate the dynamic, real-time manifestation of hypermedia with which a user or user processes (proxies) may interact.

Since inter-application linking is mediated by the LSM, applications that wish to participate in link services must be able to interact with this manager. Such applications are referred to as Participating Applications (Apps) in Figure 9. Participation requires that Apps be able to handle persistent selection and respond appropriately to messages from the LSM. For example, Apps are responsible for providing mechanisms to create, delete, and display persistent selections and manipulate persistent selection data structures. It is also assumed that the Apps' interfaces will allow users to reference persistent selections in such a way that their identities can be determined by the applications. To date, text, graphics, and bitmap editors have been developed as participating applications in the architecture. The Athena Text Widget has also been modified in order to make X Window System's Xedit a participating application [14].

The details of SP1's overall behavior is beyond the scope of this paper. The basic notion, however, is that hypermedia connections can be authored and browsed through coordinated events between Participating Applications and the Link Services Manager. As shown in Figure 9, Participating Applications communicate with HB1's Object Manager and the LSM, and the LSM communicates with HB1's OM and ASM servers. The messages that pass among the various components of the architecture essentially consist of type information and IDs (Table 1).

Implementation

The software components in HB1 and SP1 are written in C and port to UNIX variants on a range of platforms. LSM is a decentralized server and can be allocated to any physical processor in our LAN, as can Participating Application processes. HB1's Object Manager and Storage Manager are implemented as independent processes, however, they are centralized to a dedicated node on which the Storage Manager's Saberel repository resides.

In order to achieve heterogeneous distribution across a wide range of platforms, a suite of message passing IPC protocols were implemented using X Window System interclient communication facilities. It is possible for client processes on other networked workstations to access LSM and HB1 servers by using these protocols. All messaging is bidirectional. To facilitate construction of the various client and server modules, we developed the *X Link Services (Xlt) Toolkit*. As shown in Figure 9, processes that wish to communicate with one another bind liaison procedures from the Xlt Toolkit into their virtual memory space. Clients then call on these routines in order to communicate with servers, and servers, in turn, use Xlt routines to reply. These liaison procedures provide functional interfaces that abstract from details of the underlying communication protocol. The OM and ASM servers also bind to the libraries that implement the Saberel system.

Communication between the HB1 servers and client applications follows a mailbox model in which "in" and "out" mailboxes are owned by each client and shared by the servers. A client process sends a request message to the server by way of its out mailbox. Reply messages from the server are deposited into a client's in mailbox and subsequently read by the receiving client. While this pattern of message passing easily accommodates asynchronous communication, in this implementation clients wait for all server replies; hence, at the application level, all operations appear synchronous.

When an application-level process registers for either OM or ASM services, it provides the server with the window ID of some (possibly invisible) X window which it owns. X *properties* that function as mailboxes for the communication protocols are then associated with the application's window. X properties are a mechanism for interprocess communication within the X Window System [15]. The basic idea is that an X application can create and delete properties that are conceptually associated with specific application windows. They can then receive notification about and respond to events affecting those properties. In the HB1 prototype, the event of interest to servers and applications alike is the movement of data into and out of an X property.

Figure 10 shows conceptually how X properties have been associated with application windows in order for OM and ASM data to be served to the applications. In this case, the properties are named `_XLT_APP_OM` and `_XLT_APP_ASM` (the client's out mailboxes) and `_XLT_OM_APP` and `_XLT_ASM_APP` (the client's in mailbox). OM and ASM servers solicit *property notify* events on the `_XLT_APP_OM` and `_XLT_APP_ASM` properties on each client application's window. An X *event loop* within the server processes models the Ready and Active states of the server. When an X event affecting these properties is detected by the servers, the state transition from Ready to Active occurs. In the Active state, servers perform operations that implement their functionality and then return to their Ready states.

The Xlt Toolkit client liaison routines called by an application cause messages to be written to the application's X properties. These writes are detected by HB1's servers as *property notify* events. The servers respond by reading a property's data, thereby causing the client's message to be "sent" to the server. Server replies are written to an application's `_XLT_OM_APP` or `_XLT_ASM_APP` property from which it is similarly read by the application. The main X event loop in the servers handle client requests in FCFS order. Since servers respond completely to each client request in the Active state before returning to the Ready state, atomicity and freedom from deadlock or starvation is assured on these operations.

DISCUSSION

HB1 and SP1 represent significant departures from the way most existing hypermedia systems are conceived and implemented. In this section, we take a closer look at some of the features that make our approach distinctive.

HB1 Data Model

HB1's data model can be distinguished from current hypermedia models by three interrelated characteristics: (1) it defines a distinctly computational view of hypermedia, (2) it has a strong notion of both anchor and link, and (3) it abstracts information, behavior, and structure from hypermedia. We comment on anchors first.

Only a few existing systems incorporate a notion of anchor, and Intermedia typifies the most commonly held view [16, 17]. In Intermedia, anchors are essentially data structures that specify the endpoint of a link. While these endpoints can be many things, including a span of text, coordinate locations, a series of animation or video frames, or even bytes in a sound buffer, anchors in Intermedia are essentially addresses — what we would refer to as structural entities. Several recently proposed data models include a similar concept of anchor. The Dexter hypertext reference model [18] and the data models of Lange [19] and Afrati and Koutras [20] propose anchors very much like those found in Intermedia. Anchors, as perceived by most people today, correspond closely to persistent selections in HB1's model.

