Implicit Locking in the Ensemble Concurrent Object-Oriented Graphics Editor

R. E. Newman-Wolfe, M. L. Webb and M. Montes

Computer and Information Sciences Department
The University of Florida
Gainesville, FL 32611
(904) 392-1488
nemo@chameleon.cis.ufl.edu

ABSTRACT
Ensemble is an X-Windows based, object-oriented graphics editor based on the tgif graphics editor from UCLA. It relies on Unix* 4.3bsd sockets and can be used as a stand-alone program or as an application in the University of Florida's distributed conferencing system (DCS). It uses implicitly placed write locks for concurrency control, with locks placed when an object is selected and removed when it is deselected. Multiple users may read or edit a file concurrently, with all users receiving updates whenever a lock is removed. Pointers are shared by mutual consent, so that users may collaborate to the degree desired. Ensemble is a prototype lock-based approach to object-oriented concurrent graphics editing.

INTRODUCTION - CONCURRENT GRAPHICS EDITING
Traditional graphics editors are not collaboration aware: they cater to a single user editing one file at one time. These editors have no built-in control for voluntary or involuntary multiple invocations attempting to edit the same file. For example, if a file were edited simultaneously, only the changes made through one editor would affect the final version of the file: the changes saved first would be rendered ineffective as the last save would overwrite the first.

Some systems, such as SCCS [4] and RCS [29], support editors externally in preserving the integrity of files. Files need to be checked out through these systems before changes may take place. This method ensures that only one user may edit a file at one time. The CVS system [17] attempts to overcome this serial access optimistically by allowing overlapping file modifications, and RCS has merge capabilities, but these only work with text files. However, if close collaboration is desired, or if work must proceed in parallel, a collaboration-aware editor is needed.

Collaboration-aware editors provide mechanisms for more fine-grained, real-time sharing. Collaborative activities are generally not well structured, and greater bandwidth of communication is needed than is required for coordinative activities. The ability to sketch and gesture are highly desirable for communication of a wide variety of ideas; after all, "a picture is worth a thousand words." More important, recreating a picture from a verbal description is a tedious and error-prone activity. Even when the images can be transmitted the gestures that are used to focus attention are largely lost. Simultaneous access to the same canvas is often desirable, so shared graphics editors are useful for collaboration support.

This paper describes Ensemble, the concurrent, object-oriented graphics editor of the distributed conferencing system (DCS) at the University of Florida. DCS provides broad support for cooperative work, and includes several particular tools, including Ensemble [23, 24, 25, 31, 32]. Ensemble has been in use for the past two years. There are four significant contributions of Ensemble. It uses implicit locking in an object-oriented graphics environment, and gestures may be shared with filtering by both sender and receiver. It has mechanisms to provide concurrency control at the file level in two ways: collaboration groups are formed implicitly simply by accessing a file, and within DCS, access to the file must be made through Ensemble, so that there can be no collisions with edits made outside of Ensemble. Finally, the architecture of Ensemble is simple and efficient, and lends itself well to inclusion of heterogeneous interfaces to the same canvas.

We will first introduce issues in shared graphics editors, then describe some of the existing concurrent graphics editors. Ensemble is then described in detail, including its requirements, design, and implementation as well as notes on preliminary experience with Ensemble. We conclude with some observations on what succeeded and what did not with Ensemble, and where future work can take us.

ISSUES IN CONCURRENT GRAPHICS EDITING
There are many philosophical and technical issues in producing a shared graphics editor. One basic question is
the degree of collaboration awareness of the application. The reviews by Stefik et al. [28], Ahuja et al. [3], Crowley et al. [7] and Greenberg [14] are excellent, and Greif's book is a good source of seminal papers [16].

A good concurrent editing system should support features that provide the user with ease of use, good accessibility to editable objects, hidden seams, and ability to filter remote information. Ease of use is the responsibility of the user interface. Information concerning other collaborators, such as their identification and their roles in participation, should be available but unobtrusive. Users should not be hindered by the fact that the file is being shared unless it is necessary: they may be made aware of the collaboration but should not be taxed by its existence. In sum, sharing should be as transparent as possible to the user, yet details of that sharing should be available if desired.

