High Performance Infrastructure for Visually-intensive CSCW Applications

Stephen Zabele *Steven L. Rohall* *Ralph L. Vinciguerra*
gszabele@tasc.com slrohall@tasc.com rlvinciguerra@tasc.com

TASC
55 Walkers Brook Drive
Reading, MA 01867, USA
Tel: 1-617-942-2000

ABSTRACT

We describe a scalable CSCW infrastructure designed to handle heavy-weight data sets, such as extremely large images and video. Scalability is achieved through exclusive use of reliable and unreliable multicast protocols. The infrastructure uses a replicated architecture rather than a centralized architecture, both to reduce latency and to improve responsiveness. Use of 1) reliable (multicast) transport of absolute, rather than relative, information sets, 2) time stamps, and 3) a last-in-wins policy provide coherency often lacking in replicated architectures. The infrastructure allows users to toggle between WYSIWIS and non-WYSIWIS modes. That, coupled with effective use of multicast groups, allows greatly improved responsiveness and performance for managing heavy-weight data.

KEYWORDS: CSCW Infrastructure, Reliable Multicast, Scalable Architecture

BACKGROUND AND MOTIVATION

Our objective is implementing an infrastructure for computer-supported cooperative work that satisfies many diverse goals. The system must support a large number of users, users can be widely distributed from each other, perhaps across the country, and the system must efficiently handle very large data sets of varying type, such as large images and video. Given these constraints, we have implemented a replicated architecture. This choice merits more attention, as it has a large influence on how we satisfied our design goals.

Centralized vs. Replicated

The debate between centralized versus replicated architectures for multi-user applications is an old one. The two primary issues are *performance* and *consistency*. (Other issues presented in the literature, for example [2,5], largely focus on implementation details and are less compelling.) Replicated architectures have been lauded for good performance: they require less network bandwidth since only input, or state-changing information, must be transmitted between clients. Replicated architectures also provide good feedback to the user since locally-initiated input is handled locally: there is no need to wait for the input to be processed by a central authority and then transmitted out to the clients. In comparison, centralized architectures appear better at maintaining consistency among the clients: the central portion of the system sequences the various inputs from the clients and ensures that every client sees the same changes at the same time, albeit somewhat delayed. Additionally, adding late comers is much easier than in a replicated approach, due to the centrally stored state.

Both the performance and consistency arguments have been greatly influenced by the type of network used in implementing past systems. Systems either used heavy-weight, connection-oriented streams to provide reliability at the cost of bandwidth and performance, or they used light-weight, packet-oriented datagrams at the cost of reliability. Recent developments in network protocols allow us to reinvestigate the issue of centralized versus replicated and hopefully abstract away from the network implementation issues that have clouded past arguments. In particular, reliable, sequenced multicasting of packets can provide the reliability found in centralized systems and the performance found in distributed systems.

Rendezvous™ [1,2] is a good example of a centralized approach to building multi-user systems. Rendezvous relies on a central abstraction connected via bundles of constraints, or links, to multiple views. This is called the abstraction-link-view paradigm, or ALV [3]. In Rendezvous, the abstraction and the views all run as light-weight processes within the same heavy-weight operating system process. Connections across the network are via the X Window System™. X serves as a virtual terminal and is the interface between the user and the system, both for input and output. Assume that there are n users in a conference. If every user provides some sort of non-conflicting input (such as scrolling a window or clicking the mouse), then $O(n^2)$ messages are sent through the network. Any single message requires one transmission to the abstraction and n-1 transmissions from the central abstraction to the other views. For each user of n users to send a message (n messages), this becomes n*(n-1) or O(n^2).

The price in network usage, though, is not without merit. Rendezvous provides a reliably consistent view to each user. In fact, the communication mechanism worked so well that some applications relied on the *reliable, sequenced broadcast* of state changes even for updating the interface of the user who made the change[4]. This proved to be a simple and elegant way to write applications.

The Rendezvous abstractions and views described above actually ran within one process on a single processor. Assume that a distributed constraint system was implemented (as described in [2]) and that views ran on the users' machines and not on the machine running the abstraction. Network traffic is still O(n^2) as described above. However, if this system is then implemented on a network providing reliable, sequenced multicasting, the network usage is vastly improved--O(n). Any single message from a client would be sent over a reliable connection to the central abstraction and is then multicast to every other client, resulting in two network transmissions. For n clients, this becomes 2*n, or O(n). However, the overall message latency is high because the abstraction is still processing every message. If the clients are separated by great distances, such as across the country, round-trip message time becomes quite significant--300 msecs or so even on a fiber OC3 network.