In contrast, anchors in our model embody behavioral rather than structural characteristics. "Attaching an anchor" means that a behavior is associated with a given persistent selection. The rationale for a strong, computational view of anchor is simple: we believe anchors are an appropriate locus for several classes of behaviors that are only indirectly related to traversal. Anchors can cooperate with application processes to customize views, filter information, coordinate searches, or even monitor events that may happen in the future. Since they are arbitrary processes, anchors can do simple tasks or be as complex as needed. They can, for example, be interfaces to an information retrieval system, expert system, or database management system. In fact, anchors could actually *be* systems such as these.

Links in our model are also behavioral entities. In this case, however, we see links being primarily responsible for behaviors related to the traversal of associations. Links, for example, can cooperate with anchor processes to create "destination" processes, disambiguate multiple alternative "destinations", or perform other activities generally associated with navigation.

HB1's data model abstracts structure, information, and behavior from hypermedia. Until now, advanced hypermedia system designs have tended to focus on abstracting only structure and information. They have done so in order to achieve architectural advantages such as the ability to implement contexts. Since these designs have typically been translated into monolithic applications, behavioral abstraction has largely been ignored, and the behaviors that characterize a given system become encapsulated within the systems themselves. Intermedia, for example, separates connectivity data from information, but traversal behaviors are implemented within Intermedia's applications. Among emerging data models, those of the Dexter group [18], Lange [19], Afrati and Koutras [20], Schütt and Streitz [21], and Wiil [22] all abstract structure from information but do not explicitly abstract behavior.

There is an important rationale for removing addressing data from the behavioral characteristics of anchors and links: doing so promotes extensibility of hypermedia functionality through an open systems approach. In addition, it supports the massive distribution of hypermedia functionality required of network-optimized computing environments. The abstractions provided by most existing hypermedia data models all too often predispose system designers toward monolithic implementations. As a consequence, the global structure of hypermedia tends to be relatively inaccessible and behaviors are not easily modified or extended [9]. Our data model is intended to encourage implementation approaches that avoid encapsulating structural information with structure-independent behaviors. It also facilitates the implementation of operations that are essentially structural, and a number of these operations – such as computing virtual structures, constructing graphical overviews, and performing structural searches – are recognized to be crucial next-generation features [23].

Finally, a comment on our computational bias. When anchors and links become arbitrary processes in a hypermedia system architecture the potential of these systems is enormously increased. It moves hypermedia systems from the class of "interesting" applications toward operating systems and other software in the class of systems support. In fact, we see computation as ultimately allowing hypermedia systems to achieve their potential as a new operating paradigm for computing.

HB1 Architecture

HB1's overall architecture advances the notion of a tailored hyperbase service. Due largely to its underlying model, HB1's internal organization is partitioned into object and structure domains. Such partitioning allows each subsystem to be specifically optimized to provide effective services over information and behavioral objects as well as the structural elements of hypermedia.

An important design issue in this project was determining the appropriate place to draw the line between application functionality and HBMS functionality. In our view, applications are best suited to manipulate the abstractions they implement. This includes "internal" abstractions such as persistent selection. In HB1, functionality was partitioned in a way that promotes system-level integration of backend functionality at the expense of being able to perform certain types of operations at the hyperbase level. We see this approach as being more general.

HB1's simple notion of object, immutable and externally accessible object identifiers, and a simple client/server interface make the HBMS lightweight and broadly applicable. For example, it would be relatively easy to incorporate new policies and mechanisms as layers that are conceptually applied on top of HB1's basic services. Extensibility at the hyperbase level is facilitated by HB1's modular organization and simple inter-module interfaces. The use of the X Window System for IPC enables access to HB1 from a variety of physical platforms and operating systems. Few approaches to distribution encompass a wider range of possibilities than X.

Perhaps most important, HB1's overall architecture is scalable. New forms of physical storage can be incorporated into the Storage Manager component of the architecture at any time, and algorithms for handling widely distributed, heterogeneous data repositories can be incorporated into the Object Manager and Association Set Manager servers. HB1's process structure could even be tailored for use on specialized multiprocessor database – or hyperbase – computers. In summary, HB1's architecture is designed to assimilate the emerging software and hardware innovations that will ultimately allow HBMSs to manage the data capacities envisioned for future hypermedia systems.

Experiences

Some of our most important experiences relate to the development of Participating Applications and use of the Xlt Toolkit. As indicated earlier, three SP1 Apps have been developed. V and PStext Widget are text editor applications, and HyperDraw is a graphics and bitmap editor.

V is an X Windows-based text editor modeled after vi. Like the other Participating Applications, V implements a persistent selection capability, is able to participate in SP1's link services architecture, uses objects served by HB1's Object Manager, and was developed using the Xlt Toolkit. V consists of approximately 6000 lines of C code. *HyperDraw* is an object-oriented, X Windows-based graphics and bitmap editor. This application provided an important opportunity to work with large objects. HyperDraw was also developed from scratch and consists of approximately 3500 lines of C code. The *PStext Widget* is a modified Athena Text Widget. The

Athena Widget was provided in the MIT distribution of X11, Release 4 and is comprised of around 11,000 lines of code. The Widget was modified to incorporate the inter-application linking capabilities of the Xlt Toolkit. We felt this aspect of the project was particularly important since several X-based text editor applications, such as *Xedit*, inherit their functionality from this Widget. PStext Widget demonstrates the ease with which link services functionality can be incorporated into a suite of applications [14].

We were surprised to learn that only about ten percent of the source code for all three applications is devoted to link services. In these simple cases, tailoring new and existing applications to support persistent selection and participation appears to be relatively uncomplicated. Use of the Xlt Toolkit clearly facilitated the development process.