Drawing, gesturing and viewing may be separated from one another in terms of sharing. It is possible that only one user may draw at a time, while any user may gesture (move a cursor around the canvas); likewise, different users may choose to view different parts of the canvas in distinct ways (at different magnifications, for example). On the other hand, even if all users are available to draw on the canvas simultaneously, a user may not see the position of every other user's cursor.

The input sharing paradigm for concurrent graphics editors has usually been all or one: either any user could draw anywhere on the canvas at any time, or input was serialized through same floor passing mechanism. Serial floor passing mechanisms range from having a designated machine (podium) as the only site from which input is accepted, to passing input control under the control of a moderator, to FCFS satisfaction of requests for the floor, to various token passing mechanisms. Unconstrained access is usually limited to pixel-oriented drawing editors.

Gesture sharing paradigms range from no sharing of cursors, to having a distinctively shaped cursor for each user visible to all users. Some systems allow serial access to a single "conference pointer," eliminating the distraction found when many cursors are visible at once [28]. This problem can also be addressed by altering the view sharing paradigm, as is done in Ensemble.

The WYSIWIS view sharing paradigm is the earliest and most obvious paradigm for view sharing. WYSIWIS, or What You See Is What I See, supports a chalk-board type of interaction where every one sees the same view of the shared material. A major shortcoming of this method is the lack of privacy to the collaborator as strict WYSIWIS outlaws the presence of a private work space [13, 27, 28]. In addition, many problems present themselves in accommodating user control of his view of the shared data [30].

Another abstraction known as WYSIWIMs, or What You See Is What I May See, allows independent viewing perspectives and private work spaces [23, 24]. In Ensemble, just because a user's cursor position is available for another user to see does not mean that the other user must see it unless he requests it. Further, users need not observe the same parts of the canvas, or even view the parts observed in common in the same manner. This paradigm is more general than the relaxed version of WYSIWIS provided in Mermaid [30]; users need not have the same parts of the shared objects on their screens, and may not even see the same version of a shared object. This WYSIWIMs paradigm is similar to the filtering provided in Suite, which propagates changes depending on syntactic or semantic integrity in a structured editor environment [8]. However, in graphics editors, semantic integrity is somewhat harder to specify than in the more constrained world of structured editors.

Interactions occur between the various sharing paradigms and the implementation approach used. Two main approaches to sharing a drawing/gesturing space electronically are distinguished by the way graphic input is obtained by the system. One approach is to use a video camera to capture a physical drawing or object, which is then transmitted to other users, perhaps merging it with other scenes captured by camera or generated by computer (such as cursor positions). TeamWorkStation is one example of this type of system [19]. This approach makes it difficult to save or modify the shared scenes, but allows very natural input, using familiar drawing tools comfortable to users (such as pencils, brushes, etc.) as well as permitting display of actual objects. However, it depends on costly, specialized equipment at every input site, so that input can only occur at special locations, and only a user physically at the site of a televised object will be able to manipulate it.

The other implementation approach for input is to use only canvases produced by graphic editor programs. These come in two flavors: pixel-oriented (or "paint" type programs) and object-oriented editors (not to be confused with object oriented programming languages). Pixel-oriented programs simply change the values associated with pixels of the canvas, and do not associate the value at one pixel with the value at another pixel. For example, once a circle is drawn, it looses its identity as a logical component of the picture: parts of it may be erased, and it may not be moved independently of overlapping components. The canvas is stored as a pixel map. Commune, XSketch, and tools in Augment, Colab and CES are examples of this type of system [6, 22, 10, 28, 26].

Object-oriented graphics editors store the canvas as a collection of parameterized objects, which may be manipulated independently regardless of their relative positions on the canvas: a circle may be resized or moved even though it overlaps a box. Part of a circle may be "erased" only by placing an opaque object over that part of the circle. While in pixel-oriented programs, parts of the canvas are selected for manipulation based on their position (an area is selected), in object-oriented programs, objects are selected for manipulation.

