Data Management Issues and Tradeoffs in CSCW Systems

Atul Prakash, Hyong Sop Shim, and Jang Ho Lee
Software Systems Research Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109
Email: {aprakash—hyongsop—jangho}@eecs.umich.edu

Abstract— **Substantial interest has developed in recent years in building computer systems that support cooperative work among groups without the need for physical proximity. This paper examines some of the difficult data management issues in designing systems for computer-supported cooperative work (CSCW). Specifically, we consider an example CSCW system to support large-scale team science over the Internet, Collaboratory Builder's Environment; we discuss the issues of managing shared data in such systems, reducing information overload, and providing group awareness and access control. We discuss several promising approaches to these issues. We point out where a significant gap remains in addressing the requirements of such systems and where designers have to make design tradeoffs that can be difficult to evaluate. Finally, we discuss several open issues for future work.**

I. Introduction

Recent advancements in computer and communication technologies have revolutionized how computers are used. One of the application domains that is gaining increasing usage and visibility is *computer-supported cooperative work* or *CSCW*. In essence, the goal of CSCW is to provide systems that allow users to effectively collaborate on common goals using networked computer systems. The motivation is to enable users to work together across geographical and time boundaries. CSCW provides computer-based work environment in which collaborators can exchange messages, share data, and collaborate, even when they cannot physically get together at the same place and/or at the same time. As a result, CSCW dramatically increases opportunities for group work and provides ready access to resources needed for such work, e.g., shared data and diverse expertise of participants.

The computer systems designed to support CSCW are commonly known as *groupware* or *collaborative* systems. The examples of groupware systems include electronic bulletin boards, shared whiteboards, and chat rooms found on the Internet. The popularity of the CSCW technology is also evidenced by an increasing number of commercial products, such as Lotus Notes, Microsoft NetMeeting, and CoolTalk in Netscape Communicator. Many of these systems have been successfully used, usually in small groups, to facilitate data sharing among distributed participants.

In this paper, we consider some of the issues in the design of groupware systems for emerging applications such as team science, crisis management, and tele-medicine. In such applications, potentially several groups of geographically dispersed people with different areas of expertise jointly work on multiple shared data artifacts. We will specifically examine several critical data management issues that arise in the design of groupware systems to support large-scale collaborations over a wide-area network. In particular, we focus on the following issues:

- **CSCW System Architecture:** The architecture of a groupware system has a large impact on the size and replication of shared state. In addition, it affects the system's runtime performance, the set of services that can be provided, and scalability.
- **Group Awareness:** A CSCW system should provide mechanisms so that users are generally aware of what other participants are doing. Such facilities allow better coordination by enabling users to avoid duplication of work and reduce conflicts. An issue is what constitutes awareness and how best to provide it to users.
- **Reducing Cognitive Overload:** As CSCW systems are scaled up to share a large amount of data or to handle a large number of users, reducing cognitive overload on users becomes important. A large amount of data can be potentially available to group members, and the shared data can frequently change due to updates by group members or external sources. Being in a collaboration does not necessarily mean that users want to be notified of all updates or provided with all the shared data. For example, the subscription to a particular news group does not mean that a user is interested in all the posted articles. Reducing cognitive load on users by organizing coordination activity better, by filtering unnecessary information, and by delivering key information can greatly increase the effectiveness of CSCW systems.
- **Access Control:** Proper access control to shared data increases security and promotes active participation. This is especially critical for supporting CSCW in an open environment, such as the World-Wide Web.

The above list is not intended to be exhaustive. However, it serves to illustrate some of the challenges in designing CSCW systems. The decision to discuss these issues is partly influenced by our experiences with an ongoing project to support team science, called Collaboratory Builder's Environment or CBE [1]. The above issues are emerging as some of the key problems in scaling up CBE to support a large group of scientists.