The primary goal in implementing SP1 and HB1, however, was to provide a proof of principle for the systems' underlying model of hypermedia. The prototypes are, in a sense, a feasibility study to explore the overall behavior of the HBMS architecture. While performance has not been a major concern to us during this initial phase of research, preliminary results suggest that reasonable response times are achievable. To date, information spaces consisting of a few hundred objects have been manipulated. Real-time responses from the ASM server are entirely acceptable within a moderately loaded network of a dozen workstations. However, response times from the OM server for large object fetches are prohibitively low. We believe that this problem can be overcome with intelligent prefetching of objects into local caches.

Saberel's lack of constructed types and overall simplicity proved to be a problem. For example, BRIDGEs in Figure 8 would have been more accurately modeled as an aggregation type, and SIDEs and ASSOCIATIONs should really be groupings. Besides giving the semantics of object identity greater precision, the presence of constructed types would have simplified the language mechanisms required to manipulate the database. Since Saberel does not directly enforce integrity constraints at the database level, it was necessary to explicitly include these constraints in many operations. Our impression is that a more complex semantic model would have significantly reduced the coding effort by providing simpler, higher-level abstractions [9].

RELATED WORK

There is increasing interest in providing effective database support for hypermedia. In this section, we comment briefly on related research appearing in the hypermedia and database literature.

Related Work in Hypermedia

The hypermedia literature addresses data management issues in several arenas. There is an emerging body of work on hyperbases *per se*. In addition, much can be learned by examining the backend functionality of monolithic and non-monolithic systems. Work on formal modeling is also contributing to hyperbase research, as are the areas of performance evaluation, information retrieval, and techniques for hypermedia search and query.

It is probably fair to say that work on hyperbases began with Tektronix' Hypertext Abstract Machine (HAM). HAM is a general-purpose, transaction-based, multi-user server for a hypermedia storage system [24]. It has been used as a backend to the Neptune system which supports hypermedia-based CAD and CASE applications. HAM, however, is a low-level storage engine that is intended to provide sufficient generality for use with other applications [25]. HAM was designed to support monolithic applications, and, as a result, its data model corresponds to the hypermedia abstractions envisioned in these applications. Its monolithic orientation strongly influences the way functionality is partitioned between applications and the database server. Some of the operations implemented in HAM require that its server know about application-level abstractions.

In addition to HAM, there are two prominent hyperbase projects whose systems carry identical names. The University of Aalborg's HyperBase is similar to HAM, although it instantiates a simpler data model [22]. Like HB1, it takes a client/server approach to distribution, and its architecture is intended to be a general foundation upon which a wide variety of hypermedia applications can be built. A special emphasis in this HyperBase project has been database support for collaborative work.

GMD-IPSI's HyperBase is built on top of the Sybase relational DBMS and has been used extensively to support authoring tools such as SEPIA [21]. Object-oriented modeling techniques were used to implement this HyperBase, and there are plans to eventually replace Sybase with an object-oriented DBMS. Application independence has been an important design goal for both HyperBase projects.

Puttress and Guimaraes have developed a toolkit approach to the construction of hypermedia systems that includes explicit support for storage management [26]. Their storage system, Eggs, is a simplified version of the HAM. Although Eggs was modeled after HAM, there are several important differences. Eggs does not interpret application-level abstractions, implements a simple version management system, and manages objects that can reference files in the underlying

operating system. This allows users to cross back and forth between the hypermedia environment and the underlying computing environment.

The University of North Carolina's Distributed Graph Service (DGS) is a hypermedia storage service that is being developed to support the Artifact-Based Collaboration (ABC) environment [27]. Like the systems described above, DGS is intended to be sufficiently general and flexible to support other applications. However, unlike other hyperbase management systems, DGS' design is strongly influenced by distributed file systems architectures rather than DBMS architectures. Two key design goals for DGS are scalability and an open systems policy for application programs.

The latest hyperbase to emerge from the University of Aalborg's work provides several interesting extensions to their original HyperBase. The new system, called Hyperform, is a centralized hyperbase server implemented around the Elk (Extension Language Kit) Interpreter, a Scheme dialect [28]. At the heart of the system is an object-oriented data modeling facility. The novel aspect of Hyperform's design is that the concept of extensibility has been introduced at the hyperbase level. New data objects and operations can be added dynamically, thereby allowing the hyperbase to be arbitrarily customized. This feature enables client/server architectures that support extensibility at both the backend and application levels. This approach also supports multiple, simultaneous data models, and is the first heterogeneous, multihyperbase system to be prototyped.

Finally, the University of Melbourne is implementing a hyperbase system called Hyperion [29]. An interesting feature of Hyperion is that its layered architecture allows abstract "nodes" and "links" to be described independent of storage organization and physical representation.

HBMS research is also being influenced by experiences with monolithic systems. Most existing hypermedia systems implement backend functionality themselves using the file system provided by the underlying operating system. There are exceptions to this rule, however. Two such systems are Intermedia and the Virtual Notebook System.

Early versions of Intermedia were implemented on top of the INGRES relational DBMS. In fact, one of the important contributions of Brown University's IRIS (Institute for Research in Information and Scholarship) project has been an analysis of the applicability of relational and object-oriented (OO) methods to hypermedia databases [30]. They concluded that an OO approach would provide a better facility for modeling complex structures. As part of this work, an OO database schema for Intermedia was developed. However, the transition to an OO DBMS never occurred. Intermedia remains an essentially monolithic application, with functional extensibility

provided through traditional object-oriented programming techniques. The classes that implement Intermedia functionality are now called IRIS Hypermedia Services, and one of its components is the Link Server. The Link Server is a network accessible backend to Intermedia. Persistent storage is currently provided by the commercially available C-Tree system.