Concurrency control is based on avoidance or resolution of collisions. Within a file, a collision may happen when two or more users edit the same region of the canvas. When a pixel-oriented editor is used, raster operations are performed in the order they arrive and collisions present little problem. In an object-oriented graphics editor, collisions occur only when the same object is selected for manipulation. If modification of graphics objects are allowed, then some form of mutual exclusion must be used to maintain integrity.

After an editing session is completed, the resulting document should be consistent at all sites. During editing, the stable, committed portions of the document should be identical at all sites, within some reasonable update delay. It is incumbent upon the concurrent editor to ensure that all users have access to the same document, at
least in the parts that are not under active modification.

Concurrency control is not only important at the level of internal editing within the file, but also at the level of file access. Collisions at the file access level are acceptable only if the application is collaboration-aware and users are only allowed to access the file through coordinated programs. For many editors there is no access control to the file itself forcing use of the groupware system for editing. While some users could edit the file using the groupware system, another could edit it using a single-user application, producing the standard problems of lost updates or inconsistency. Those that do limit users to accessing the file through groupware generally require the group of users editing the canvas to be formed explicitly. In Ensemble, the group is formed implicitly by the act of opening a file to edit.

In addition to the user interface and internal representations, a shared application must have a distributed architecture. A collaboration-aware system may have a centralized or a replicated architecture, or some hybrid schemes combining these.

In centralized architectures, there is only one instance of the application program running (e.g., the graphics editor engine) [2, 3, 14, 20, 21]. All input is channeled to that single process, which selects inputs and produces output that is broadcast to display processes at all user sites (see Figure 1).

Figure 1. Centralized architecture.

These architectures have the advantage of keeping a consistent state across all views, since the input is serialized and only one copy of the internal structure exists. However, in order for the results of manipulations to be displayed, the input must travel to the central site and back again, potentially allowing some "stickiness" to exist in the system [7]. This is rarely a problem on local area networks [2], but could be significant over slower, longer communication paths.

Replicated architectures have a copy of the application running at each site in addition to an input and a display process (see Figure 2). The input processes broadcast input from each site to all the sites, where each site selects and acts on its application's internal representation. Since input data usually uses far less bandwidth than display information, this reduces load on the network. Further, the response time is excellent since the local display is driven by a local copy of the application reacting to the local input. Another advantage includes the ability to collaborate in spite of some differences in hardware architectures [1]. In principle, if all sites receive the same messages in the same order, the internal representations remain consistent across all copies of the application. However, in practice there can be significant synchronization problems.

Figure 2. Fully replicated architecture.

The hybrid architecture used by DCS and most of its applications, including Ensemble, combines these two approaches (see Figure 3). There is a central coordinating process that maintains the master copy of the internal representation, but each site has input, display and application processes as the replicated architecture. The key difference is that input that could cause inconsistency is first sent to the coordinator for approval, while operations guaranteed not to cause inconsistency are handled and displayed locally. This enables the response time to remain good, while input from the central coordinator is broadcast to all copies of the application; thus it also has

Figure 3. Ensemble’s hybrid architecture.
lower bandwidth requirements than the centralized architecture. In addition, this permits customization of the local copy of the application. Finally, local processes can decide not only when to send input information to the central process, but also to which inputs they will react. This gives a natural mechanism for implementation of the WYSIWIMs view-sharing paradigm.

EXISTING CONCURRENT GRAPHICS EDITORS

Collaboration systems may be either open or closed; that is, they may support arbitrary serial applications, or may implement a particular collaboration aware tool. In open systems collaboration, single user programs are shared by input and output multiplexing in a pseudo-serial fashion [20][21]: collaboration awareness is added external to the application program. In this manner, unaltered, serial programs may be shared at the expense of either requiring serial control of the program or risking semantically incorrect concurrent access. In collaboration aware applications, the application itself manages concurrent access to its objects, permitting a much greater degree of control and finer granularity of access. Typically, open systems collaboration restricts not only the types of input sharing possible, but also view sharing, requiring the WYSIWIS paradigm to be used.

