Architecture of the Artifact-Based Collaboration System Matrix

K. Jeffay, J.K. Lin, J. Menges, F.D. Smith, J.B. Smith

University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC, USA 27599-3175
(919) 962-1700 {*jeffay,linjk,menges,smithfd,jbs*}*@cs.unc.edu*

ABSTRACT The UNC Collaboratory project is concerned with both the process of collaboration and with computer systems to support that process. Here, we describe a component of the Artifact-Based Collaboration (*ABC*) system, called the *Matrix*, that provides an infrastructure in which existing single-user applications can be incorporated with few, if any, changes and used collaboratively. We take the position that what is needed is not new tools but better infrastructure for using familiar single-user tools collectively. The paper discusses the Matrix architecture, a Virtual Screen component, and generic functions that provide conferencing, hyperlinking, and recording of users' actions for all applications.

INTRODUCTION AND MOTIVATION

Our research is concerned with both the process of collaboration and with computer systems to support that process. We focus on shared intellectual activity as required, for example, in system design and other similar tasks, in which groups of scientific and technical professionals work together to build a large, complex structures of ideas. Further, we assume that these groups are geographically distributed, interact with one-another using communications networks, and that they normally produce some form of tangible *artifact* as the goal of their work.

Our project is attempting to address the fundamental issues of collaboration in a comprehensive way. First, we are a multidisciplinary group including cognitive psychologists, anthropologists, and computer scientists that is conducting studies of collaborating groups and developing theories of the collaborative process. A study of four software development groups is described in [6]; a discussion of Collective Intelligence as the basis for a process model of collaboration is presented in [12]. Second, we are building a collaboration-support system called the Artifact-Based Collaboration (*ABC*) system that supports both *asynchronous* work — individuals working alone on different parts of a project — and *synchronous* work — people, perhaps geographically distributed, working together on the same part of the artifact at the same time through high-speed communications networks. An overview of the *ABC* system appears in [13].

We have taken as our driving problem the design of software systems. Brooks has described the essence of software design as follows:

> The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions. The essence is abstract in that such a conceptual construct is the same under many different representations. It is nonetheless highly precise and richly detailed. I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. [2]

We agree with Brooks but go further to suggest that the problem of building large "conceptual constructs" is *fundamental* not just for software design, but for all large collaborative projects. For software systems, this construct typically consists of concept papers, architecture documents, requirements, specifications, programs, diagrams, user documentation, and maintenance manuals, as well as various administrative documents. Similar sets of materials are produced by other types of collaboration, such as designing aircraft, planning projects, or defining military doctrine. Some materials represent the final goal of the project; others assist the development process. However, if the work of the group is to have integrity, then this entire conceptual construct — including instrumental as well as target products — must be consistent, coherent, and correct. To emphasize this requirement, we consider the entire construct to be a single artifact that is developed and maintained as a whole.

For organizing the group's shared artifact, we have adopted a hypermedia data model; its hyperlinking capabilities allow

This work supported in parts by the National Science Foundation (Grant # IRI-9015443), the IBM Corporation, and the Digital Equipment Corporation.

users to explicitly denote dependencies within the artifact so that when changes are made in one place, users (or automated processes) can follow these links to other places to verify or modify those parts of the artifact, accordingly. We have also developed a small group of browsers that allow users to navigate through the artifact, to visualize its structure, and to reorganize it; these tools help users comprehend the artifact and provide a sense of context.

Our system design is based on our hypothesis that what is most needed is not new tools for collaboration, but better infrastructure to support collaborative use of existing tools. We believe our system must allow users to incorporate, with minimal effort, familiar single-user applications of their choice, such as editors, drawing tools, spreadsheets, CAD/CAM programs, *etc.* Users should be able to use for group-related work — in both asynchronous and synchronous modes — the same computer tools they are accustomed to using as individuals. Consequently, the emphasis in the rest of this discussion is devoted to describing an architecture in which much of the large base of existing software can be incorporated for collaborative use.

This approach is practical only if existing single-user applications can be incorporated with few, if any, modifications to those programs. To meet this requirement, we are developing a set of generic functions that can be attached to existing applications. These functions include *hyperlinking* between *anchored* points within different applications; a shared-workspace *conferencing* function that can be invoked from any application or group of applications; and tools for *recording* and *recreating* the behaviors of group members for studying the collaboration process from a human point of view.

The architecture of the *ABC* system can be divided into three large components, as shown in Figure 1. A Distributed Graph Storage System (DGS) [11], provides the hypermedia data model implemented in a distributed architecture. A set of browsers and an extensible set of existing application programs provide tools for the user. A third component, shown in the middle of the figure, launches browsers and applications, joins them to the storage system and to one-another, and provides a set of generic collaboration functions that apply to all tools. We call this component the *ABC* System *Matrix* — in the geological sense of a surrounding context in which objects are embedded; in this case, the objects are browsers and applications.

In following discussion, we will focus on the architecture of this encompassing matrix. Section 2 describes the user's mental model of the system. Section 3 provides an overview of the architecture. After that, individual components are described that support generic collaboration functions. Section 4 describes the Generic Function Manager. Section 5 describes a conferencing facility available to all applications within the system. Section 6 describes a general hyperlinking and anchoring component. Section 7 relates this work to other systems and other research. Section 8 provides a summary and discusses our future plans.