In this paper, we discuss the CSCW requirements for

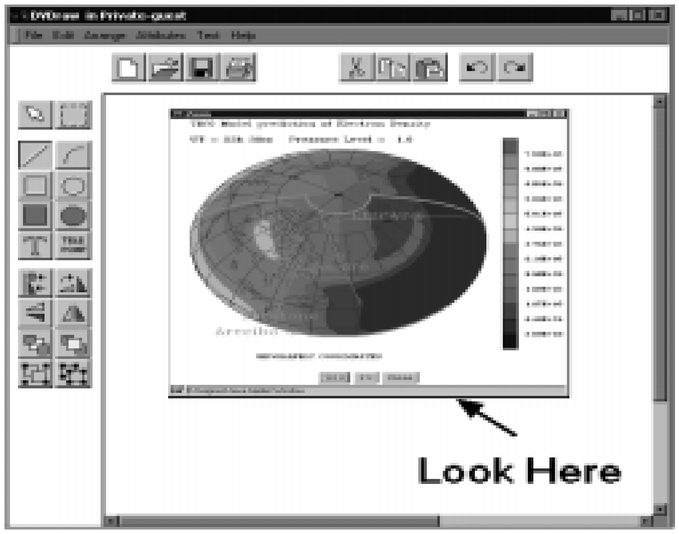

Fig. 1. Shared Whiteboard. Users can freely express their ideas by making free-hand sketches, drawing diagrams, entering text, etc. as well as observe updates made by others in real time.

addressing these issues, present novel approaches found in existing systems, and evaluate the presented approaches in the context of CBE. Although the evaluation focuses on the applicability of the presented approaches in the CBE, we expect the evaluation to be broadly applicable to many other CSCW systems.

The rest of the paper is organized as follows. Section II provides some background information needed for the discussions presented in the paper. Section III discusses the key design considerations in CBE to provide an illustrative example of issues in building a groupware system. Section IV discusses the architectural issues of CSCW systems. Section V presents some of the approaches to dealing with reducing cognitive overload. Section VI addresses access control in CSCW systems. Section VII discusses several approaches to providing group awareness. Finally, Section VIII presents some concluding remarks and directions for future work.

II. CSCW Basics

To illustrate some of the data management issues in the context of a simple CSCW application, consider a shared, distributed whiteboard (Figure 1). A shared whiteboard allows geographically distributed users to draw or write on it and see each other's work. Ideally, it can be used for interactive discussions and brainstorming sessions, just like a physical whiteboard, except that it allows participants to be geographically distributed.

In order to ensure that the users see the same contents, the whiteboard should maintain its state consistent in the presence of concurrent accesses. Many approaches to concurrency control problems exist in the distributed systems literature. However, a direct application of an existing solution, say, a distributed transactions approach, may not work, not because the solution cannot synchronize the state of the board, but because it would not satisfy the demands for high interactivity on the board. If the users have to wait for their previous actions to commit before they can proceed, the interactions on the whiteboard could lose the sense of fluidity, making users' work less effective or the system unusable in practice. Thus, concurrency control solutions have to be devised that ensure interactive response times and yet provide consistency. This may involve designers making tradeoffs between interactive response time, latency in propagating updates, consistency of display with shared data, and the kind of updates users are allowed to make. Or it may require designers to build flexibility into the system and allow end-users to make the tradeoff, depending on their task [2].

Another issue is group awareness. For users collaborating on shared work, an awareness of what each user in the group is doing can play a critical role in coordinating their activities. A traditional distributed system usually takes a passive approach in providing the awareness information. Unix commands such as `who`, `finger`, and `zlocate` usually require users' explicit actions, i.e., type in a command at the prompt. Collaborative systems, in contrast, need to take an active approach in providing the awareness information. For instance, in the shared whiteboard, support may be required to show the users who are active (with the list automatically updated as users join or leave the session), to provide *telepointers* so that other users know where a particular user is pointing, or to provide new user-interface metaphors, such as multiple scrollbars, in order to show where each user is working in a large scrollable whiteboard. Audio/video conferencing may be used to provide additional awareness, but that requires good infrastructure to provide reasonable quality. Providing group awareness has tradeoffs with scalability and performance of the system, and with privacy concerns of participants. In addition, some users may find group awareness information distracting (e.g., seeing several telepointers moving on the whiteboard), so careful design is required so that reasonable tradeoffs can be made.