Open systems are a very flexible approach to shared graphics editing, using shared terminal emulators and view-sharing facilities for sharing familiar serial applications without modification (NLS [10], Augment [11], Project Nick [5], Dialogo [20], Rapport [1], Timbuktu [12], MMConf [7], Share [14] and Mermaid [30]). However, since the original program is not collaboration aware, it has no provision for concurrency control - only one user is allowed to edit the canvas at a time. Some floor control mechanism is used to pass control between users to allow access to the editing facility. These systems, while allowing multiple users to observe editing as it occurs, place a substantial collaboration distance between users.

Collaboration-aware graphics applications typically implement a distributed whiteboard, e.g., Commune [6], Colab's Boardnote [27, 28], MBlank [26]. Here it is not necessary to limit editing access since the raster operations may be performed on each pixel as they arrive. Access is generally either serial or totally unconstrained, with real-time broadcast of modifications.

Commune is modeled after a notepad of paper, and is intended to allow geographically separated designers share a drawing surface [6]. Each site has a horizontal monitor and transparent digitizing pad with a stylus that serves as the writing surface. Each user’s stylus appears in a different color on the shared screen. Commune supports free-hand drawing with no notion of lines, boxes, curves, etc. Some problems were found in the interface to the shared canvas due to resolution and sensitivity of the stylus and digitizer used, and to the small size of the shared screen. Also, many pointer movements unrelated to the discussion were distracting, and it was hard to distinguish relevant gestures from random ones.

MBlank allows users within a CES conference to share a bitmap [26]. It runs on a DEC Vax minicomputer, and users view the shared space on Xerox Alto workstations connected by a LAN. The conferencing facility allows workstations to be added and removed from the conference in which a bitmap is shared. All user’s cursors are uniquely shaped and visible on all screens. In addition, each user’s own cursor is displayed twice: once for local tracking, and once for the position seen by other users after it is echoed by the Vax.

The Boardnote application in Colab supports the shared whiteboard paradigm. It has an advantage in providing multiple boards through its "stampsheet" mechanism. However, it is designed only to support users in the same room, and its formats are only useful to the other tools in the Colab environment.

All of these shared whiteboards use the pixel-oriented paradigm, which is appropriate for quick sketching and is amenable to uncontrolled access to multiple users. Object-oriented collaboration aware graphics editors are much rarer, and demand more care in coordination of access by multiple users. Ensemble and Xsketch are such tools.

Xsketch supports a cut-and-paste facility for editing, erasing and moving objects on the canvas [22]. Primitive object types include polyline, box, and text, which constitute a subset of types supported by Ensemble. Drawings are saved in Xpic format, and may be imported by other tools. The architecture consists of client, arbiter, and name server agents in addition to the X11 display server, with TCP connections used for communication. Data is replicated over the clients and is managed by the arbiter, which validates and broadcasts edits. Its architecture is similar to that of Ensemble. A significant shortcoming of Xsketch is that it only supports cut-and-paste modification of objects.

The systems above have all dealt exclusively with electronic objects; other systems support access to physical objects as well. TeamWorkStation supports up to six users by integrating desktops with a shared workspace [19]. The shared workspace is captured by a micro-CCD camera focused on a fixed region of the desk at each workstation. The digital video images are then integrated with computer produced objects such as text and lines. Here is impossible to manipulate the physical objects displayed from other stations, and the collaborative effort put into the session cannot be stored for later use.

In summary, existing systems fall generally into three categories: those using video camera input, open systems, and collaboration-aware graphic editors. Systems using video input have limitations on sharing access to objects and on saving work. Open systems are flexible, but have limited concurrency, especially for input to the shared, serial application. Collaboration-aware graphic editors may be either pixel-oriented "whiteboards" or object-oriented. Shared whiteboards are useful for quick sketching, and may either allow one user to control input at a time or allow input from all users at once. Their editing abilities and resolution may be limited, however. Object-oriented graphics editors are rarer, and are more suitable for production of detailed figures. The only other object-oriented concurrent graphics editor of which the authors are aware has limited modification functionality (cut and paste), and because of this does not need locks. Ensemble differs from these in its implicit locking, WYSIWIMs view-sharing philosophy, implicit formation of editing groups upon concurrent access of an edited file, its architecture, and in belonging to the larger collaboration support environment, DCS.