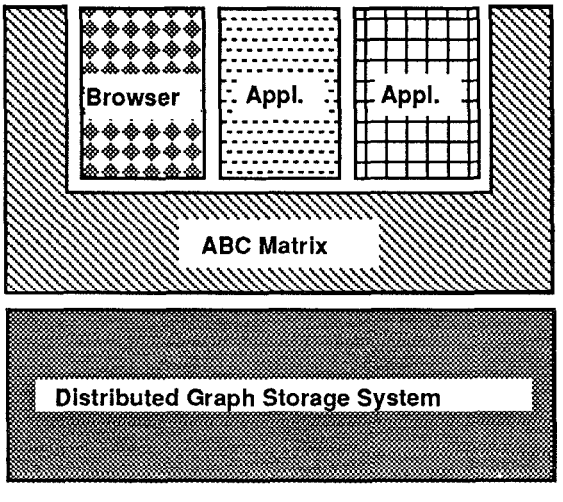


Figure 1. ABC system components.

USER'S MENTAL MODEL

Here we describe the visual appearance of the *ABC* system and the user's mental model of it. Much of what gives *ABC* its particular character as a system derives from its hypermedia data model, supported by the underlying DGS and presented to the user through a set of browsers. Consequently, we begin the discussion with the data model and then discuss other components the user sees and works with.

Data Model. The basic model for the artifact is a collection of separate *graphs*, each consisting of a set of nodes and a set of links that denote structural relationships between nodes. A node's content can consist of a block of data — such as a conventional file — or it may be another graph. By storing individual graphs as the content of nodes, graphs can be composed to form a structure that can be viewed both as a single large, integrated artifact but also as separate objects that permit multiple users to work concurrently. *ABC* supports several different graph types, including trees, lists, general directed graphs, etc. Finally, a special type of link, called a hyperlink, is provided to define semantic relationships that would violate the integrity of a graph type if they were denoted by structural links (*e.g.*, a relationship between siblings in a tree) and to join separate graph structures.

Virtual Screen Environment. The *ABC* system runs within the X Window System environment under the UNIX operating system. Because the underlying *ABC* storage system is based on a hypermedia model instead of a conventional file system model, it is important to signal the user that a given window or application is referencing the *ABC* storage system versus the UNIX file system. We elected to do this by providing a separate contiguous area within the overall X display for the *ABC* system and its associated browsers and applications rather than marking individual *ABC* windows and permitting them to be intermingled with conventional X windows. Thus, users have a sense of "entering" and "leaving" the *ABC* environment, which they do simply by moving the mouse from one area of the screen to another. We call this environment a Virtual

Screen because it resembles the entire screen in appearance; it can contain multiple windows within its borders, as shown in Figure 2; and users may use within it any X window manager they choose, including one that is different from that for the rest of the screen [9].

Browsers and Applications. Within the *ABC* virtual screen, individual tools display output as X windows. A collection of some half-dozen browsers written by our project are provided for developing, navigating, and accessing individual graph structures within the artifact. A set of familiar applications, not written by our project, are provided for creating and editing conventional forms of data, such as text, diagrams, spreadsheets, etc., stored as node content in the DGS. This set of applications is open-ended. Thus, users may bring into the *ABC* environment familiar tools of their choice that can run without modification so long as hyperlink anchoring capabilities (explained in a later section) are not required. If anchoring is needed, the application program must be modified; however, we have tried to minimize these changes by providing a generic anchor-support toolkit. Both browsers and applications are opened on the content of nodes, and the graph or data object they produce is viewed by the user as being stored "inside" the node.

Generic Function Management. Across each *ABC* window is a menu bar that provides access to an additional set of functions common to all *ABC* browsers and applications. The functions accessed through this bar, which we call a Generic Function Manager, include starting, stopping, and anchoring hyperlinks, and sharing an application or browser in a conference with other users. The same underlying architecture that supports generic functions also supports the tracking and replay functions mentioned above.

ARCHITECTURAL OVERVIEW

The design of the *ABC* system has been driven by two general principles. The first is our desire to support synchronous collaboration over networked computer systems. This lead to our adoption of the X Window System as the software platform for the *ABC* system. Standardizing on X also allows us to maximize the number of hardware and software platforms that can support the *ABC* environment (given the constraints of a moderate sized university research project). Although all of our development efforts to date have taken place on UNIX workstations, the *ABC* system could, in principle, run on any computer that supports X. For example, we have demonstrated the conferencing portion of the *ABC* system (the shared windows) between an Apple Macintosh and a UNIX workstation.