In contrast, MMConf [5] is a good example of the replicated approach to multi-user applications. Although its performance is good (O(n) network messaging traffic in theory but no centralized bottleneck to add latency), in practice, applications built on top of MMConf quite often lost synchronization. In addition, applications were arbitrarily limited in their functionality. For example, MMConf explicitly used rigid floor control and token passing to avoid some of the synchronization problems. This meant that some users would have to wait to interact with the application or would not be allowed to interact with it at all. Besides user dissatisfaction, this floor control policy was a complicated piece of code that relied on unique tokens and sequence numbers to work properly--it often did not. As another example, certain user-oriented features such as continuous scrolling were disabled, again to alleviate some synchronization problems. As a result, application programs presented unnatural interfaces to users or were less-powerful than their single-user counterparts. Much of this is due to the fact that MMConf was not implemented with true, reliable multicast--instead it was implemented as best as possible on top of TCP/IP.

The modified Rendezvous with distributed ALV described above and the MMConf system are more alike than an initial appraisal reveals. While the two systems vary greatly in programming style and implementation detail, the high-level architectures are remarkably similar. In essence, by splitting the Rendezvous abstraction and distributing it among the now-distributed views, a replicated architecture has been produced. Assuming that Rendezvous' reliable, sequenced message delivery is maintained, the end appearance to the user should stay the same, with O(n) messaging and much less latency for user feedback. Likewise if MMConf were enhanced with truly reliable, sequenced multicasting, the appearance of applications would look very similar to those implemented with Rendezvous.

The Main Issue

The fundamental realization is that it is more important that every client see a *consistent* set of events rather than trying to maintain any notion of a *correct* set of events. Reliable, sequenced multicasting can provide that function, even in a replicated environment. Higher-level, human-to-human protocols can then evolve to resolve any seemingly bizarre behavior that remains, in much the same way that policies naturally evolve in phone conferences. This realization compelled us to investigate mechanisms for reliable multicasting and to use those mechanisms in building our multi-user application infrastructure.

DESIGN GOALS

The desire to evolve an infrastructure that is scalable and flexible, that offers high-performance, and that well supports diverse user interactions and data types strongly influenced our design activities.

Scalability

A primary goal of our design effort has been distilling a CSCW architecture that supports a large number of users and that supports efficient exchange of large data volumes, such as image and video data sets. CSCW applications are traditionally implemented using unicast protocols which function well for two or three users; however, performance degrades quickly as the required bandwidth can increase (as illustrated above) as the square of the number of users. As a consequence, collaborative applications built upon unicast protocols are quickly mired with even moderate numbers of users, particularly for imaging applications. We have therefore emphasized use of multicast protocols whenever possible and appropriate, as the required bandwidth will increase at most linearly with the number of users. To achieve this end, the CSCW communications model centers on the reliable multicast protocol developed by TASC for the ARPA-sponsored Image Networking (ImNet) project [6,7].

Flexibility

A second goal has been designing a CSCW architecture that supports the breadth of human interactions typically encountered at meetings and conferences. Traditional CSCW architectures constructed using shared windowing systems, such as Hewlett Packard's SharedX [8] and Farallon's Timbuktu [9], use a What-You-See-Is-What-I-See (WYSIWIS) [10] interaction model where all users see the same data and all manipulations have global effect; however, typical human interactions do not fit this model. For example,

- Within a single conference there can be multiple sessions that an individual moves between;

- Within a session there are often short, spontaneous, side discussions between members from the same organization in addition to any central discussions being held by the group as a whole;

- An individual will often privately examine material within conference proceedings or briefings other than what is currently being presented in order to clarify a previous point or to preview upcoming material; and

- An individual will occasionally contact a colleague not involved with the conference to discuss the consequences of any newly presented information or ask for additional information before making a presentation.

Our objective is to support a full range of interaction modes, including the ability to switch between public and private control of views and the ability to set up subconferences where a subset of the conference participants can share and manipulate information without affecting the group as a whole.

Eliminating mandatory global synchronization of all views has the added advantage that users can independently control the presentation and layout of screen area, thereby allowing different users to have different views open or to have views arranged differently on the display. As applications become more complex and competition for screen area increases, independent control of screen layout by individual users is extremely desirable.

Performance

A third goal in formulating the CSCW architecture has been optimizing performance for image conferencing in a heterogeneous network environment. Consequently the architecture leverages the adaptive compression and hierarchical image transport mechanisms also developed under the ImNet project that allow user-controlled trades of image quality for speed of response.

