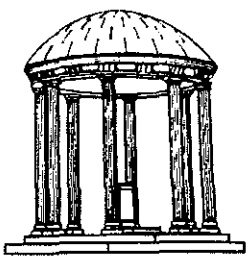
The UNC Graph Server: A Distributed Hypermedia Data Management System

TR91-019

April, 1991

Douglas E. Shackelford
John B. Smith
Joan Boone
Barry Elledge



The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175
919-962-1792
jbs@cs.unc.edu

A TextLab Report
Major support has come from NSF (Grant #IRI-9015443) and the IBM Corporation (SUR Agreement #866), with additional support from ONR (Contract #N00014-86-K-00680).

Abstract

Our project is studying the process by which groups of individuals work together to build large, complex structures of ideas and is building a distributed hypermedia system to support that process. This description focuses on the hypermedia data management system -- which we call a graph server -- we are developing to support groups of collaborators working on individual workstations within a distributed computing environment. The discussion covers the graph-based data model provided by the server to client hypermedia programs as well as the architecture of the system. A number of research issues are raised and discussed in context including: partitioning the hypermedia graph; an open, extensible architecture for applications; composite objects; anchored links; and the integrity, consistency, and completeness of the artifact as a whole.

Introduction

Our research is concerned with the activities of groups as they work collaboratively to build a large, common database of materials. As part of this project, we are building a distributed hypermedia environment for software development. That system -- which we call the Artifact-Based Collaboration (ABC) system -- is described in more detail in a companion paper [Smith & Smith, 1991]. Here, we focus on one component -- the hypermedia data management system, which we call the *graph server*.

The key issue that must be addressed by a hypermedia data management system is scale. Most hypermedia systems have been single user systems running on individual microcomputers or workstations. They support structures containing from a few hundred to a few thousand nodes. To meet the requirements of an O(10) person software team working over a period of months, a hypermedia data management system must support O(10,000) nodes and O(100,000) links. For larger industrial projects, such as those carried out by defense contractors, requirements are likely to be several orders of magnitude larger. No current system or design can scale to meet these needs. For organization-wide hypermedia applications currently being contemplated, the requirements are larger still. As a university research project, we cannot build a system that meets these requirements, but we can build proof of concept systems that elucidate key issues.

To be scalable, a hypermedia data management system must be partitionable. This is true from the standpoints of capacity, performance, and human comprehension. Any single-platform storage system will be outgrown eventually, and transmission bandwidth will pose problems in servicing from a single site the needs of large, widely distributed groups for at least the foreseeable future. The demands for human comprehension of large structures are different but no less important. When hypermedia structures become too large or too complex, human beings lose their orientation and become "lost in hyperspace" [Halasz, 1987]. To avoid this condition, users must be able to isolate small, coherent portions of large hypermedia structures in order to understand them and to work with them, but they must also be able to freely create links between "distant" parts of the database. Thus, partitioning poses a variety of problems, both for the data model provided by a server and for the system architecture that supports it.

In the discussion that follows, we describe the hypermedia data management system we are building. The discussion begins with an overview of the ABC system to provide context; the majority of the discussion, however, concerns the data model, the architecture of the server, and their relation to other research and other points of view. We also identify and discuss in context issues that we believe are important for hypermedia data management systems if they are to be be scaled upward to accommodate large, distributed hypermedia applications.

System Overview

ABC has six key components (see Figure 1) that include the *graph server*, a set of *graph browsers*, a set of *data application* programs, a *shared window* conferencing facility, and real-time *video and audio*. The sixth component, a set of *protocol tools* for studying group behaviors and strategies, is not illustrated in the figure.

The *graph server* is the (logically central) data management system in which all of the data objects associated with a project are stored. Individual documents (or other collections of data) are represented as separate graph structures in which nodes in one subgraph may be related to nodes in another subgraph. The working environment is distributed across multiple workstations connected