The Virtual Notebook System (VNS) is being designed to facilitate information acquisition, sharing, and management in collaborative scientific and medical research [31]. A principal goal of the VNS project has been to distribute VNS within a heterogeneous computing environment. VNS's design envisions a number of work group servers (WGSs) connected through a network. WGSs provide local information storage and management by maintaining the group's hypermedia data in a relational database on the group's server. Through a special Gatekeeper computer, users of a WGS can share information with other groups and access information stored in other information and database services.

This architecture makes VNS fairly distinctive. It is an essentially monolithic system with a separable backend. However, the VNS backend is a truly decentralized, distributed, homogeneous, database management facility. This is in contrast to systems such as Knowledge Management System (KMS) that accrue network accessibility and decentralization by way of the underlying operating system's distributed file system protocols [32].

Efforts to develop open architectures for hypermedia systems are influencing HBMS development as well. Three important systems whose architectures are primarily oriented toward an open systems approach are: PROXHY, Sun MicroSystem's Link Service, and the University of Southern California's Distributed Hypertext (DHT) architecture.

PROXHY is an architecture for constructing hypermedia systems that is able to integrate diverse applications under a common hypermedia model [33]. It is unusual in that the architecture integrates principles from hypermedia, process, and object-oriented models of software construction. The PROXHY architecture allows links to cross application boundaries, thereby supporting a notion of inter-application linking. PROXHY's backend layer can be organized in a number of ways, however, in its prototypic implementation, the system utilizes the functionality available in the underlying file system.

Sun MicroSystem's Link Service is a product that is shipped with Sun's programming-in-the-large software development environment called Network Software Environment (NSE) [34]. NSE allows users to make and maintain explicit and persistent links between objects managed by

autonomous frontend applications. Its capacity to support inter-application linking makes NSE's notion of extensibility similar to that seen in PROXY and SP1. The NSE server is centralized and can be accessed over a network by multiple applications.

USC's Distributed Hypertext (DHT) architecture illustrates an interesting move toward an open systems architecture combined with distributed, heterogeneous database techniques [35]. DHT is based on a client/server model and includes four components: a common hypermedia data model, a communication protocol, servers, and client applications. The heart of the system is its collection of database servers. Each server consists of a gateway process and an information repository. The gateway process transforms hypermedia operations into local access operations and local information objects into DHT nodes or links. The information repositories are databases, file systems, or special-purpose storage managers. From the repository's view, the gateway process appears as another local application accessing the repository's data. In this way, DHT can incorporate existing databases without having to copy their data or modify their schemas.

There is currently widespread interest in developing formal reference models for hypermedia, and this work is contributing indirectly to HBMS design. Among the most prominent are those described in Afrati and Koutras [20], Delisle and Schwartz [25], Furuta and Stotts [36], Garg [37], Halasz and Schwartz [18], Kacmar and Leggett [33], Lange [19], Schütt and Streit [21], and Tompa [38]. Not surprisingly, these models vary along many dimensions. A detailed consideration of their merits is beyond the scope of this paper. Unfortunately, some of the more influential models to appear in the past few years, such as the Dexter model and that of Afrati and Koutras, have not yet been fully implemented.

So far, it seems that the greatest amount of database-related work appearing in the hypermedia literature pertains to the issue of data access. Several research projects are extending information retrieval techniques to hypermedia [39, 40, 41, 42, 43, 44]. Others are developing formalisms to support structural queries [20, 45, 46]. Finally, there is a contingent examining ways of combining traditional querying with the type of user-directed browsing that characterizes data access in hypermedia systems [47, 48, 49].

Related Work in Database

In the database area, several advancing fronts will undoubtedly shape the future design of hyperbase systems. Research on object-oriented, semantic, structural object-oriented, and extended

relational database systems is particularly relevant. One of the important challenges to both the hyperbase and database communities is discovering how existing database techniques can apply to the data management problems of hypermedia systems.

Many of the application domains requiring database support today, such as computer-aided design (CAD), computer-aided software engineering (CASE), multimedia databases, office information systems (OIS), and, of course, hypermedia, require objects that have arbitrarily complex internal structures and behavioral components. There is also the requirement that they participate in long-duration transactions where humans interact with information over an extended period of time. These factors have resulted in a search for new and more effective database technologies.

The object-oriented database paradigm is currently of enormous interest to the database community. It is based on the notion of encapsulating code and data into a single unit referred to as an object. The interface between an object and the rest of the system is defined by a set of messages. This abstract data type approach, where operations, or methods, are embedded within types, derives from object-oriented programming language design. In this view, an object is in control of its own behavior, behavior that is invoked by the messages that are sent to the object. Similar objects are grouped to form a class, and each object is an instance of its class from which it inherits key characteristics and behaviors.

Object-oriented data management refers to a collection of run-time issues such as naming, persistence, concurrency control, distribution, version control, and security. These issues have been addressed for many years by traditional database methodologies. Recent work on object-oriented database systems has focused on transferring this technology to the problem of managing objects. Several prominent object-oriented database systems are the result, including GemStone from ServioLogic [50], ORION from MCC [51], Ontologic's ONTOS [52], and Iris from Hewlett-Packard Labs [53]. An overview of existing object-oriented databases is presented in [10, 54, 55, 56].