ENSEMBLE

Ensemble Requirements

Ensemble was built to be used as DCS's graphics editor.
The application interface required by DCS suggested an architecture for Ensemble; it also discouraged separate development of meta-communication channels specifically for Ensemble, since they already exist in DCS. Though the main requirement is that Ensemble be the shared graphics editor of DCS, Ensemble is required to exist as a stand-alone system as well.

Other requirements for Ensemble are based mostly on the issues discussed in the preceding sections. Some products, even though powerful, are under-utilized because of their difficulty of use or unfamiliarity [17]. This suggests that editors with heterogeneous command sets and interfaces be able to interact with each other through Ensemble: collaborators may use interfaces familiar to them. Ensemble is intended to be a modular concurrent editor and a basis for development of other concurrent editors.

In Ensemble, the user should not have to know that there is someone else sharing a file unless they are both trying to modify the same part, or unless close collaboration is desired. For example, the user should be able to see the work of a remote site when needed, but remote cursors and multiple edit updates should not clutter the display. Updates from remote sites should be reflected as soon as they are complete, but not while in progress so that distractions are kept to a minimum. Ensemble should also minimize "collaboration distance" by minimizing editable granularity and maximizing the number of users who could access a canvas concurrently. The next subsection describes the capabilities of Ensemble that meet these requirements.

**Ensemble Design**

Ensemble is invoked through the X Window System. It is based on the tgif graphics editor from UCLA. When it first appears, the user may load a file through the file menu. Each file has a logical file manager associated with it, so that the concurrency control is automatically started when users start to share a file. When used within DCS, this is the only way in which a user may access the graphics file. Also, there can be no inconsistent access at the file level by using incompatible or collaboration-unaware editors.

Ensemble appears as another window on the user's screen. It can be iconified and resized without affecting other users. The user is free to move between the Ensemble window and his private work areas. The file to edit is loaded through the window. Multiple Ensemble windows may be invoked by a user to edit the same or different files. Most of the functionality of the tgif serial graphics editor, on which Ensemble is based, is achieved through menus and mouse click operations.

Concurrency control within a file is handled by locks placed implicitly when a user selects an object. If a selected object is already locked by another user, the selection operation fails and the user is informed. Any number of writers may edit the file concurrently: locks logically partition the canvas into mutually exclusive sets of objects, so there can be as many active editors as there are objects. Locks are relinquished when the objects are deselected. At this time, and not before, other viewers may see the new version.

Users are informed of locks held by other users by clear boxes, similar to the black boxes a user obtains when he selects an object. Currently, users have no option to observe an edit in progress until the other user releases his locks on the objects. The only continuous, real-time updates received are on pointer positions.

Users may gesture with their pointers. In order to prevent the distraction of many pointers moving around the canvas at once, and to reduce the demands on both the network and the graphics window managers, a user may choose not to export his pointer location, or not to import other users' pointers. So, a remote pointer is displayed on a user's canvas only if he imports pointers and another user exports his pointer.

**Ensemble Stand-Alone Architecture**

We have chosen the implementation platforms of Ensemble on the criteria of speed, availability, portability, extensibility, and ease of implementation. Ensemble is written in C, a language that can be exploited for speed in execution. The first version is implemented on the UNIX operating system, which has a wide usage in academic and scientific circles. The user interface is built on the X Window system for similar reasons. Interprocess communication is done using internet sockets. We preferred socket communication over RPC for reasons of speed, wide compatibility, and wide area network (WAN) capabilities.

An important feature of Ensemble's architecture is its process distribution. It uses a hybrid, federated architecture with the processes spread among users' workstations to avoid any overloading (see Figure 4). Ensemble consists of three types of processes. Figure 4 shows process hierarchy and the connection topology of the whole system. Numbers in parentheses in the text below refer to the labels in this figure. The file manager (FM) is first in the hierarchy and handles session control. Each session consists of a graphics manager (GM) and a collection of graphics editor windows (GWS) that make up a partially replicated architecture. The communication and process synchronization of Ensemble depends on broadcasting and multicasting of messages. We have organized the processes in a star topology to minimize discrepancies in message delays, and to provide input synchronization.