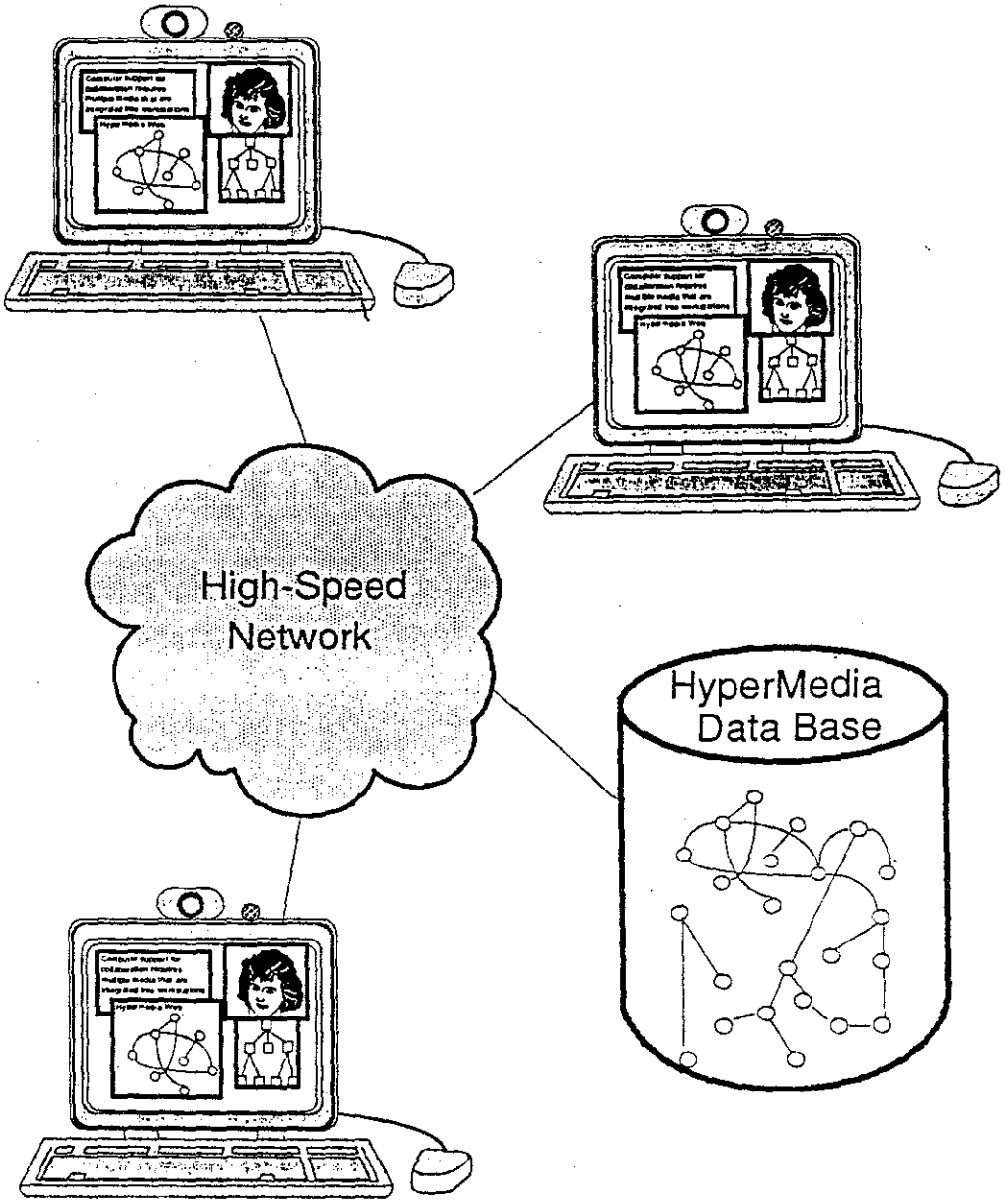


Figure 1:
Artifact-Based Collaboration System Overview

to one another and the graph server through a data communications network. One kind of tool available on the workstations is a set of *graph browsers*, built by our project, that support development and use of different types of subgraphs -- e.g., trees, general directed graphs, lists, etc. A second kind of tool is a set of conventional *data applications* used to work with "raw" data stored as the contents of nodes within graph structures. We are taking an open architecture approach so that people may use familiar text editors, drawing packages, etc., as applications

A *shared windows* facility permits any browser or application program to be shared by two or more users. Users can, thus, set-up conferences, add and delete members, pass control from one member to another, launch an application, etc.. A *video/audio system* enables users to see and talk with one-another as they work on the same drawing or document using the shared windows facility. At present, video is supported by a closed-circuit analog network, but we are developing a real-time digital video system, based on DVI technology, that will allow voice and video to be sent over the data network and integrated directly into the workstation.

A sixth component, included in the system but not illustrated in the figure, is a set of *protocol tools* for recording users' actions in machine-readable form, for analyzing them, and for displaying collaborative strategies. These data will be analyzed to identify patterns of behavior so that we can see who works with whom, on what portions of the artifact, at what time during a project, etc.