The design goal of supporting a wide range of interaction modes coupled with the design goal of supporting a large number of users offers an additional means for optimizing image conferencing performance. By eliminating the mandatory global synchronization of all views and allowing users to select and manage views independently, the particular data sources that each user is actively engaging are explicitly identified. Combining this information with the explicit group setup properties of multicast, the transport of individual sets of image data can be restricted to only those users currently needing the data. The combined approach offers substantial gains in conferencing performance, even over systems such as Rendezvous that support multiple, simultaneous user interactions but remain tied to TCP/IP protocols.

Diversity

A final goal has been designing an architecture that readily supports a full range of data types, including images, graphics, text, audio, and video, as well as an architecture that is extensible so that new data types, interaction modes, or manipulation tools can be integrated as quickly and as easily as conceived or become available.

DESIGN ISSUES

The central issue in the design of our CSCW infrastructure was maintaining coherency between distributed object states while minimizing delays in responding to user actions.

Maintaining System State

Canonically, CSCW operation involves translating actions by each user, such as key clicks or mouse movements, into changes in the state of the CSCW environment. The implicit problem in designing a CSCW architecture is maintaining a coherent definition of the state of the environment across all user workstations involved in the session.

As even subtle differences in the state of the CSCW environments on different workstations can have disastrous effects, state changes made by one user must in general be propagated to all other users using reliable protocols. Some state changes can be communicated with unreliable protocols. For example, fleeting states changes that only affect the presentation of a single user's environment, such as intermediate pointer positions, are unimportant, and use of unreliable protocols reduces latency. However, any state changes that are non-transient, such as ending pointer position, or any state changes that affect the CSCW environment as a whole, must be communicated using reliable protocols.

Use of reliable protocols does not, however, guarantee coherency between environments. CSCW by definition involves multiple users and as such there is a strong potential for different users to modify the state of the environment in contradictory ways. For example, suppose two users each load a different image into a shared viewer at very nearly the same time. Since there are unavoidable communication delays, each workstation will receive a remote request to load an image from another workstation immediately after having serviced a local request to load an image. Without a mechanism or protocol for maintaining coherency between environments, each workstation can easily arrive at a state with completely different images in a supposedly common viewer.

A common approach for resolving coherency problems involves the use of a round-robin or token passing control scheme, as is employed in several commercial products such as HP's SharedX and NeXT's Greyboard implementations. With a token passing approach, only the user currently in possession of the token can make changes to the state of the CSCW environment. As there is only a single (but moveable) point of control, absolute sequencing of user actions is assured, such that a token passing approach provides a straightforward mechanism for enforcing coherency. However, the approach induces unnecessary latencies and suffers potentially severe performance problems for even moderate numbers of users. For example, consider the typical 150 millisecond (or greater) round trip time necessary to pass a token reliably from an east coast user to a west coast user. Ignoring any token dwell time at individual workstations, the time between requesting the token and acquiring control of CSCW environment could easily exceed several seconds for a fifteen user conference.

As delays on the order of seconds are unacceptable from a user perspective, we have adopted an asynchronous, multi-threaded control model that provides multiple users with simultaneous control over the state of the CSCW environment. Under this model, any absolute ordering of user actions is necessarily precluded and requires use of a conflict resolution protocol to maintain coherency between workstations.

Within the architecture, entities such as viewing tools (or views) and images are referred to as objects. Whenever an object is manipulated or changed, the change is transmitted along with a timestamp indicating the time of the change. If a workstation receives a message to modify an object in a way that conflicts with a more recent change, the message is simply ignored. Although the approach is more complex, it offers greatly increased responsiveness as well as considerably more flexibility.

Absolute synchronization of clocks between workstations is *not* necessary for the described approach to work, as CSCW synchronization only requires that the distributed copies of the CSCW state be the same rather than fair or correct in any way. In the worst case where a workstation clock is greatly ahead of all other clocks, no changes made by the user at that workstation can be undone by any other conference participant. This occurs as time stamps of attempted changes by other users will always predate changes made by the workstation with the errant clock and will be discarded. Although this is not fair to the other users, synchronization is maintained.

Obviously loose calibration of system clocks, say to within 100 milliseconds, is desirable operationally and is achieved by a periodic multicasting of a reference clock time. When each user's workstation receives a reference time message, the reference time is compared with the local clock time to derive a relative time shift value. Time stamps are then constructed by adding the time shift value back to the local clock time.