A shared whiteboard is a relatively simple groupware application. The issues become more challenging when an attempt is made to scale up CSCW systems for more involved tasks and to a larger number of users. Figure 2 shows the interface presented by the UARC system to support team science activities over the Internet. A CSCW session in UARC usually consists of multiple collaboration tools, such as chat, whiteboard, shared data windows, and shared URLs displayed via a browser. Collaboration sessions can last several days and involve tens to hundreds of scientists and students. The system architecture has to be flexible to support evolving needs of scientists and yet be robust so that it can be successfully used over the Internet for both synchronous and asynchronous collaboration. It also becomes more challenging to handle issues of providing collaboration context and group awareness, and reducing cognitive overload when multiple activities are taking place. In the following sections, we examine various approaches to addressing these issues.

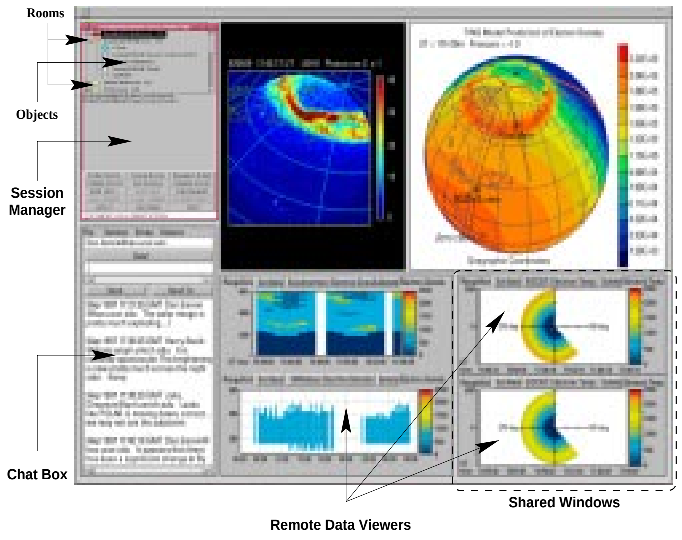

Fig. 2. UARC Collaborative Applications. Session Manager allows users to organize their collaborative and private work. The chat box allows users to exchange textual messages. A whiteboard (not shown) allows users to annotate images and brainstorm about their work. The remote data viewers receive atmospheric data from various instruments at remote locations and graphically display the data according to data types and user settings. Users may export CBE-compatible windows to each other, allowing synchronized views of data.

III. Collaboratory Builder's Environment

The Collaboratory Builder's Environment, or CBE [1], provides a computer-based shared workspace that facilitates team science over the Internet and World Wide Web. It is being used to support an NSF-sponsored project, called the Upper Atmospheric Research Collaboratory or UARC [3]. The UARC project focuses on the creation of an experimental testbed for wide-area scientific collaboration work. This testbed is implemented as an object-oriented distributed system on the Internet and provides a collaboration environment in which a geographically dispersed community of space scientists perform joint scientific experiments with real-time atmospheric data from various sites without having to leave their home institutions. This community of space scientists has extensively used the UARC system over the last three years.

Based on our experience with CBE, we have identified the following usage and data management requirements. While not exhaustive, the identified requirements are general and should be taken into account when building large-scale groupware systems.

• **Users with both individual goals and team goals:** Users often have individual goals as well as team goals. For instance, in the use of the CBE for UARC, scientists often have different areas of expertise. They are interested in advancing their own research, but at the same time, they want to collaborate with other researchers in order to develop models of Upper Atmosphere events.

• **Participation of users in multiple activities:** Users often engage in a number of related activities simultaneously. For example, a scientist in UARC may monitor different instruments, engage in discussions with several groups, and do group work as well as private work.

• **Synchronous and asynchronous collaboration:** Users may require multiple sessions to accomplish their goals and they may not always be able to schedule a common time for synchronous collaboration. Therefore, support is required for collaborative activities that consist of a series of sessions, centered around *collaboration artifacts*, such as annotated data, paper drafts, and design sketches, that should persist from session to session. When a session starts, any artifacts needed for the session should be available so that the participants can quickly get on with the work. For users who miss some sessions or a part of a session, mechanisms for providing collaboration context are needed so that the users can catch up on the missed work quickly.

• **Dynamic multi-group collaboration:** A collaboration may involve sharing of work across group boundaries. In UARC, one group of scientists may be more interested in data from radar instruments and another group in satellite image data. But, when interesting data is observed, group boundaries may need to quickly change, with scientists being able to dynamically move data from one group to another.