The second principle is the desire to integrate as many existing applications (*e.g.*, editors, drawing programs, text processing utilities) as possible into the *ABC* system. This allows users to retain significant portions of their traditional working environment while using the *ABC* system. There are two levels of integration within the *ABC* system. At a minimum, any X application can be launched and used within the *ABC* system. By virtue of executing within the *ABC* system, such an application is automatically capable of being conferenced and having user interactions recorded for later study. Moreover, one can create and follow node-level hyperlinks from or to unmodified applications. In this manner, the *ABC* system adds powerful collaboration functions to existing single-user applications. Existing applications that operate on byte streams (*e.g.*, UNIX files) can also be used to create and modify the contents of nodes in the DGS. (Any temporary or auxiliary files needed by an application reside outside the DGS.) A second data integration function, allowing the contents of UNIX files to be imported into a DGS object, is also provided. X cut/copy/paste functions can also be used to move data into and out of applications.

The second level of integration is the integration of anchoring functions (*e.g.*, *create anchor, delete anchor, etc.*) into existing applications. This requires modification and recompilation of the application. The extent of the modifications is a function of the complexity of the application. Hyperlinking and anchoring are discussed in more detail below.

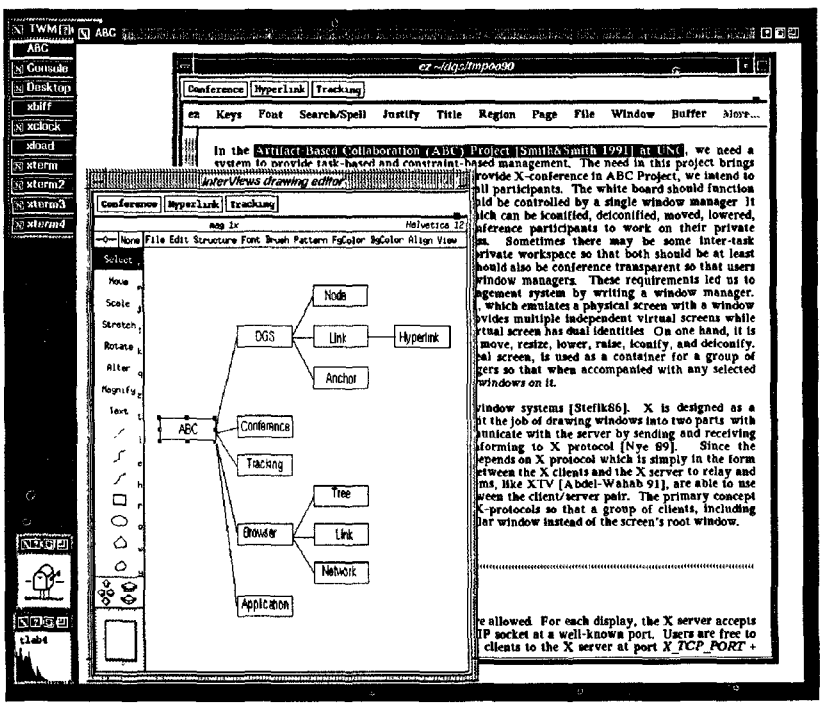


Figure 2. An ABC Virtual Screen and Generic Function Manager.

Figure 3 shows the basic structure of the *ABC* System Matrix. It can be thought of as a pipeline of processes that filter streams of X Window System protocol data. At the highest level, the Matrix itself exports an X server interface to applications (*e.g.*, browsers, editors, *etc.*) and an X client interface to an X server. Therefore the Matrix can be viewed as an X

"pseudo-server." All applications manipulate their visual interface indirectly through the Matrix. Similarly, the X server interacts with user applications indirectly through the Matrix. If applications are "*ABC* system aware," such as the browsers we have constructed, they also interact with the graph server. For example, screen management for the *ABC* system, that is, the placement of windows within the larger, encompassing, *ABC* system window is the responsibility of the Virtual Screen component of the Matrix described in Section 2. The Virtual Screen is an X protocol filter, *i.e.*, it intercepts and modifies X protocol data sent between an X client used in the *ABC* system and an X server. The modifications re-parent clients to the virtual screen instead of the root window of the workstation console. (The Virtual Screen is simply a window as far as the X server is concerned.) As is the case with all our protocol filters, the Virtual Screen filter modifies very few of the messages (typically less than 10%) that are sent between an application and an X server.

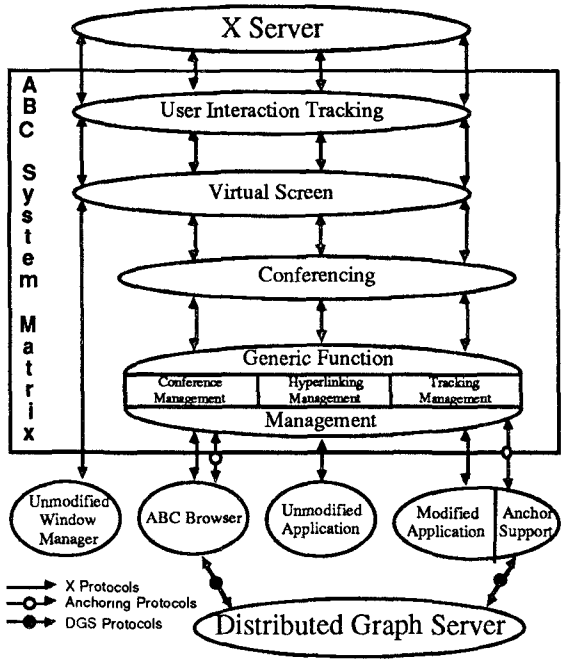


Figure 3. Architecture of the *ABC* System Matrix.