Communicating State Changes

There are (at least) two design alternatives for communicating changes made to the state of the CSCW environment. Either modified objects can be transmitted in totality, or only the specific changes made to the objects can be transmitted. While fundamentally simpler, an approach involving transmission of complete objects needlessly compounds the problem of maintaining environment coherency. For example, consider the independent actions of one user scrolling an image in a view, and a second user changing image contrast within the same view. If the entire state of the view object is transmitted by each user at nearly the same time, the end state of the respective CSCW environments will not reflect changes made by one of the users. Our design breaks the state information within an object into the smallest units that can be modified independently, offering still greater flexibility and responsiveness.

Information contained within an object can be divided into state and data. *State* refers to attributes such as zoom settings or scroll positions, whereas *data* refers to the usable information such as pixels in an image. As maintaining coherent environment state is critical, all changes in state information must be transmitted to all users within the conference; however, state information is intrinsically light weight in that relatively small amounts of data must be transmitted. As such, communications of state information is inexpensive, particularly if multicast protocols are used.

In contrast, data can be either light weight or heavy weight. Heavy weight data sets, such as images or video (and potentially audio for low bandwidth links) require significant amounts of bandwidth, whereas light weight data sets, such as text or annotations, require small amounts of bandwidth. For example, a circle is completely specified by origin and radius, requiring transmission of only a few bytes of information, whereas a 1K by 1K image can require transmitting 100 kilobytes or more, even in compressed format.

As one of the main communication challenges for image-based CSCW is minimizing network traffic for functions involving access of heavy weight data volumes, we explicitly differentiate between transfers involving light weight data from transfers involving heavy weight data. In our CSCW architecture, light weight data is always propagated to all users, whereas heavy weight data is only propagated to conference participants actively using the larger data sets. With this scheme the control and state information critical for maintaining coherency between CSCW environments is fully and immediately available, yet the network is not needlessly choked by unnecessary data propagation involving large data transfers.

DESIGN LAYOUT

The design centers on the use of a shared area, referred to as a bulletin board, where shared objects, representing users, tools, and data, are placed and manipulated. Interactions between objects are accomplished through the messaging paradigm common to object-oriented languages, where messaging functions have been extended to provide transparent network communications between distributed copies of the objects. Coherency between distributed copies is maintained through a synchronization facility intrinsic to the messaging service.

Bulletin Boards

The communications infrastructure of the CSCW design resembles the distributed blackboard architectures commonly encountered with distributed processing models for expert systems. Because of the intrinsic human focus on the data interchange, and due to the topical similarities of the CSCW system with various chat capabilities on electronic bulletin boards, we refer to our design as a distributed bulletin board, or simply a bulletin board.

Within a CSCW session, each client workstation maintains a bulletin board locally that is a copy of the master bulletin board maintained by a *Conference Manager*. Bulletin boards support management of objects on individual workstations and receive messages concerning the creation or change of objects from other workstations. A master bulletin board differs from client bulletin boards in that the master bulletin boards have additional functions for exporting state information. The export functions allow client bulletin boards created by late-comers to synchronize with the conference.

Bulletin boards contain three basic object types: user objects, tool objects, and data objects, as illustrated in Figure 1. A user object represents a user participating in the conference. A data object represents data sets such as images, video sequences, audio sequences, graphics sets, or text. A tool object represents a public or shared view of a data object and a set of methods for accessing and manipulating data objects.

Objects

Within the architecture, objects can be either atomic objects or collection objects. An atomic object represents a single user, a single tool, or a single data set such as an image or a graphical annotation. Collection objects are used to group atomic objects into more manageable sets for convenience. Collection objects can also contain other collection objects, allowing expression of hierarchical relationships. For example, a collection data object can consist of an image data object, a text data object, and a collection containing several graphical annotation data objects. A collection user object can contain multiple user objects, each of which can be an atomic user object or another collection user object.

User, data, and tool objects all derive from a common parent referred to as a Bulletin Board Object or BBObject. Each BBObject maintains descriptions of associations with other BBObjects. For example, a user object maintains descriptions of all public views that the corresponding user is currently accessing. A data object maintains references to all public tools currently accessing it.

The BBObject hierarchy, shown in Figure 2, supports a wide variety of data objects and allows the addition of new data types as needed. The first level of the data hierarchy supports the primary data types, including audio, image, video, graphics, and text. The set of graphics objects is subclassed to support graphic annotations of images. The audio class is specialized to distinguish between conversation, which can be transmitted unreliably, and voice annotations, which are archived and so must be transmitted reliably.