A session starts when a user loads a file in Ensemble. The initial GVs contacts the FM to request a file list (I), from which the user chooses the desired file. If there is no GM for the requested file, the FM spawns a new GM (2), which then acts as a coordinator for that file. After the GVM informs the FM that it is functional (3), the GM provides the GVM with the GM's address (4), and the GVM establishes a connection with the new GM (5, 6). As additional users enter the editing session on that file (A), the FM informs them of the existing GM's address (B) and they connect to the GM (C, D) until the session terminates by all users leaving the session. The FM then kills the GM.

The GVM is the user's interface to Ensemble. It generally runs on the local machine and is fully replicated at each user location. The GW carries the heavy processing load of editing the canvas, and each GW keeps an internal copy of the representation of the canvas. It may differ from the copy kept at the GM in two ways: the GM may have just received an update from another GW that has not yet propagated to the copy kept at the GW; or the user may be manipulating some selected objects locally that have not yet been committed and sent to the GM. Also, the GM keeps track of all exported pointer positions, which may not be imported by a particular GW.
amount of information to DCS regarding the application name and usage. Applications in DCS are required to have a (virtual) application manager (AM) for each instantiation of the application. This AM is created by the conference manager (CM) when needed, and users may be directed to an AM based on parameters of the application requested. These applications then have access to DCS’s facilities for registration, communication, and conference control. This distributes the workload within DCS in two ways. First, each process that handles multiplexing within an application can be relatively simple since they handle only a fraction of the communication load. Second, a new AM may be started on a different machine from the CM or existing AMs, so no one machine will become overloaded.

When Ensemble is incorporated in DCS, the conference manager (CM) takes the place of the FM in the stand-alone version. The GW is invoked as a DCS application through the status window. Figure 5 shows the place of Ensemble in the DCS system.

In DCS, a set of mechanisms for conference management supports a wide range of control paradigms. The small, flexible message-passing interface to the conferencing management processes permits shared applications to be installed easily in DCS. Currently, in addition to Ensemble, DCS has applications that support concurrent text editing (MACE); single input control, open systems sharing of “glass-ty” applications (MATE); a discussion window (multi-person talk); a status window (for conference control and user-defined motions); and automatic creation of transcripts of meetings including motions made and voting results.

Shared applications may be made accessible within DCS by adherence to the DCS message-passing interface within the application code, and by providing a minimal

**Figure 4. Ensemble’s stand-alone federated architecture.**

**Figure 5. Ensemble in the DCS Architecture.**
Ensemble Graphics Manager
Each canvas that is edited is associated with an GM. Together they form a logical object. The GM is the server that controls the edit session. Since the GM is the bottleneck of the topology, we have designed it to carry a minimum process load: the GM is not involved in the actual text edit at all. It is responsible for the paging mechanism, granting of locks and message multicasting of messages. By using sockets, the lock requests are serialized and the GM may detect overlapping locks easily. Since all lock requests for a selection are sent in one message, and failed lock requests are simply returned as failures, there is no possibility of deadlock. If a user has held a lock too long in the opinion of other users, Ensemble has the ability to purge all locks on request.

All communication among processes is done through message passing. Since the connections are implemented as connection oriented stream sockets, the order of messages in the channels is preserved. We prevent deadlock from occurring by having only asynchronous message delivery and reception. There is no acknowledgement system nor do processes ever wait for specific replies. At present, Ensemble does not have fault tolerant features and we assume an error free channel.

The internal representation of the canvas is replicated in Ensemble. The GM keeps the primary copy of the object list of the canvas being edited, while each GW holds its version of the canvas, reflecting changes made locally but not yet committed to the GM. Edits are made locally, with changes sent to the GM only when the user saves his edit by deselecting the edited object(s).