Data Model

The data model provided by the UNC Graph Server is based on the formalism of directed graphs. It is a multilevel model that includes four levels, excluding physical storage. One of its primary distinctions from other models and one that we believe will help to make partitioning practical is the concept of *subgraph*. Thus, one should pay particular attention to its role in the data model.

Primitive objects in the UNC Graph Server include *nodes* and *links*, with *attributes* on both nodes and links. Nodes, but also links, may have associated *contents*, which in our case are most often data files whose access is controlled by the graph server. This much of the data model is fairly standard, except that some systems -- e.g., GRAS [Brandes & Lewerentz, 1985] -- restrict the number and/or use of attributes for one element or the other.

Subgraphs play a dual role in the model. From one perspective, they form a second level since they consist of sets of nodes and links. They may be named and, like nodes and links, may have associated attributes and contents, as well. However, subgraphs may be viewed from a second perspective as primitives. That is, within the architecture of the server, they are primary objects in their own right and are handled analogously to nodes and links. The most important function of this dual property is with respect to *contents*. Subgraphs, as well as files, may serve as the contents for a node or for other primitive objects in the model. The implications of this are subtle, but important, and can perhaps best be seen from the end user's point of view.

The overall organization for a document or project might be represented as a tree. This tree may be quite large, which can result in performance problems as well as concurrency problems if more than one person attempt to work on it at the same time. Under the Graph Server data model, overall organization may be specified down to a certain point where the leaves might represent, for example, the chapters of a document. At this point, separate subgraphs may be identified as the contents of the leave-nodes in which the more detailed structure of the document is defined. These subgraphs are disjoint with respect to the first tree in terms of links, but related to it through the content relation. This process can be repeated to as many levels of depth as needed. Relationships among disjoint subgraphs associated through the *contents* property can be seen in Figure 6, below.

The subgraph property is also useful for defining temporary subgraphs that comprise portions of a larger graph (e.g., a branch of a tree) for locking and access control Thus, the subgraph concept provides a simple, clean mechanism that addresses the problem of composite structures, partitions large graph structures into tractable and understandable pieces, and, thus, facilitates distribution across multiple platforms.

The third level of our data model is based on a typing scheme for subgraphs. Subgraphs are classified and maintained as one of five basic *graph types*: *general directed graph, acyclic graph, connected graph, tree,* and *list.* As subgraphs are created and nodes/links added and deleted, the graph server checks each operation to ensure that it does not violate the integrity of the particular graph type. Thus, for example, a tree object can not have a (structural) link that would create a cycle. This typing scheme provides a first step toward addressing issues concerned with consistency, completeness, and correctness of software systems and other complex hypermedia artifacts.

The fourth level of the data model is based on the concept of *composite subgraphs.* In an earlier model, we identified two basic kinds of composite subgraphs: *embedded* and *bridge.* Embedded composites handle problems with cross-references. Since links that violate the integrity of a subgraph type (e.g., crosslinks in a tree) are prohibited, cross references between nodes in different branches of a tree are prohibited. To provide needed flexibility, we defined a second subgraph whose nodes set was restricted to that of the original subgraph but whose links were separate. Since this second subgraph could be of a different graph type -- e.g., general graph -- its links could freely denote relationships of any sort. The two subgraphs -- the original tree and its related subgraph of cross-reference links -- were considered an *embedded* composite subgraph. A similar structure applied to links between two independent subgraphs -- e.g., two trees representing two different documents and the references between them; this kind of structure we termed a *bridge* subgraph composite.

This approach proved awkward. The basic issue being addressed is a distinction between links that are structural with respect to a particular subgraph type -- the backbone of a hierarchical document, the constrained structure of a dag, etc. -- versus more flexible hypertextual relations in which users can represent any kind of semantic relation meaningful to them. Consequently, in a second design, we eliminated the embedded and bridge concepts and, instead, dropped all the way down to level one and introduced a distinction between *structural links* and *hyperlinks.* This led, in turn, to a distinction between structural subgraphs (normally referred to as just *subgraphs*) and hyper subgraphs (normally referred to as *hypergraphs*), the latter consisting of sets of nodes and sets of hyperlinks. Type restrictions apply only to structural links. Thus, a given node may be a member of one or more subgraphs, but it may also be a member of one or more hypergraphs. Two important properties derive from these distinctions. First, subgraph type integrity is preserved in the data model while permitting flexible hypermedia linking. Second, it further partitions the graph space. Thus, when a given (structural) subgraph is loaded for a browser, the server can also identify the hypergraph associated with that subgraph, consisting of all hypertextual references (hyperlinks) to and from nodes in the subgraph. Our current architecture includes an intelligent cache that anticipates that users are likely to access hyperlinks and loads the relevant hypergraph as a background activity soon after a given subgraph is loaded.