Starting at a user application and working towards an X server, an application's X protocol data stream is first processed by the *ABC Generic Function Manager* (GFM). This is a program that the user interacts with to create conferences, or to create or manipulate hyperlinks. The output from the GFM (a valid X client protocol stream) is processed by a second protocol filter that implements conferencing. A third filter implements the user interaction protocol recording. A final filter is the Virtual Screen. The filters that implement the conferencing and screen management are described in greater detail in separate sections below. The filter that implements user interaction protocol recording is similar to the conferencing filter. The implementation of anchored hyperlinks is distributed between individual user level applications and the GFM. This is also discussed below.

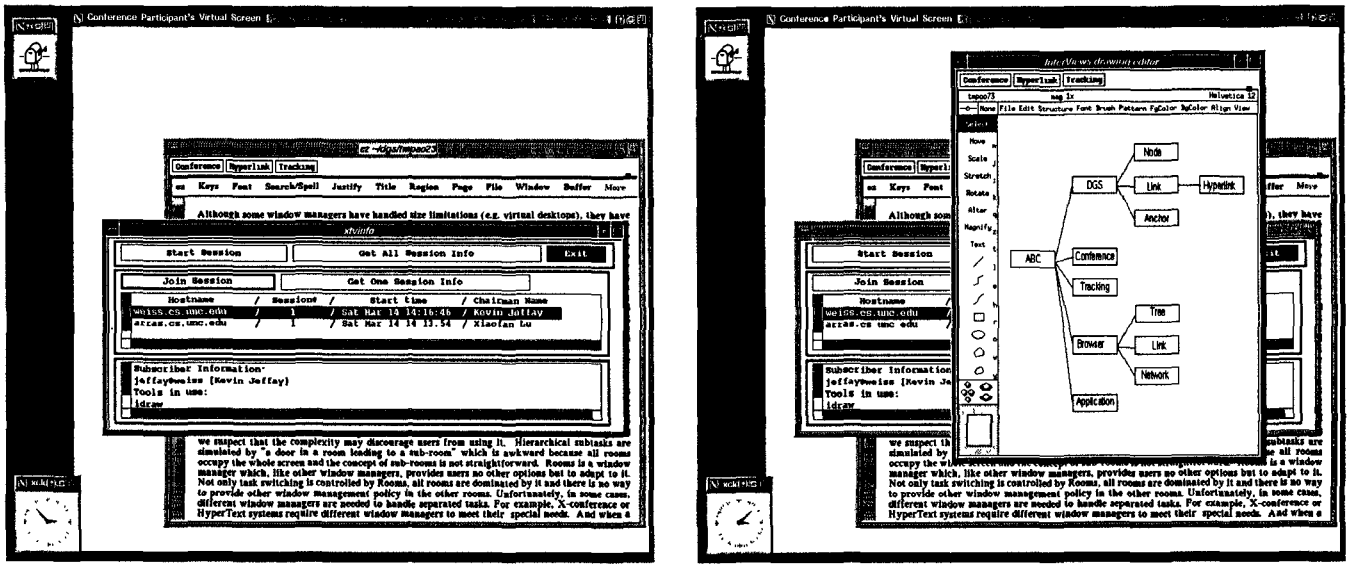
GENERIC COLLABORATION FUNCTION MANAGEMENT

Any application that is used within the *ABC* environment (*i.e.*, within a Virtual Screen) is transparently endowed with conferencing, hyperlinking, and user interaction recording functions. By modifying applications, anchoring function can also be added. These functions are referred to as *generic collaboration functions*. These functions are invoked and controlled via an entity called the *Generic Function Manager* (GFM). To the user, the GFM appears as a title bar that resides between each top level window of an application and the associated window manager's title bar as shown in Figure 2. Our title bar is created and managed by the GFM, in the same way that the window manager's title bar is created and managed by the window manager. The GFM "re-parents" the application's window so that the application window and the generic function buttons appear to the user as a single coherent window. The GFM maintains one connection (*e.g.*, one socket) for each application displayed within the virtual screen. Indeed, for each application, the X server displaying the Virtual Screen windows considers the GFM/application pair as a single X client with multiple subwindows (the GFM title bar and the application window(s)). The appearance and interaction of the GFM title bar is customizable on a per-user basis. That is, how the user invokes various operations and how feedback is displayed are not prescribed, but can be defined in a GFM initialization file. We do, however, provide reasonable defaults. The GFM is capable of managing buttons, pop-up and pull-down menus, dialogue boxes, miniature icons, *etc.* The GFM provides a framework for extending the an application's capabilities without disturbing its user interface.