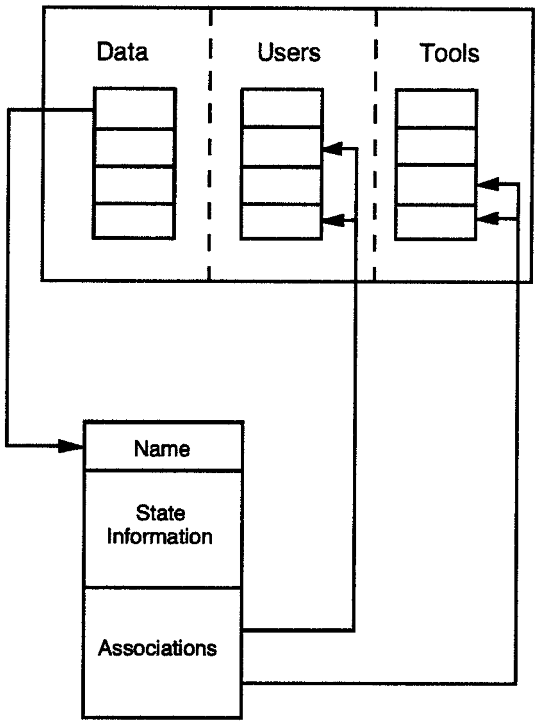

Figure 1. Bulletin Board Structure

Conference Startup

A scheduling process is used to initiate a conference, shown in Figure 3. First, a user calls up the *Registration Interface,* which notifies the *Conference Registrar* of the time the conference is to begin along with an optional set of initial users. The information is placed in the *Conference Schedule*. The Registration Interface is also used to join scheduled conferences, either before conferences start or while conferences are in progress, with the Conference Schedule updated as required. If desired, the Registration Interface and Conference Registrar can enforce various security policies to restrict conference membership; however security issues are not currently addressed.

The Conference Registrar monitors the Conference Schedule, and is responsible for instantiating the Conference Manager for each conference at the designated time. The Conference Manager is responsible for maintaining a master copy of the Bulletin Board that supplies state information for late joining participants, and for conference suspend/resume functions.