The final concept remaining at level four of the data model is the notion of *bound subgraphs.* When multiple users are working on different portions of the same tree, they must be able to lock their respective branches. This is done by designating some specified portion of the tree to be a separate (perhaps temporary) subgraph. At times, users will want their changes to be kept separate from the underlying tree; at other times they will want changes made in the subgraph to be reflected immediately in the underlying tree. Both options are provided through a binding mechanism. Unbound subgraphs provide a convenient mechanism for maintaining different versions of a component while bound subgraphs provide a mechanism that prevents conflicts while supporting immediate updates to the underlying object.

Our understanding of the graph model for a hypertext data management system is still evolving. We believe the issues it raises are deep and that no design-first strategy will work. Instead, we are following a "spiral" development approach. Thus, in the preceding discussion, we have tried to identify problems and point out paths that we followed but did not work out as well as describe our current data model.

Architecture

The architecture of objects within the data model is layered. Nodes are defined in terms of N+2 levels, as shown in Figure 2, where N is the number of subgraphs in which a given node appears. The top layer is the system layer; it contains information about a node that is accessed only by the server, such as the unique id of the node. The second layer contains user-defined information that is common to all subgraph contexts in which the node appears -- for example, *content* is normally the same in all subgraphs. One additional layer is defined for each subgraph in which the node appears. Each layer contains both the incoming and outgoing links for the subgraph and any hyperlinks also beginning or ending with that node within the context of the subgraph for that layer. Node attributes restricted to the particular subgraph are, of course, also stored within the layer, such as the date the node was added to the particular subgraph.

Links also have N+2 layers, as shown in Figure 2, where N is the number of subgraphs in which the link appears. Temporary subgraphs, such as the branch of a tree, provide examples of links existing in more than one subgraph. Again, the architecture contains both system and common layers. Source and target nodes are stored in the system level. A *type* attribute specifies whether the link is a structural link or a hyperlink; thus, both forms of links have the same architecture.

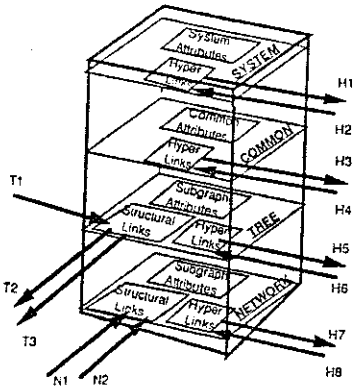
Subgraphs and hypergraphs have the same architecture, as shown in Figure 2, but provide different functions. They contain two layers: a system layer and a common layer. A subgraph is composed of a set of nodes and a set of structural links. Both the source and target nodes of each link must also be contained in the subgraph. A hypergraph is composed of a set of nodes and a set of hyperlinks. Both the source and target nodes of each hyperlink must be contained in the subgraph. Furthermore, every node in the hypergraph must be either the source or target of at least one hyperlink in the hypergraph. Thus, a hypergraph cannot contain unconnected nodes.

A procedural programing interface is provided that is used by client programs -- i.e., browsers and applications -- to work with nodes, links, attributes, subgraphs, etc. Since it includes over 50 methods or procedures, we will not describe it further here.

From a system point of view, the "server" is a logically central data management system that is distributed across multiple platforms. It is programmed in ISIS [Birman & Joseph, 1987] and makes extensive use of its distributed systems capabilities. Figure 3 provides an overview of a basic distributed configuration. The architecture has two principal kinds of components: a *local server* that runs in the client environment and a set of *distributed servers*, including at least one for each of three *process groups* -- subgraphs, nodes, and links.

The *local server* normally runs on the host machine although this is not required. A communication manager communicates with the client program through a language-independent protocol. It packs and unpacks client requests and sends and receives messages to and from the data managment system. A local data manipulation manager implements basic operations on data objects provided by the procedural programming interface, mentioned above.