Ensemble Graphics Window
What we refer to as Ensemble’s graphics window is really a set of processes, including a local copy of the modified tgif editor and the X11 display process. The GW provides a user interface that is both the editor and drawing surface display and the control panel for communication with the GM or FM. Pulldown menus allow users to import remote pointers and to export their local pointer. Currently, users do not have the ability to selectively import remote pointers. A banner line at the top of the screen allows messages from the GM to be displayed to the user. Figure 6 shows two graphics windows in Ensemble, displaying local and remote locks in both windows, and remote pointers in one window.

The editor supports a superset of the object primitives found in Xsketch, and is able to save its canvas in a superset of the formats used by these other editors. Besides Xpic format, Ensemble can save canvases in postscript format, as was done with the figures in this paper. In addition to the polyline, box, and text primitive types, tgif (and hence, Ensemble) supports ovals, polygons, arcs, and rectangular-circular objects. Objects may not only be moved, but also resized or reshaped. Unlike the whiteboards, it does not support free-hand sketching.

EXPERIENCE WITH ENSEMBLE
We have used Ensemble both within DCS and as a stand-alone tool at the University of Florida for over two years. As a stand-alone tool, we found on occasion that there were collisions at the file level because users applied tgif to a file that was also opened under Ensemble. This was rare, but did happen when other tgif features not supported in Ensemble were desired.

Figure 6. Two graphics windows in Ensemble

When used under DCS, these collisions never happened. The automatic concurrency management was found to be very smooth, and often users were not aware that others were accessing the same canvas. On several occasions, we deliberately chose to produce a canvas concurrently. We were able to do this without much difficulty once the task was partitioned: in one case one collaborator drew a process box diagram while the other wrote out the steps and labeled the boxes and arrows.

It was convenient not to have to either request the floor, as in serialized access facilities, or to have to place explicit locks, as in MAPE and some other lock-based tools. The implicit locking paradigm is very natural to object-oriented editors. It was sometimes a surprise that an object that a user had attempted to select failed; perhaps the clear boxes indicating a remote lock on an object are not obtrusive enough, but this does not seem to be much of a problem.

The ability to import or ignore remote pointers has been found useful. In our experience, the lack of ability to selectively import remote pointers has not been a problem, since we could limit the pointers that were exported at their source. The shape of the remote pointer was unfortunate, in that it closely resembles one of the selection boxes. It is rare that this causes confusion, and may be fixed easily.

When working in close collaboration, we found that we needed to be able to speak to one another. This was no problem when we were colocated, but when we were in different rooms it became a problem. The discussion window, a text interface, was too slow for much beyond the initial setup of the editing effort. When editing it was cumbersome to move from the graphics window to the discussion window and back again, so we are pursuing an audio channel to augment DCS. Currently we use telephone connections when we need audio augmentation.
CONCLUSIONS
Ensemble is a collaboration-aware, concurrent, object-oriented graphics editor that uses implicit object locking for concurrency control. It makes several types of sharing possible, implementing a WYSIWIMS (What You See Is What I May See) paradigm. The sender (editor) and the receiver (viewer) have joint control over the degree of view sharing. Users see updates from other users as they are committed.

The architecture of Ensemble is federated, a hybrid between replicated and centralized architectures. By using an intermediate editing notation, any number of editing user interfaces may be supported. Because of this modular design, Ensemble may be considered an open system for editing, allowing third party components to be added later.

As a stand-alone tool, Ensemble permits consistent updates to a concurrently edited graphics file. As part of the Distributed Conferencing System (DCS), Ensemble inherits facilities for real time communication with collaborators, mechanisms for controlling the disposition of jointly created objects, and other application windows that may also be useful.

Additional editing interfaces that do not require X Window, as well as some supporting a command interface more familiar to various users (such as xfig or pbdraw) would permit greater flexibility and more use of Ensemble. Also, the use of a text-based discussion window for communication has been found awkward, particularly when simultaneous gesturing is desired. The incorporation of an audio discussion facility would greatly enhance the usefulness of Ensemble. This is currently under development within DCS, using the audio facilities provided in Sun4 workstations.

REFERENCES