The concept of object varies widely [10]. In particular, there is growing recognition that the traditional notions of object derived from object-oriented programming are often too narrowly construed. While dynamically invoked methods, inheritance hierarchies, and encapsulation are desirable for many applications, the variability present in typing and invocation mechanisms makes database support for a simpler notion of object appealing. This has resulted in work on persistent object stores such as Mneme [8].

A Mneme object is a contiguous block of memory associated with an object identifier. Execution semantics are not applied to Mneme objects, and they have no type, class, associated methods, or inheritance. The notion is that such an approach maximizes applicability and efficiency of the store and provides a flexible framework upon which other mechanisms – including OO features – may be built. These ideas substantially originated in work on the Multics operating system and appear in a variety of storage systems [10].

Semantic database models attempt to provide more powerful abstractions for specifying database schemas than are supported by traditional models. They also tend to encourage a navigational view of data. Semantic modeling was initially introduced in the early 1970s to facilitate the design of database schemas. Since then, research has resulted in the development of powerful mechanisms for representing the structural aspects of data. A number of full-fledged semantic database management systems now exist and offer a wide range of data modeling capabilities. For a review of existing semantic models see [11, 12].

Object-oriented database systems provide *behavioral* abstractions while semantic database systems provide *structural* abstractions. Another important trend in the design of advanced database systems is the integration of behavioral and structural modeling techniques. *Structural object-oriented* database systems attempt such an integration [57, 58, 59, 60]. The synergy between the two views is impressive. Together they provide compatible ways of encapsulating both the structural and behavioral aspects of complex objects. Work in this area and work on persistent object stores have clearly influenced the design of HB1.

Some new systems are attempting to address the shortcomings of existing database technology through extensions to the relational model. POSTGRES, for example, provides many of the same capabilities of object-oriented and semantic models by adding extensions to the relational model [61]. In this regard, POSTGRES could be viewed as an example of a structural object-oriented database system. Philosophically, the goal is to make as few changes as possible to the relational model since there is broad familiarity with relational methods, they are efficient, and they are relatively simple to understand.

The relational model underlying POSTGRES has been extended with abstract data types including user-defined operators and procedures, relation attributes of type procedure, and attribute and procedure inheritance. These mechanisms can be used to simulate a wide variety of semantic and object-oriented data modeling constructs including aggregation and generalization, complex objects with shared subobjects, and attributes that reference tuples in other relations.

We close this section by commenting on several areas of database research that are also indirectly influencing work on hyperbase management systems. Of particular relevance is work on the data modeling and access requirements of multimedia applications [62, 63, 64]. In addition, the problems posed by transaction management and version control in VLSI CAD and CASE environments appear to be similar to those in hypermedia systems [65, 66, 67]. Database extensibility is an important issue that is currently receiving considerable attention [68, 69]. The notion of mediators or agency in databases systems [59, 70, 71] has contributed to the work done on HB1, as has research on heterogeneous multidatabase systems [72, 73].

RESEARCH ISSUES AND FUTURE WORK

Research on hyperbase management systems is in an early, experimental phase. As a result, the field is in tremendous flux. Comprehensive and unifying solutions to the data management problems posed by advanced hypermedia systems have yet to emerge. The following, however, are among the major issues that must be addressed in future HBMS research:

- *Models and architectures.* Issues of scalability, extensibility, interoperability, architectural openness, distribution, and platform heterogeneity are of critical importance.
- *Node, link, and structure management.* (Hypermedia data and metadata management.) Data management facilities for hypermedia must address issues relating to object identity, naming, and constraints on object and structure integrity. In addition, support for object composition, contexts, and views is critical, and the management of exotic data types, including spatial, temporal, image, sequence, graph, probabilistic, user-defined, and dynamic types is essential. The effective management of behavioral entities will be important in many settings.
- *Browsing / search and query.* Optimizing the synergy between hypermedia's navigational approach to data access and traditional search and query must be addressed at the HBMS level. Important issues include the introduction of multi-level store indexing techniques, agency, hyperbase heterogeneity, extensibility, and optimization.
- *Version control.* Effective support for versioning will require an understanding of precisely which entities need to be versioned and when version creation occurs. In hypermedia, there are opportunities to version not only information, but structure as well. It will also be important to understand how version control should be partitioned between the HBMS and application levels.

- *Concurrency control, transaction management, and notification.* The types of interactivity and operations that characterize hypermedia applications create a requirement for managing short, long, and very long hyperbase transactions. HBMS support for collaboration and the sharing of information will be of critical importance.

Clearly, HB1 lacks much of the above functionality. Future work will be directed toward addressing the system's current shortcomings. In particular, we will introduce transaction management in order to transition the HBMS into a multi-user environment. In addition, we are designing policies and mechanisms for version management and object composition. The next implementation of the HBMS will utilize the extended relational database system POSTGRES as a Storage Manager.

CONCLUSION

The unique data management requirements of advanced hypermedia systems require capabilities that are not generally provided by the current generation of database management systems. In this paper, we have described the design and prototypic implementation of a hyperbase management system intended to address some of these needs. The prototype, HB1, implements several distinguishing features. For example, the HB1 system:

- meets the storage requirements of an open, extensible, object-based system architecture for distributed, inter-application linking;
- abstracts structure, behavior, and information from hypermedia;
- is composed of two network-accessible server processes: one providing object management, the other managing hypermedia connectivity information;
- uses a semantic network database management system to manage physical storage;
- promotes a toolkit approach to client / hyperbase construction;

- instantiates an architectural framework that is scalable, flexible, and useful as a vehicle for general research on HBMS organization.