Layered Model of a Node

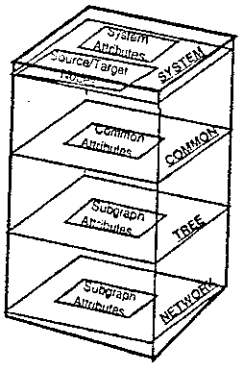


Each of the "Links" boxes on the left represents an ordered set of InLinks and an ordered set of OutLinks

Example:

Structural Links for the tree
InLinks: { T1 } OutLinks: { T2, T3 }

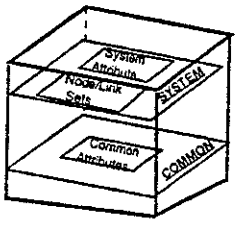
Layered Model of a Link



The System Attributes space in both nodes and links includes a special attribute called the uniqueID.

In addition, links have a type attribute which specifies whether the link is a structural link or a hyperlink.

Layered Model of Subgraphs and Hypergraphs



Subgraphs and hypergraphs are similar in structure, but provide different functions.

A subgraph is composed of a set of nodes and a set of structural links. Both the source and target node of each link must also be contained in the subgraph.

A hypergraph is composed of a set of nodes and a set of hyperlinks. Both the source and target node of each link must also be contained in the hypergraph. Furthermore, every node must be either the source or target of at least one link in the hypergraph.

Figure 2:
Graph Server Layered Architecture

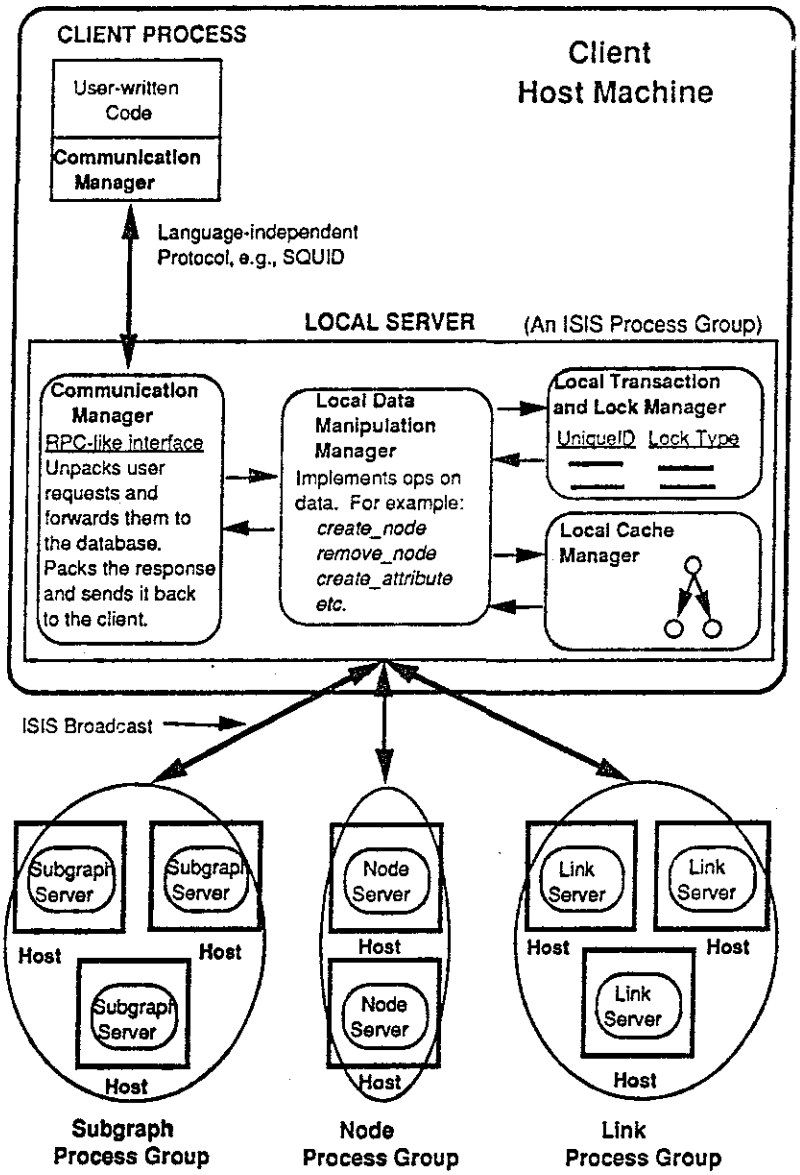


Figure 3:
Graph Server Distributed Architecture

The *local cache manager* maintains the local data structure with respect to a client program that is loaded from the data managment system and on which all operations are performed. It is an *intelligent* cache, in the sense that the system anticipates data likely to be needed by the end user and loads that information from the server into the local cache before it is requested. For example, when a given subgraph is loaded in response to some action by a user, the intelligent cache will also load, as a background operation, the associated hypergraph in anticipation that the user may soon wish to follow a hyperlink.