CONFERENCING

Conferencing refers to the use of an application by multiple users simultaneously. In the Matrix, conferencing is implemented by (1) distributing the visual interface of an application to one or more remote workstations thereby allowing remote users to view the interactions of other users with the applications, and (2) providing input paths from each conferee back to the application so that any participant in the conference may interact with the application. Since all applications we currently use within the *ABC* system are single user applications, it is not possible in general to allow more than one user to interact with a conferenced application at any one time. The Matrix supports a simple protocol for negotiating access to a conferenced application. The basic paradigm of group input to a conferenced application is that of "passing the keyboard." The model is one of cooperative collaboration wherein users might gather in an office and take turns interacting with an application by passing the keyboard among themselves. While this paradigm is certainly not a panacea for group conferencing, it is necessitated by our desire to work with existing single user applications. The conferencing subsystem requires no modifications to applications. The conferencing subsystem is based on our earlier work on a stand-alone conferencing system [1, 3].

One X protocol filter in the Matrix is dedicated to implementing conferencing. The filter records every message an application sends to an X server that effects the state of the



The idraw window in Figure 2 has been placed into a conference. A remote user (left) joins the conference by contacting an *ABC* conference server via a Matrix-provided conference management application and selecting a conference from a menu of on-going conferences. An exact copy of each window of each application in the conference then appears on the remote user's virtual screen (right).

Figure 4. An example of joining a conference.

internal data structures the server maintains for the application's visual interface. (In more technical X terms, the filter records all X resources created by the application.) When an application is used in non-conferenced mode (the common case), the conferencing protocol filter passively records salient X information that is later used to create a copy of the application's visual interface on a remote machine.

A conference is initiated when a user presses the conference button on the GFM menu bar for an application. The GFM then contacts a global conference server and registers the user's name, unique machine identifier (*e.g.*, its internet address), and the name of the application in the conference. The user can then invite other users to act as participants in the conference by notifying them of the existence of this conference. Participants join the conference as shown in Figure 4. In the present system there are no access control mechanisms for joining conferences. The initiator of a conference may, however, eject a participant from a conference at any time.

When a remote participant joins a conference, the remote participant's conference management application creates a process on the participant's workstation that acts as a surrogate for the conferenced application as shown in Figure 5. The surrogate application is responsible for maintaining a visual image of the conferenced application on the participant's display that is consistent with the view on the conference originator's workstation. When an application is placed in a conference, it is actually conferenced as an GFM/application pair. This is in keeping with our "passing the keyboard" paradigm of interaction. By including the GFM in the conference, remote participants can create, for example, hyperlinks to and from data object viewed in conferenced applications. A network connection is established between the remote surrogate process and the conference originator's protocol filter and all X messages sent by the application are distributed to all surrogate processes. In addition, the original X filter and the surrogate process speak a Matrix conference control protocol that negotiates access to the input path back to the application from remote users.

Two paradigms of conferences are supported in the *ABC* system. The first is conferencing of individual applications (described above). The second is conferencing of Virtual Screens. The latter is useful for providing a coherent visual context for conferences that involve multiple applications. Conferences in the *ABC* system are "lightweight" in the sense that they can be started at any time. For example, if a user is editing a document and would like to review a section with a colleague, she could initiate a conference around her current editing session, independent of how long she had been using the application in non-conference mode. When the

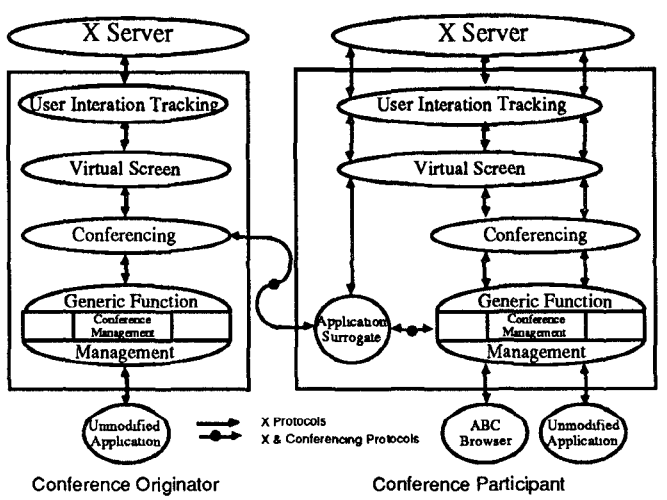


Figure 5. Conference architecture.

colleague joins the conference, a window will appear on their workstation that is identical to that of the initial user's.

In order for conferences to be effective, higher bandwidth communication media than simple "talk" windows are required. We are interested in providing integrated digital voice and video communication links to the workstation and are pursuing this in a parallel research effort [7]. For high-bandwidth communication in the present system we use an in-house broadband CATV system.

HYPERLINKING AND ANCHORING

One of our major goals is to provide a mechanism for adding anchored hyperlink capabilities to existing X applications. A hyperlink in the *ABC* system is a directed reference from a node in one graph to a node in the same or another graph. Either the source or the destination endpoint (or both) of a hyperlink may be unanchored (in which case the endpoint refers to a node as a whole) or anchored (in which case it refers to some portion of the content of a node). For example, suppose $node_1$ is a text document and $node_2$ is a drawing. If a hyperlink relates a paragraph in $node_1$ to the entire drawing represented by $node_2$, the source of that hyperlink is anchored to the paragraph, and the destination is unanchored [11]. Examples of hyperlinks and anchors are presented in Figure 6.