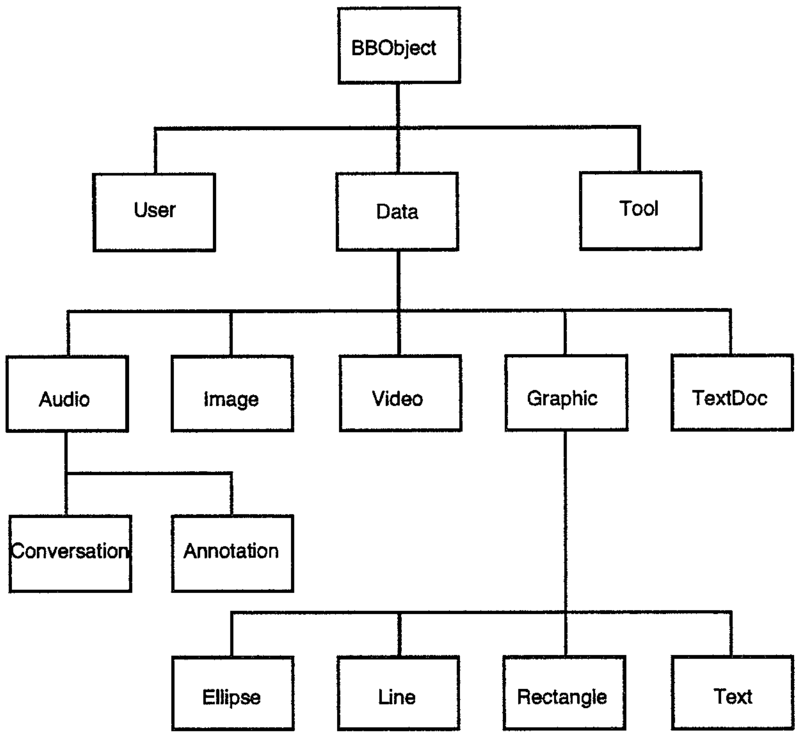

Figure 2. BBObject Class Hierarchy

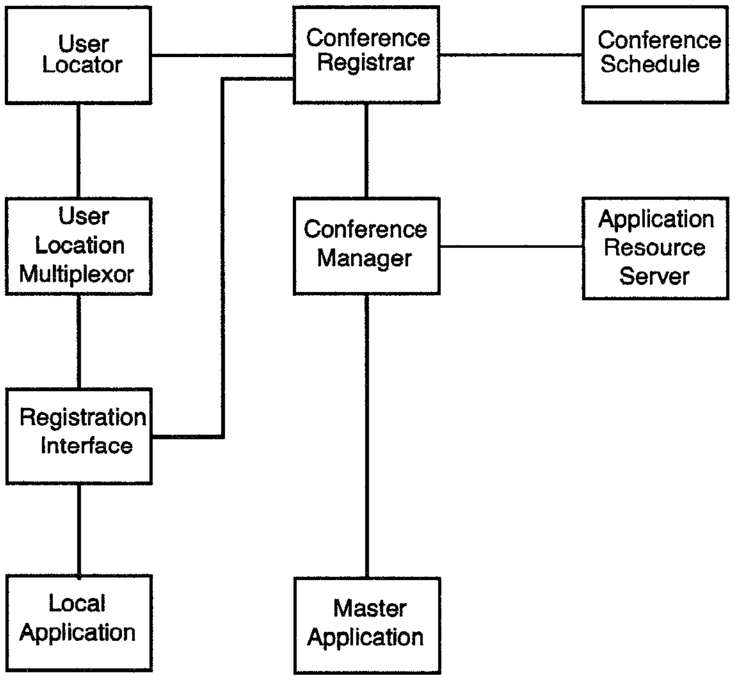

Figure 3. Conference Scheduling

Upon instantiation, the Conference Manager requests conference resources (here, a conference multicast addresses) from the network *Application Resource Server*. The multicast address is passed back to the Conference Registrar for distribution to other participants joining the conference. A Master Application (here, a Master Bulletin Board) is then started, and the Conference Registrar is notified that the conference is ready to begin.

Upon acknowledgment that the Conference Manager has completed its tasks, the Registrar hands off the list of conference participants and acquired resource pointers to the *User Locator*. The User Locator inspects the schedule and contacts the appropriate *User Location Multiplexor* on the appropriate workstation for each registered conference participant. The User Location Multiplexor then launches the appropriate conference application (here, a Bulletin Board) for each user and hands off the resource pointers. The Bulletin Boards are now free to exchange state information as needed to achieve synchronization over the allocated multicast address.

Messaging

Once a conference is in session, a message passing scheme is used to maintain coherency between the individual copies of the bulletin board. All messages transmitted within a conference use a common header format, as shown in Figure 4. The magic number field of the header allows validating the incoming message as a legitimate conference message and contains a value that allows receivers to determine whether the message originated from a little-endian machine or a big-endian machine, so that the remainder of the message can be properly interpreted. The protocol version field is used to identify CSCW implementation versions that are incompatible. The message length is the length of the message after the header, and the sequence number is used for identification of messages sent using a reliable protocol. The message type field is used to distinguish between receivers if multiple message types are sent to a single multicast group address; currently, only messages between Bulletin Boards have been implemented. Finally, the timestamp field is used for ordering messages and for maintaining environment coherency.

In addition to the CSCW message header, a second-level header is used for Bulletin Board messages, as shown in Figure 5. Bulletin Board messages share the timestamp from the CSCW message header, which reduces the message overhead. In addition, a unique object identifier is used that uniquely identifies each object within a conference. Unique object ID's are formed by concatenating the unique ID assigned to the local bulletin board creating the object with an incrementing object ID maintained by that bulletin board. This prevents individual bulletin boards from creating duplicate IDs, yet does not require a central ID service or any explicit communications between bulletin boards to determine the next available ID. The approach also has the advantage that the creator of an object can always be derived from the ID and is useful for archiving purposes.

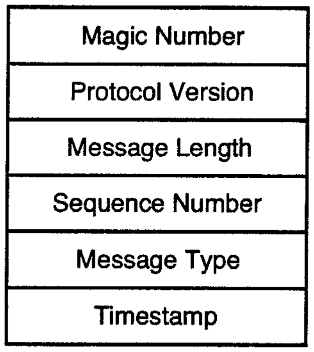

Figure 4. CSCW Message Header

A remote member-function (i.e., procedure) call protocol between objects has been developed to maintain consistency and coherency among bulletin boards. Each object class contains a static array of pointers to remotely callable member functions, the contents of which are class-specific. These member functions process user interactions, such as scrolling a view, accessing a new image, or deleting an object in an object-specific manner. They then propagate the changes as appropriate to mirror objects on remote bulletin boards.