A *local transaction and lock manager* is used to maintain concurrency within the overall data managment system. A strict two-phase locking scheme is used. The transaction manager is responsible for obtaining all necessary locks during a transaction and releasing them after the transaction has completed.

Remote server processes, shown in Figure 3, are primarily responsible for storing data on disk. Each contains a transaction and lock manager and a storage manager. There are three types of remote server processes -- subgraphs, nodes, and links -- each responsible for a different portion of the data managment system. When a local server needs information from the data managment system, it broadcasts the request to the appropriate process group. The one that contains the requested information, determined by a unique id for the data objected requested, handles the request. When a local server commits a transaction, it again broadcasts a message to all applicable groups, and the appropriate remote servers handle the request. Additional servers within a process group are created automatically by the system under certain load and capacity conditions and in response to network conditions, including failures.

Related Research

To place the UNC Graph Server in context with other perspectives of hypermedia data models and management systems, one must consider the design space of possible decisions along several dimensions. First, data models can be differentiated according to their underlying assumptions as to the relationship between documents and graph structure. Second, storage and access requirements vary according to the nature of the data object provided to users. Third, systems differ as to whether they include the data store as an integral part of the application system or whether it is treated as a separate component or server. Fourth, they differ according to the specific details of the graph abstraction that underlies the model. A fifth dimension concerns the support platform on which the model is implemented.

Two points of view predominate regarding the relationship between document and graph. One perspective takes the position that the text (or other form of content) is fundamental and that the graph structure overlays the document, linking content in one location with content in another location. The other perspective views the graph structure as fundamental and views content as being contained within individual nodes. In this second model, links join nodes but may be extended by anchors into the contents of nodes. Examples of systems that have taken a document-oriented perspective include NLS [Englebart, et.al., 1973], Xanadu [Nelson, 1981], Intermedia [Meyrowitz, 1986], and MINOS [Christodoulakis, et.al., 1986]. A variation on this point of view are systems in which links are "virtual," implemented by fast text or content search algorithms. Examples include SuperBook [Remede, et.al., 1987] and Document Examiner [Walker, 1987]. Systems that have viewed the graph structure, rather than document content, as fundamental include NoteCards [Halasz, 1987], gIbis [Conklin, & Begeman, 1987], and WE [Smith, et.al, 1987]. Design decisions along this dimension result in fundamentally different data models and implementations, as can be seen, for example, in Xanadu's "tumbler notation" approach, Document Examiner's inverted/hashed keyword approach, and NoteCard's graph storage approach. The UNC Graph Server views the graph structure as fundamental.

A second distinction concerns the data object made available to users. Two approaches dominate. One approach retrieves and provides the user with a basic window of content. KMS presents single fixed-sized frames of information [Akscyn, et.al., 1988]; HyperCard retrieves and displays one card at a time [Apple Computer, Inc., 1989]. The alternative approach is to retrieve all or a substantial portion of the graph structure and present a representation of that structure for display and manipulation; NoteCards [Halasz, 1987], gIbis [Conklin & Begeman, 1987], Trellis [Furuta & Stotts, 1990], and WE [Smith, et.al., 1987] are all examples. The Graph Server is in this second camp, providing the application program with complete subgraphs through a single access. This distinction is important from the standpoint of performance for access and retrieval. For example, a one-second response time for a frame of information is acceptable but a fifty-second response to retrieve a relatively small fifty node graph is not.

Most hypermedia systems have been developed as closed architectures in which the data store was included as an integral component. The alternative is to adopt a server approach, as was done for the UNC Graph Server, in which the data store exists as a separate, stand-alone object on the communications network that can be accessed by independent hypermedia applications. Since most earlier systems have adopted integral architectures, we will not list examples. Notable exceptions, that have adopted separable stores, include HAM [Campbell & Goodman, 1988], GRAS [Brandes & Lewerentz, 1985], and HyperBase [Schutt & Streitz, 1990].