The implementation of anchors for hyperlinks requires application source code modifications. We wish to minimize this effort by keeping per-application code changes small and reducing the need to understand the implementation of each application in detail. We also want to make it possible to invoke hyperlink operations without changing the user interfaces of existing applications and window managers (*e.g.*, key and mouse bindings). This minimizes interference with the user's established work habits. Moreover it enables hyperlink functions to be invoked similarly for all applications. In order to make anchors fit the semantics of each application, we do not prescribe how any particular application should define or display anchors, or how the user should specify them.

Hyperlinking operations are invoked via buttons on the GFM titlebar. Unlike conferencing, hyperlinking functions require interactions between the user application (the GFM client) and the GFM. For example, for the *Create Hyperlink* function the GFM should just notify the client that the *Create Hyperlink* operation was invoked and wait for feedback on the success or failure of the operation; it then displays the feedback as the user-defined means of invoking the function and user-defined feedback specifications dictate. Other operations, *e.g.*, *Follow Hyperlink*, might require the GFM to ask the client for information to display in a menu or dialogue box; in this case a list of hyperlinks associated with the currently selected anchor. The user would then select one of the links and the GFM would tell the client which was selected and wait for feedback as before. A hyperlinking example is illustrated in Figure 7.

Most X-based applications already have some means of selecting and highlighting items to be operated upon; most likely these same items are appropriate anchors for the data manipulated by the application. In this case we do not need a new mechanism for specifying anchors; we only need to add new operations on selected data (*e.g.*, *Create Anchor*, *Follow Hyperlink*, *Modify Hyperlink*). These operations are invoked via the GFM as described above. It is the application's responsibility to maintain the values of anchors created by the user. Unfortunately, existing mechanisms for selecting items typically assume that such selections are not persistent; that is, there is only one selection and it can be discarded when the data is edited or the application exited. Anchors, however, must be persistent. Their values must remain constant (or be modified in intuitively appropriate ways) when the data is edited, and they must be maintained between application invocations. This is the area in which the application implementation must be understood in the

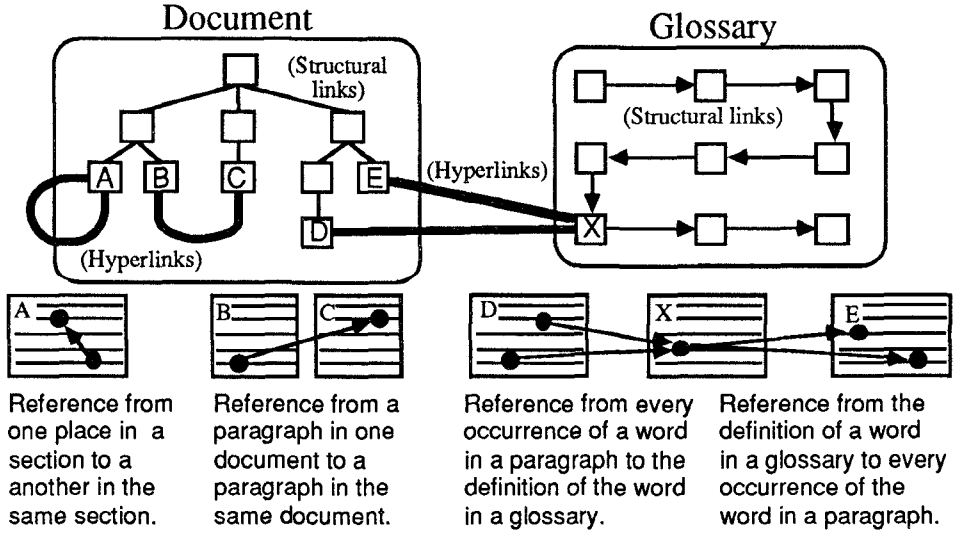
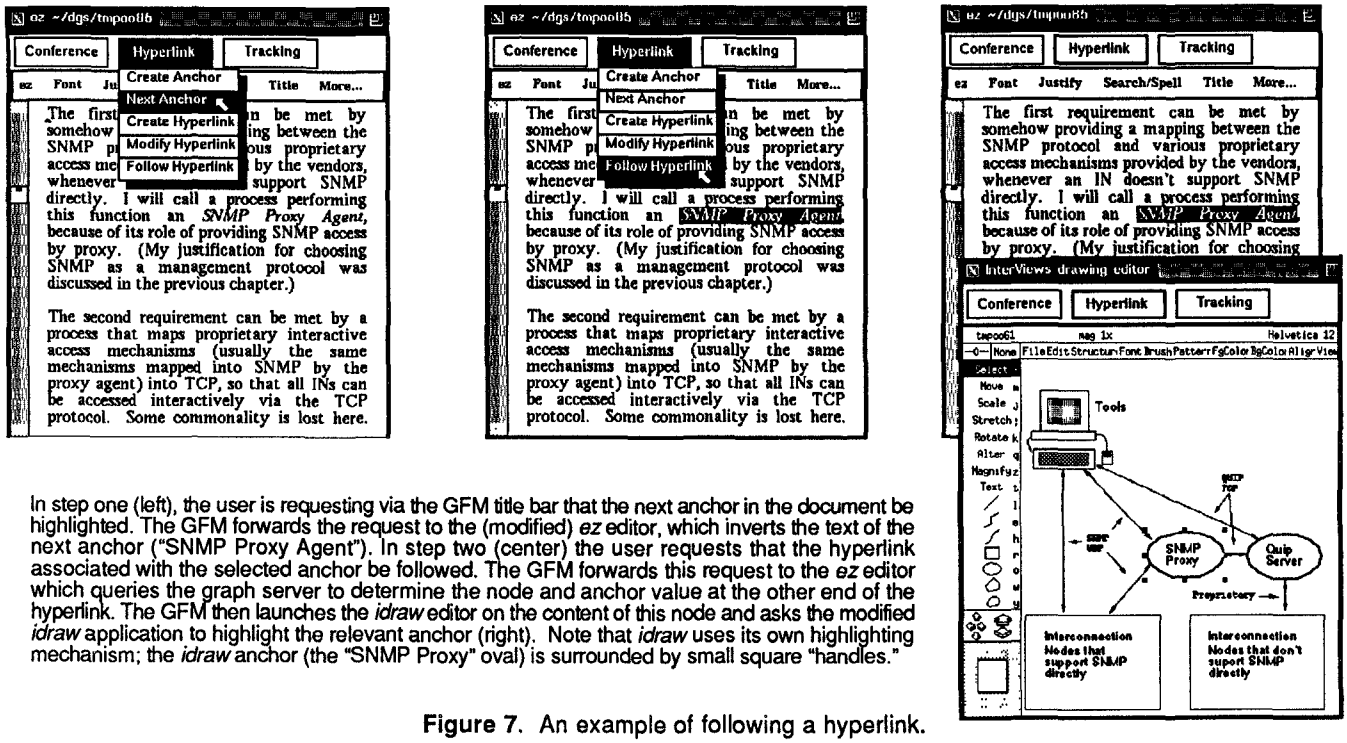