Whenever a user interacts with the conference, the workstation on which the interaction occurs builds a message that identifies the unique object ID, the index of the member function responsible for processing the action, and the parameters needed by the member function, if any. The object ID and the member function index are stored in the BB Identifier field and the Function Index field of the message header, respectively. Any associated parameters are contained in the message-specific data, which is actually the message body. The message is then sent to the BB Multicast Address and received by the conference participants. Upon receipt, the function index is used as an index into the function pointer array, with the parameters extracted and then passed to the function.

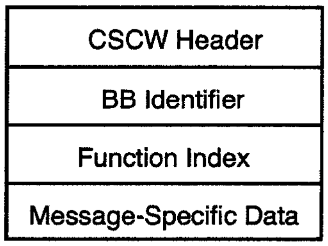

Figure 5. Bulletin Board Message Header

An Example

The use of these messages is best explained using an example. When the conference is started, the Bulletin Board is empty. When the first user joins the conference, a *Create User* message is sent to the BB Multicast Address, say for User R. Each BB Client receives the message and adds User R to the user section of the their local Bulletin Board, while the BB Server adds the user to the master Bulletin Board. Similarly, as User V and User Z join the conference, corresponding user objects are placed on each of the Bulletin Boards.

User V now opens an image viewing tool, and a message is sent to the BB Multicast Address creating a *Tool* object on each of the Bulletin Boards. When user V opens the tool object by clicking on the viewing tool object icon, an association between the user object and the viewing tool object is created, and the corresponding state change is propagated to each Bulletin Board. When User R and User Z see the viewing tool icon appear, each opens the viewing tool, again by clicking on the icon, and the corresponding associations are created. The resulting state changes are propagated to each Bulletin Board.

User V now loads an image into the viewing tool. The sequence of events involved in servicing the request are shown in Figure 6. Initially, the request is received by the viewing tool object. A unique multicast group is used for each image so as to minimize the propagation of heavy weight image data. As this is a request to view a new image, the viewing tool requests a multicast address from the Conference Manager. If the Conference Manager has any unused addresses, it immediately assigns an address to the Tool; otherwise, it first requests a new block of addresses from the Address Server.

The Tool then multicasts an *Intent to Request Image* message to the BB Multicast Address, which contains the multicast group address that will be used for image transfer. The message is received by the Bulletin Boards and is passed to the Tool object with the BB Identifier contained in the message header on the remote workstations. The Tool objects read the message and join the image multicast group. Next, the Tool on User V's workstation issues a request to the appropriate *Image Server* to multicast the relevant portion of the image to the image multicast group. The image data is received by all of the Tools and displayed on the workstations.

Synchronization

Within a network-based, asynchronous, multi-threaded control environment, messages from different users can be received in an unnatural order and require special handling. For example, a message sent by a user manipulating an object can appear before a message sent by a different user creating that object. This occurs if the creation message is for some reason delayed or dropped, and the manipulation message appears intact before the local host completes detection and retransmission processing for the missing creation message. Similarly, messages for manipulating objects can be delayed and arrive after an object has been deleted.

Solutions for processing out-of-sequence messages in general require knowledge about object types and are more appropriately handled by the individual objects. The CSCW environment does, however, provide basic support to assure delivery of messages to objects. Specifically, processing of messages addressed to non-existent objects is handled through the use of a *deleted object list* and a *delayed message queue* maintained locally by each bulletin board.

The deleted object list and the delayed message queue provide the needed services in the following way. When a message (other than a creation message) is received for a non-existent object, the bulletin board checks to see if the specified object is in the deleted object list. If the specified object is a deleted object, then the message is dropped; otherwise the message is placed in the delayed message queue. Whenever an object creation message is received, any pending messages for that object are retrieved from the delayed message queue and delivered (in order of time stamp) immediately.

IMPLEMENTATION CONSIDERATIONS

We have emphasized the use of standards to the greatest extent possible in developing our CSCW prototype. The communications infrastructure is based on IP/Multicast because of its widespread support and use in the Internet community. The complexity of the communications and user interface components required for a CSCW application mandates an object-oriented implementation; consequently we have chosen C++ for its efficiency. Finally, our CSCW prototype is implemented on SGI Indigo™ workstations under UNIX™.

SUMMARY

The CSCW implementation achieves scalability using a mixture of reliable and unreliable multicast protocols to eliminate redundant transmission of data whenever possible. As such, bandwidth requirements do not increase as the square of the number of users, providing a substantially more scalable approach than can be achieved using unicast protocols. Extensibility is incorporated into the design by distilling communication and database functions into a consistent, flexible, well-defined set of primitives and is achieved using an objective programming approach by deriving disparate tools from those primitives.