A fourth distinction concerns the formal properties of the graph model underlying the data model supported by the store. This dimension is more varied, and extends from actual systems to abstract designs and proposed standards. Several systems have adopted data models that are basically hierarchical, sometimes with secondary links across the hierarchy. Examples include NLS [Engelbart, et.al., 1973] and Xanadu [Nelson, 1981]. Most have adopted some form of general directed graph model, but they differ in details and extensions. Delisle and Schwartz [1987] propose contexts for use in collaborative environments to organize information and to reduce user interference by partitioning information into collections of nodes and links which may be related by cross-context links. HyperBase defines complex objects which represent collections of nodes and links [Schutt & Streitz, 1990]. Garg's set theoretical model, which has not yet been implemented, defines the aggregation, generalization, and revision abstractions for the management of information with different views and granularity [Garg, 1988]. The Trellis hypertext model is based on a Petri net model which specifies information structure and content, as well as browsing and execution semantics [Furuta & Stotts, 1990]. The Strawman reference model describes a system architecture characterized by advanced features such as multi-user and distributed data support, computational completeness, and versioning [Thompson, 1990]. Van Dyke Parunak's reference model [1990] identifies the salient features of a hypermedia data model: nodes are characterized by contents, type, and structure; links are characterized by directionality, topology, type, anchors, and nodes. The Dexter reference model [Halasz & Schwartz, 1990] provides a formal definition of hypermedia abstractions consisting of a network of components and interconnecting links; components are the basic unit which represent generic placeholders for hypermedia contents. Lange's model [1990] defines nodes and links as the basic units, with abstractions for aggregation and node substructure. The UNC Graph Server is unusual in its emphasis on typed, composite subgraphs and its distinction between structural links and hyperlinks, the second resulting in hypergraphs. It is perhaps closest to the Dexter model, but the Graph Server provides additional layers of abstraction not defined in that model, and it differs in its perspective with respect to anchored links. For a more detailed examination of similarities and differences with respect to basic data models, see [Boone, 1991].

The fifth difference concerns basic implementation strategy. There seems to be strong agreement in the field regarding the desirability to buy, rather than build, the underlying storage management platform on which hypermedia data management systems are built. Three basic approaches predominate. Many projects have built their own stores. Regrettably, the UNC Graph Store is among that group, although our current implementation effort is using a high-level distributed system support package, ISIS, as a base. We still hope to purchase a complete object-

oriented platform in the future. Perhaps the most wide-spread approach has been to use a B-tree utility or similar subsystem; Hyperties [Schneiderman, 1987] is an example of such a system. The third approach has been to use a standard relational database utility. Systems that have gone this route include Intermedia [Meyrowitz, 1986] and HyperBase [Schutt & Streitz, 1990]. The problem posed by this choice is performance and the inherent awkwardness of implementing graph structures within the relational paradigm. There is strong hope that emerging object-oriented databases will provide suitable platforms. We share this hope, but do not expect near-term help. These systems, first, require considerable more development effort to implement data models than is required to build systems on top of traditional database utilities; and, second, performance is likely to remain a problem for some time. (Our experiments conducted just over a year ago found performance penalties, relative to SmallTalk, of 10x to 35x for one such commercial object-oriented database system.)

Conclusion

In describing the UNC Graph Server, we have discussed both the data model it supports and the architecture of the system. We have emphasized what we believe to be the fundamental problem that must be confronted by hypermedia data management systems if they are to scale up to support large industrial applications -- that they be partitionable in natural ways both for system access and human comprehension. We have also identified a number of other technical issues, that include an open, extensible architecture with respect to application programs; composite nodes and data objects; information hiding and user comprehension; and the integrity, consistency, and completeness of the artifact as a whole. We have produced a working prototype server, and we are in the process of completing a second version of the data model. We expect to complete by the end of the year a second prototype that implements the distributed architecture described here. By describing the development path we have actually followed, we hope to contribute to a growing body of knowledge about what does not work as well as what does.

Acknowledgments

A number of organizations and individuals have contributed to this research. Major support has come from NSF (Grant # IRI-9015443) and the IBM Corporation (SUR Agreement # 866), with additional support from ONR (Contract # N00014-86-K-00680). We are grateful to all of our faculty and student colleagues on this project who have contributed to the common body of ideas in which we work. Especially important have been the contributions of Don Smith, Rick Snodgrass, Mike Wagner, Zhenxin Wang, and John Hilgedick.

References

Akscyn, R.M.; McCracken, D.L.; & Yoder, E.A. (1988). KMS: A distributed hypermedia for managing knowledge in organizations. *Communications of the ACM*, 31(7), 820-835.

Apple Computer, Inc., (1989). *HyperCard User's Guide* Cupertino, CA: Apple Computer, Inc.