Figure 6. Examples of anchored hyperlinks in and between graphs.



In step one (left), the user is requesting via the GFM title bar that the next anchor in the document be highlighted. The GFM forwards the request to the (modified) *ez* editor, which inverts the text of the next anchor ("SNMP Proxy Agent"). In step two (center) the user requests that the hyperlink associated with the selected anchor be followed. The GFM forwards this request to the *ez* editor which queries the graph server to determine the node and anchor value at the other end of the hyperlink. The GFM then launches the *idraw* editor on the content of this node and asks the modified *idraw* application to highlight the relevant anchor (right). Note that *idraw* uses its own highlighting mechanism; the *idraw* anchor (the "SNMP Proxy" oval) is surrounded by small square "handles."

Figure 7. An example of following a hyperlink.

most detail and where the most code changes must be made. One must either modify the application's internal data structures to store anchor values, or maintain an anchor table separate from the application's internal data structures and modify all editing operations so that they update this table appropriately.

Creating and modifying hyperlinks requires communication between multiple applications. The entity performing the hyperlink operation must gather information as to which nodes and anchors are to be the endpoints of the hyperlink. We use the X selection mechanism to effect this communication. We define five selections: *SourceAnchor*, *SourceNode*, *HyperlinkID*, *DestNode*, and *DestAnchor* for hyperlink operations. Ownership of these selections is obtained by applications when the user performs the appropriate actions. For example, to create a hyperlink, the user would put application *A* in source mode, create or select an anchor (which grabs the *SourceAnchor* and *SourceNode* selections), put application *B* in destination mode, create or select an anchor (which grabs the *DestNode* and *DestAnchor* selections), and invoke *CreateHyperlink*. The entity creating the hyperlink would request these four selections and send the *CreateHyperlink* request to the graph server. The *HyperlinkID* selection is then grabbed by the entity creating the hyperlink for use in subsequent *ModifyHyperlink* operations. Splitting Anchor selections from Node selections simplifies the creation of unanchored hyperlinks and adding or deleting anchors to or from existing hyperlinks.

In summary, existing applications must be modified to maintain anchors, communicate with the GFM to provide a user interface to new operations, communicate with the graph server to effect hyperlink operations, and communicate with other X clients via hyperlink X selections. With the exception of anchor maintenance, these code modifications can be made by simply inserting calls to library routines at the appropriate places. We also expect to be able to facilitate anchor maintenance by providing a set of library routines to perform common functions.

RELATED WORK

The collaboration-support environment based on Suite [5] focuses on flexible methods for coupling users' views of shared objects. Where most shared-workspace systems strictly replicate an application's windows for all users, the Suite architecture allows each view to be tailored in several dimensions. Views may be customized to specify which values in shared windows are coupled (change when the underlying object changes), how "committed" a value must be before it is used to update a view, when and in which views changes are reflected, how formatting may differ among views, and which window elements (*e.g.*, scrollbars, menus) are coupled. Applications (*e.g.* text or line drawing editors) must be written as multi-user programs using the Suite environment.