HB1 has demonstrated its effectiveness in providing backend services to an advanced hypermedia system prototype. Experiences gained so far in this work have been used to identify several important issues to be addressed in future HBMS research. These include developing techniques for transaction management, controlled sharing, and versioning that will be effective in large-scale, multi-user, hypermedia-based information systems of the future.

ACKNOWLEDGMENTS

The authors would like to thank other members of the team that constructed the software described in this paper: Ron Szabo, Doug Miller, Joe Drufke, Don Dylla, and Tim Roden. Thanks also to Ed Cunniss for helpful advice on early drafts of the paper and to Cindy Kunz for assistance in preparing the manuscript. Bob Griffith, Jeff Barber, and Ken Briskey provided valuable assistance in our work with Saberel. This work was supported in part by a grant from the Advanced Workstations Division of IBM Corporation, Austin, Texas, under Research Agreement 194.

REFERENCES

1. *ACM Transactions on Office Information Systems*, **7** (1) (1989).
2. *Communications of the ACM*, **31** (7) (1988).
3. *Proceedings of the Hypertext '87 Conference*, Chapel Hill, NC, ACM Press, (November, 1987).
4. *Proceedings of the Hypertext '89 Conference*, Pittsburgh, PA, ACM Press, (November, 1989).
5. *Proceedings of the Hypertext '91 Conference*, San Antonio, TX, ACM Press, (December, 1991).
6. J. Conklin, 'Hypertext: An introduction and survey', *IEEE Computer*, **20** (9), 17–41 (1987).
7. A. Rizk, N. Streitz, and J. Andre, eds., *Hypertext: Concepts, Systems, and Applications. Proceedings of the European Conference on Hypertext*, France, Cambridge University Press, Cambridge, UK, (November, 1990).

8. J. E. B. Moss, 'Design of the Mneme persistent object store', *ACM Transactions on Office Information Systems*, **8** (2), 103–139 (1990).
9. J. L. Schnase, J. J. Leggett, D. L. Hicks, and R. L. Szabo, 'Semantic data modeling of hypermedia associations', *ACM Transactions on Information Systems*, **11** (1), 27–50, (1993).
10. R. G. G. Cattell, *Object Data Management: Object-oriented and Extended Relational Systems*, Addison-Wesley, Reading, MA, 1991.
11. R. Hull and R. King, 'Semantic database modeling: Survey, applications, and research issues', *ACM Computing Surveys*, **19** (3), 201-260 (1987).
12. J. Peckham and F. Maryanski, 'Semantic data models', *ACM Computing Surveys*, **20** (3), 153–189 (1988).
13. J. L. Schnase, J. J. Leggett, and D. L. Hicks, 'HB1: Initial design and implementation of a hyperbase management system', *Department of Computer Science Technical Report No. TAMU-HRL 91-003*, Texas A&M University, College Station, TX (1991).
14. J. E. Drufke, J. J. Leggett, D. L. Hicks, and J. L. Schnase, 'The derivation of a hypertext widget class from the Athena text widget', *Department of Computer Science Technical Report No. TAMU 91-002*, Texas A&M University, College Station, TX (1991).
15. R. Scheifler, J. Gettys, and R. Newman, *X Window System*, Digital Press, Bedford, MA, 1988.
16. M. Palaniappan, N. Yankelovich, and M. Sawtelle, 'Linking active anchors: A stage in the evolution of hypermedia', *Hypermedia*, **2** (1), 47–66 (1990).
17. N. Yankelovich, B. Haan, N. Meyrowitz, and S. Drucker, 'Intermedia: The concept and the construction of a seamless information environment', *IEEE Computer*, **21** (1), 81–96 (1988).
18. F. Halasz and M. Schwartz, 'The Dexter hypertext reference model', in *Proceedings of the NIST Hypertext Standardization Workshop*, NIST, Gaithersburg, MD, pp. 95-133, (1990).
19. D. Lange, 'A formal model for hypertext', in *Proceedings of the NIST Hypertext Standardization Workshop*, NIST, Gaithersburg, MD, pp. 145–166, (1990).
20. F. Afrati and C. Koutras, 'A hypertext model supporting query mechanisms', in *Hypertext: Concepts, Systems, and Applications. Proceedings of the European Conference on Hypertext*, France, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, Cambridge, UK, pp. 52–66, (November, 1990).