Birman, K.P.; & Joseph, T.A. (1987). Exploiting virtual synchrony in distributed systems. *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, (Austin, Texas), pp. 123-138

Boone, J. (1991). *A Survey of Data Models for Hypermedia.* Chapel Hill, NC: UNC Department of Computer Scieence Technical Report # 91-022.

Brandes, T.; & Lewerentz, C. (1985). GRAS: A nonstandard data base system within a proramming support environment. *Proceedings of the GTE Workshop on Software Engineering Environments for Programming in the Large*, pp. 113-121.

Bush, V. (1945). As we may think. *Atlantic Monthly*, 176(1), 101-108.

Campbell, B.; & Goodman, J.M. (1988). HAM: A general purpose hypertext abstract machine. *Communications of the ACM*, 31(7), 856-861.

Christodoulakis, S.; Theodoridou, M.; Ho, F.; Papa, M.; & Pathria A. (1986). Multimedia document presentation, information extraction, and document formation in MINOS: A model and a system. *ACM Transactions on Office Information System,* 4(4), 345-383.

Conklin, J.; & Begeman, M. L. (1987). gIBIS: A hypertext tool for team design deliberation. *Proceedings of Hypertext '87*, pp. 247-252.

Delisle, N.; & Schwartz, M. (1986). *Neptune: A Hypertext System for CAD Applications*. Beaverton, Oregon: Tektronix Computer Research Laboratory Technical Report # CR-85-50.

Delisle N.; & Schwartz, M. (1987). Contexts-A partitioning concept for hypertext, *ACM Transactions on Office Information System*, Vol, 5(2), 168-186.

Engelbart, D.C. (1963). A conceptual framework for the augmentation of man's intellect, *Vistas in Information Handling* P.D. Howerton and D.C. Weeks (editors), Spartan Books, Washington, D.C., 1-29.

Engelbart, D.C.; Watson, R.W.; & Norton, J.C. (1973). The augmented knowledge workshop, *AFIPS Conference Proceedings,* 1973 National Computer Conference and Exposition (June 4-8, 1973 New York, New York), 9-21.

Furuta, R.; & Stotts, P.D. (1990). The trellis hypertext reference model. Proceedings *of the Hypertext Standardization Workshop* (Gaithersburg, Maryland), 83-93.

Garg, P.K. (1988). Abstraction mechanisms in hypertext, *Communications of the ACM*, Vol. 31(7), 862-870.

Halasz, F.; & Schwartz, M. (1990). The dexter hypertext reference model. Proceedings *of the Hypertext Standardization Workshop* (Gaithersburg, Maryland), 1-39.

Halasz, F. (1987). *Hypertext 87 Proceedings.*

Lange, D. (1990). A formal model of hypertext. Proceedings *of the Hypertext Standardization Workshop*, (Gaithersburg, Maryland), 145-166.

Meyrowitz, N. (1986). Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. Proceedings *of the Conference on Object-Oriented Programming System, Languages, and Applications (OOPSLA '86)* (September 29-October 2, Portland, Oregon).

Nelson, T.H. (1981). *Literary Machines.* Swarthmore, PA: Available from the author.

Remede, J.R.; Gomez, L.M.; & Landauer, T.K. (1987). SuperBook: An automated tool for information exploration -- Hypertext? *Hypertext 87 Proceedings.*, pp. 175-188.

Schneiderman, B. (1987). Userinterface design for the hyperties electronic encyclopedia. *Hypertext 87 Proceedings.*, pp. 189-194.

Schutt, H.; & Streitz, N. HyperBase: A hypermedia engine based on a relational database management system, Integrated Publication and Information Systems Institute, West Germany, submitted to ECHT '90.

Smith, J.B.; & Smith, F. D. (1991). *ABC: A Hypermedia System for Artifact-Based Collaboration.* Chapel Hill, NC: UNC Department of Computer Scieence Technical Report # 91-021.

Thompson, C. (1990). Strawman reference model for hypermedia systems. Proceedings *of the Hypertext Standardization Workshop*, (Gaithersburg, Maryland), pp. 223-246.
Tomp89

Parunak, H. V. D. (1990). Toward a reference model for hypermedia. Proceedings *of the Hypertext Standardization Workshop* (Gaithersburg, Maryland), pp. 197-211.

Walker, J. (1987). Document Examiner: Delivery interface for hypertext documents. *Hypertext 87 Proceedings.*, pp. 307-323.