Rendezvous [10] is another framework specifically designed for creating multi-user programs. A rule and constraint system provided in an underlying UIMS is used to control three major aspects of sharing: how underlying objects are shared, how views of objects are shared, and how input (update) access to objects is shared. These appear to be similar in spirit to Suite facilities (but perhaps less flexible). Rendezvous also provides a session (conference) manager for dynamic creation, joining, and leaving a multi-user application. Another framework for creating shared multi-user applications is MMConf [4]. MMConf focuses

on conferencing issues such as conference management, input access (floor control), and distribution of shared data files. No functions for tailoring views are provided. "Conference-aware" applications are created using the MMConf toolkit for these functions.

On the surface, ConversationBuilder [8] has greater similarity to *ABC* — it supports artifacts with hypertext storage and browsers, can accommodate conventional editors such as Epoch and idraw, and does tracking of user interactions. It represents, however, a fundamentally different approach to collaboration support. ConversationBuilder is best understood as a framework for creating a coordination system using protocols based on speech/act theories. Protocols are templates that define roadmaps for various interactions among group members. For example, process protocols coordinate activities and checkpoints in a goal-oriented process such as joint paper writing; discussion protocols can be invoked to resolve questions or issues; and transient protocols can be used for simple ad hoc coordination.

Suite, Rendezvous, and ConversationBuilder all take a strong role in structuring the user's working environment, either by view tailoring and control over sharing, or by activity coordination. In contrast, *ABC* takes a "hands off" approach; it provides a suite of tools useful in both synchronous and asynchronous activities by group members but takes no direct role in tailoring or coordinating user interactions. Unlike MMConf, Suite, and Rendezvous, the only toolkit needed in *ABC* is for extending conventional applications to maintain anchors for hyperlinks.

SUMMARY

The UNC Collaboratory Project is concerned with both the process of collaboration and with computer systems to support that process. Because so many of the problems associated with collaboration derive from the fundamental problem of constructing a coherent, integrated whole from a collection of individual contributions, we have attempted to build a comprehensive system that supports equally collective, synchronous work and individual, asynchronous work. In this discussion, we emphasized the second issue by focusing on the *ABC* System Matrix that provides an infrastructure in which an extensible set of browsers and existing single-user applications can be incorporated. We have purposely and with considerable effort taken a minimalist approach to make the system approachable. We were able to do this by capitalizing on the generality of the X Windows System architecture and by developing a set of generic functions for conferencing, hyperlinking, and tracking, that gracefully extend familiar application programs.

Our future agenda includes incorporating voice and, perhaps, video into the workstation, extensions to the set of generic functions, and increasing performance and robustness to support actual use studies.

REFERENCES

1. Abdel-Wahab, H. M. and Feit, M. A. XTV: A framework for sharing X Window clients in remote synchronous collaboration. In *Proceedings, IEEE Conference on Communications Software: Communications for Distributed Applications & Systems*, Chapel Hill, NC, April, 1991, pp. 159-167.
2. Brooks, F. P., Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20, 4 (April 1991), 10-19.
3. Chung, G., Jeffay, K., and Abdel-Wahab, H. Dynamic Participation in a Computer-based Conferencing System. *Computer Communications* (to appear).
4. Crowley, T., P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson. MMConf: An Infrastructure for Building Shared Multimedia Applications. In *Proceedings ACM CSCW'90 Conference*, October 1990, pp. 329-342.
5. Dewan, P. and R. Choudhary. Flexible Interface Coupling in a Collaborative System. In *Proceedings ACM CHI'91*, New Orleans, LA, April 1991, pp. 41-48.
6. Holland, D., Reeves, J. R., and Larme, A. The construction of intellectual work by programmers. Dept. of Computer Science Tech. Report 92-013 (March 1992), Univ. of North Carolina, Chapel Hill, NC, 27599.
7. Jeffay, K., Stone, D.L., and Smith, F.D. Kernel Support for Live Digital Audio and Video. *Computer Communications*, 16, 6 (July 1992), pp. 388-395.
8. Kaplan, S.M., A.M. Carroll, and K.J. MacGregor. Supporting Collaborative Processes with ConversationBuilder, *Proceedings ACM Conference on Organizational Computing Systems*, Atlanta GA, November 1991, pp. 69-79.
9. Lin, J.K. Virtual Screen: A Framework for Task Management, *The X Resource*, 1, 1 (Winter 1992), pp. 191-198.
10. Patterson, J.F., R.D. Hill, S.L. Rohall, and W.S. Meeks. Rendezvous: An Architecture for Synchronous Multi-User Applications, *Proceedings ACM CSCW'90 Conference*, October 1990, pp. 317-328.
11. Shackelford, D. E., Smith, J. B., and Smith, F. D. *A Distributed Graph Storage System for Artifacts in Collaboration.* Department of Computer Science Tech. Report 92-012 (March 1992), Univ. of North Carolina, Chapel Hill, NC, 27599.
12. Smith, J. B. *Collective Intelligence in Computer-Based Collaboration: An Introduction.* Dept. of Computer Science Tech. Report 92-011 (March 1992), Univ. of North Carolina, Chapel Hill, NC, 27599.
13. Smith, J.B., and Smith, F. D. ABC: A Hypermedia System for Artifact-Based Collaboration. *Proceedings Hypertext '91*, San Antonio, TX, December 1991, pp. 179-192.