21. H. A. Schütt and N. A. Streitz, 'Hyperbase: A hypermedia engine based on a relational database management system', in *Hypertext: Concepts, Systems, and Applications. Proceedings of the European Conference on Hypertext*, France, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, Cambridge, UK, pp. 95–108, (November, 1990).
22. U. K. Wiil, 'Design and implementation of a hyperbase', *Institute for Electronic Systems Technical Report No. IR 90-03*, Department of Mathematics and Computer Science, The University of Aalborg, Aalborg, Denmark (1990).
23. F. Halasz, 'Reflections on NoteCards: Seven issues for the next generation of hypermedia systems', *Communications of the ACM*, **31** (7), 836–852 (1988).
24. B. Campbell and J. Goodman, 'HAM: A general-purpose hypertext abstract machine', *Communications of the ACM*, **31** (7), 856–861 (1988).
25. N. Delisle and M. Schwartz, 'Neptune: A hypertext system for CAD applications', in *Proceedings of the ACM International Conference on the Management of Data (SIGMOD)*, pp. 132–143, (1986).
26. J. J. Puttress and N. M. Guimaraes, 'The toolkit approach to hypermedia', in *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext* France, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, Cambridge, UK, pp. 25–37, (November, 1990).
27. J. B. Smith and F. D. Smith, 'ABC: A hypermedia system for artifact-based collaboration', in *Proceedings of the Third ACM Conference on Hypertext (Hypertext '91)*, San Antonio, TX, pp. 179–192, (December, 1991).
28. U. K. Wiil and J. J. Leggett, 'Hyperform: using extensibility to develop dynamic, open and distributed hypertext systems', in *Proceedings of the European Conference on Hypertext (ECHT '92)*, Milan, Italy, pp. 251–261, (1992).
29. 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 First Australian Multi-Media Communications, Applications and Technology Workshop*, pp. 152-161, (1991).
30. K. Smith and S. Zdonik, 'Intermedia: A case study of the differences between relational and object-oriented database systems', in *Proceedings of the ACM OOPSLA '87 Conference*, ACM, New York, pp. 452–465, (1987).
31. F. M. Shipman, R. J. Chaney, and G. A. Gorry, 'Distributed hypertext for collaborative research: The Virtual Notebook System', in *Proceedings of the Second ACM Conference on Hypertext (Hypertext '89)*, Pittsburgh, PA, pp. 129–136, (November, 1989).

32. R. Akscyn, D. McCracken, and E. Yoder, 'KMS: A distributed hypermedia system for managing knowledge in organizations', *Communications of the ACM*, **31** (7), 820–835 (1988).
33. C. J. Kacmar and J. J. Leggett, 'PROXHY: A process-oriented extensible hypertext architecture', *ACM Trans. Inf. Syst.*, **9** (4), 399–419 (1991).
34. A. Pearl, 'Sun's link service: A protocol for open linking', in *Proceedings of the Second ACM Conference on Hypertext (Hypertext '89)*, Pittsburgh, PA, pp. 137-146, (November, 1989).
35. J. Noll and W. Scacchi, 'Integrating diverse information repositories: A distributed hypertext approach', *IEEE Computer*, **24** (12), 38–45 (1991).
36. R. Furuta and P. D. Stotts, 'The trellis hypertext reference model', in *Proceedings of the Hypertext Standardization Workshop*, Gaithersburg, MD, pp. 83–93, (1990).
37. P. K. Garg, 'Abstraction mechanisms in hypertext', *Communications of the ACM*, **31** (7), 862–870 (1988).
38. F. W. Tompa, 'A data model for flexible hypertext database systems', *ACM Transactions on Office Information Systems*, **7** (1), 85–100 (1989).
39. P. D. Bruza, 'Hyperindices: A novel aid for searching in hypermedia', in *Hypertext: Concepts, Systems, and Applications. Proceedings of the European Conference on Hypertext*, France, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, Cambridge, UK, pp. 109–122, (November, 1990).
40. W. B. Croft and H. Turtle, 'A retrieval model incorporating hypertext links', in *Proceedings of the Hypertext '89 Conference*, Pittsburgh, PA, pp. 213-224, (November, 1989).
41. D. B. Crouch, C. J. Crouch, and G. Andreas, 'The use of cluster hierarchies in hypertext information retrieval', in *Proceedings of the Hypertext '89 Conference*, Pittsburgh, PA, pp. 225-237, (November, 1989).
42. M. Frisse, 'Searching for information in a hypertext medical handbook', *Communications of the ACM*, **31** (7), 880–886 (1988).
43. M. E. Frisse and S. B. Cousins, 'Information retrieval from hypertext: Update on the dynamic medical handbook project', in *Proceedings of the Hypertext '89 Conference*, Pittsburgh, PA, pp. 199–212, (November, 1989).

44. D. Lucarella, 'A model for hypertext-based information retrieval', in *Hypertext: Concepts, Systems, and Applications. Proceedings of the European Conference on Hypertext*, France, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, Cambridge, UK, pp. 81–94, (November, 1990).
45. C. Beeri and Y. Kornatzky, 'A logical query language for hypertext systems', in *Hypertext: Concepts, Systems, and Applications. Proceedings of the European Conference on Hypertext*, France, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, Cambridge, UK, pp. 67–80, (November, 1990).
46. M. P. Consens and A. O. Mendelzon, 'Expressing structural hypertext queries in GraphLog', in *Proceedings of the Hypertext '89 Conference*, Pittsburgh, PA, pp. 269–292, (November, 1989).
47. C. Clifton and H. Garcia-Molina, 'Indexing in a hypertext database', in *Proceedings of the 16th Conference on Very Large Data Bases*, Brisbane, Australia, pp. 36-49, (August, 1990).
48. L. Gallagher, R. Furuta, and P. David Stotts, 'Increasing the power of hypertext search with relational queries', *Hypermedia*, **2** (1), 1–14 (1990).
49. C. Watters and M. A. Shepherd, 'A transient hypergraph-based model for data access', *ACM Transactions on Information Systems*, **8** (2), 77–102 (1990).
50. P. Butterworth, A. Otis, and J. Stein, 'The GemStone object database management system', *Communications of the ACM*, **34** (10), 64–77 (1991).
51. W. Kim, N. Ballou, H. Chou, J. F. Garza, and D. Woelk, 'Features of the ORION object-oriented database system', in *Object-Oriented Concepts, Databases, and Applications*, ed. W. Kim and F. H. Lochovsky, Addison-Wesley Publishing Co., Reading, MA, pp. 251–282, (1989).
52. T. Andrews, C. Harris, and K. Sinkel, 'ONTOS: A persistent database for C++', in *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD*, ed. R. Gupta and E. Horowitz, Prentice-Hall, Englewood Cliffs, NJ, pp. 387–406, (1991).
53. D. Fishman, D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M. Neimat, T. Ryan, and M. Shan, 'IRIS: An object-oriented database management system', *ACM Transactions on Office Information Systems*, **5** (1), 48–69 (1987).
54. *Communications of the ACM*, **34** (10) (1991).
55. R. Gupta and E. Horowitz, *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