ACKNOWLEDGMENTS

This work was sponsored in part by the Advanced Research Projects Agency, contract number F19628-91-C-0086.

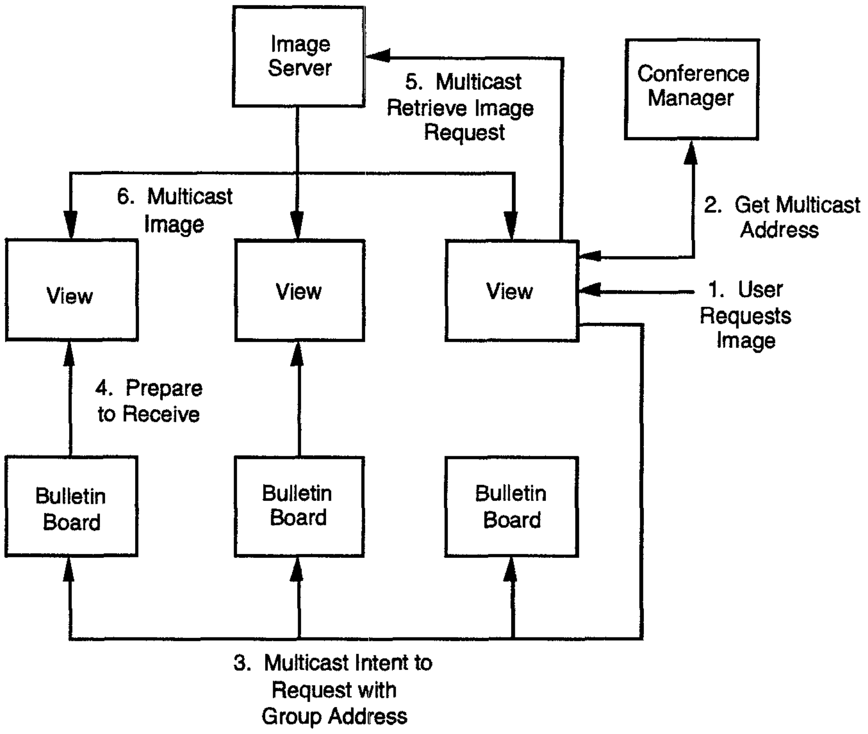

Figure 6. Image Request Processing

REFERENCES

1. Patterson, J.F., R.D. Hill, S.L. Rohall, and W.S. Meeks, *Rendezvous: An Architecture for Synchronous Multiuser Applications*, Proc. CSCW '90 (Oct. 7-10, Los Angeles, CA), pp. 317-328.
2. Hill, R.D., T. Brinck, J.F. Patterson, S.L. Rohall, and W. Wilner, *The Rendezvous Language Architecture*, Comm. of the ACM, Jan. 1993, pp. 62-67.
3. Hill, R.D., *The Abstraction-link View Paradigm, Using Constraints to Connect User Interfaces to Applications*, Proc. CHI '92 (May 3-7, Monterey, CA) pp. 335-342.
4. Brink, T., and L. Gomez, *A Collaborative Medium for the Support of Conversational Props*, Proc. CSCW '92 (Oct 31-Nov 4, Toronto, Ontario), pp. 171-178.
5. Crowley, T., P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson, *MMConf: An Infrastructure For Building Multimedia Applications*, Proc. CSCW '90 (Oct. 7-10, Los Angeles, CA), pp. 329-342.
6. Braudes, R.E, and G.S. Zabele, *Requirements for Multicast Protocols*, IETF RFC 1458, TASC, May 1993.
7. Braudes, R.E., and G.S. Zabele, *ImNet: Very High Speed Image Communication Progress Report: June 1, 1992 - November 30, 1992*, TASC TR-6341-3, TASC, Reading, MA, May, 1992.
8. Garfinkel, D., P. Gust, M. Lemon, and S. Lowder, *The SharedX Multi-user Interface User's Guide, Version 2.0*, HP Research Report No. STL-TM-89-9, Hewlett Packard, Palo Alto, CA, 1989.
9. "Farallon Timbuktu User's Guide", Farallon Computing Inc., Berkeley, CA 1988.
10. Stefik, M., D.G. Bobrow, G. Foster, S. Lanning, and D. Tatar, *WYSIWIS Revisited: Early Experiences With Multiuser Interfaces*, ACM Trans. on Office Information Systems 5, 2(1987) pp. 147-167.