56. W. Kim and F. Lochovsky, *Object-Oriented Concepts, Databases, and Applications*. ACM Press/Addison-Wesley, Inc., New York, NY, 1989.
57. A. J. Berre, 'SOOM and Tornado-^{*}: Experience with database support for object-oriented applications', in *Advances in Object-Oriented Database Systems: Proceedings of the 2nd International Conference on Object-Oriented Database Systems*, ed. K. Dittrich, Springer-Verlag, New York, pp. 104–109, (1988).
58. K. R. Dittrich, W. Gotthard, and P. C. Lockemann, 'DAMOKLES – A database system for software engineering environments', in *Proceedings of the IFIP Workshop on Advanced Programming Environments*, Trondheim, Norway, ed. R. Conradi, R. M. Didriksen, and D. H. Wanvik, Springer-Verlag, New York, pp. 353–371, (June, 1986).
59. S. E. Hudson and R. King, 'Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system', *ACM Transactions on Database Systems*, **14** (3), 291-321 (1989).
63. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and System Design*, Prentice-Hall, Englewood Cliffs, NJ, 1990.
61. M. Stonebraker and G. Kemnitz, 'The POSTGRES next-generation database management system', *Communications of the ACM*, **34** (2), 78–92 (1991).
62. M. Caplinger, 'An information system based on distributed objects', in *Proceedings of the OOPSLA '87 Conference*, pp. 126–137, (October, 1987).
63. S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria, 'Multimedia document presentation, information extraction, and document formation in MINOS: A model and a system', *ACM Transactions on Office Information Systems*, **4** (4), 345-383 (1986).
64. D. Woelk and W. Kim, 'Multimedia information management in an object-oriented database system', *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, Brighton, England, pp. 319–329, (1987).
65. D. Batory and W. Kim, 'Modeling concepts for VLSI CAD objects', *ACM Transactions on Database Systems*, **10** (3), 322–346 (1985).
66. R. H. Katz, 'Toward a unified framework for version modeling in engineering databases', *ACM Computing Surveys*, **22** (4), 375–408 (1990).
67. D. Maier, J. Stein, A. Otis, and A. Purdy, 'Development of an object-oriented DBMS', in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 472–482, (September, 1986).

68. D. S. Batory, J. R. Barnett, F. F. Garza, K. P. Smith, K. Tsukauda, B. D. Twichell, and T. E. Wise, 'GENESIS: A reconfigurable database management system', *IEEE Transactions on Software Engineering*, 1258–1272 (1990).
69. M. Carey, D. DeWitt, J. Richardson, and E. Shekita, 'Object and file management in the EXODUS extensible database system', in *Proceedings of the Twelfth International Conference on Very Large Data Bases*, Kyoto, Japan, pp. 91-100, (August, 1986).
70. C. A. Ellis and S. J. Gibbs, 'Active objects: Realities and possibilities,' in *Object-Oriented Concepts, Databases, and Applications*, ed. W. Kim and F. H. Lochovsky, Addison-Wesley Publishing Co., Reading, MA, pp. 561–572, (1989).
71. G. Wiederhold, 'Mediators in the architecture of future information systems', *IEEE Computer*, **25** (3), 38–49 (1992).
72. *ACM Computing Surveys*, **22** (3) 1990.
73. *IEEE Computer*, **24** (10) (1991).

Table 1. Message protocol for the HB1 hyperbase management system

Server Request Message (Parameters)		Server Response Message (Parameters)	
<i>Object Manager Messages</i>			
Create Object	(Data, Name, Type, Size)	CO Acknowledge	(OID, RC) ¹
Delete Object	(OID)	DO Acknowledge	(RC)
Retrieve Object	(OID)	RO Acknowledge	(Data, Name, Type, Size, RC)
Store Object	(OID, Data, Name, Type, Size)	SO Acknowledge	(RC)
Get Name	(OID)	GN Acknowledge	(Name, RC)
Set Name	(OID, Name)	SN Acknowledge	(RC)
Get Type	(OID)	GT Acknowledge	(Type, RC)
Set Type	(OID, Type)	ST Acknowledge	(RC)
Resolve Attribute	(Name, Type)	RA Acknowledge	(OIDs, RC)
<i>Association Set Manager Messages</i>			
Attach Anchor	(n {PSID, CompID, AppID}) ²	AA Acknowledge	(RC)
Attach Link		AL Acknowledge	(RC)
Detach Anchor	(n {PSID, CompID, AppID}) ²	DA Acknowledge	(RC)
Detach Link	(n {PSID, CompID, AppID}) ²	DL Acknowledge	(RC)
Follow Association	({PSID, CompID, AppID})	FA Acknowledge	(m {LinkID, n {AnchorID, o {PSID, AppID, CompID}}], RC) ³

¹ RC = return code.

² $n \geq 0$.

³ $m \geq 0$; $n \geq 0$; $o \geq 0$.