• **Information overload:** As CSCW systems are used by larger or multiple overlapping groups and collaborations involve increasing amount of on-line, changing data, mechanisms are needed to filter out unnecessary information and present overviews/summaries of shared data and updates to it, so that users can work efficiently and with as few distractions as possible. Limited size of desktop displays can also be a constraining factor, requiring efficient use of desktop real-estate in CSCW applications.

• **Collaboration in a wide-area network:** Scientists in the CBE collaborate over the Internet, with various bandwidths and qualities of connection. Furthermore, clients can often be unreliable, as compared to servers, in CSCW systems; clients may leave ungracefully (e.g., by turning off the computer). CSCW systems have to be robust against individual client failures and provide adequate performance even when some of the clients have poor bandwidth connections.

IV. Architectural Considerations for Groupware Systems

In this section, we discuss architectural considerations for groupware systems. In particular, we present the discussions with respect to the following performance requirements. While not meant to be comprehensive, the following requirements are based upon our experience with building a large number of collaborative systems of various scales and are found to be critical to the success of these systems:

• **Predictable performance for late-comers:** In a synchronous collaboration, participants may join an ongoing collaboration activity. A late-comer should be able to receive a consistent state of collaboration in a "predictable" amount of time – largely independent of client failures or of bandwidths to other clients in the system. Our experience with CSCW systems indicates that users expect a reasonable response time (i.e., limited largely by their bandwidth to the network and the size of the state) for state transfer when they join the system. Furthermore, existing collaborators should be able to continue their work while latecomers are joining.

• **High performance in interactive response time and latency:** Users expect groupware tools they use, e.g., group editors and whiteboards, to have a similar response time as single-user applications. Collaboration with other users should have minimal effects on the fluidity of users' interactions with their applications. Also, latencies should be low.

• **Persistence:** A collaborative activity may be both synchronous and asynchronous. It is often the case that collaborators may not accomplish all their goals in a single session; they may have to adjourn a session and reconvene at a later time. In such cases, it may be necessary to save a part or all of shared data used in a collaboration persistently to retrieve for a later session.

• **Synchronization:** The shared data in a collaboration should be consistently synchronized during the entire collaboration. Furthermore, wherever possible, collaborators should be allowed to access and modify the shared state concurrently without disrupting each other's work.

• **Robust collaboration:** A collaboration session should be robust. It should tolerate various failures of collaborators' host machines and network connections and continue to support the work of non-faulty collaborators.

Two key architectural considerations for groupware systems are: *the administration of shared state* and *the size of shared state*. The former refers to the organizational decision as to which collaborating processes maintain the consistency of shared data. The latter refers to the amount of shared application data in group work. In this section, we discuss existing approaches to these issues and describe the approach used in CBE.

A. Centralized vs. Replicated Administration of Shared Data

In general, there exist two approaches to managing shared data in groupware systems: *centralized* and *replicated*. With a centralized approach, there exists only a single copy of the shared data. A well-known process, often called *the server*, controls access to it. Many approaches to centralizing shared data exist. Application sharing systems, such as Microsoft NetMeeting and XTV [4], are inherently centralized because only a single instance of the application runs, but its GUI display goes to multiple users. User input to the application, such as mouse-clicks, is sent to the application process for processing, and the resulting changes to the interface of the application are propagated to client sites. Jupiter [5], an outgrowth of the popular MUD [6], also allows sharing of applications. In Jupiter, an application consists of application objects that define the functionality of the application and graphical user interface objects that allow users access to the application objects. When the application is shared, the application objects remain at the server site, and only the user interface objects are replicated at client sites. User input to the application is sent to the server site for processing, and the results are returned to the client sites. NSTP [7] also takes a centralized approach in which shared objects are centrally stored at the NSTP server site, and collaborating processes acquire remote accesses.

With a replicated approach, collaborating processes manage shared data among themselves; the processes have their own copies of shared data and coordinate concurrent accesses in order to keep the state of the replicas consistent. Most group editors adopt a replicated approach in which text buffers are replicated at each site, and editor processes run a synchronization algorithm among themselves in order to maintain the consistency of the replicated buffer. For example, DistEdit [8] provides locks of varying granularity that can consist of any number of consecutive characters, and the size of a lock dynamically grows as a user continues to type. The dOPT algorithm of Grove [9] uses an optimistic strategy in that it locally transforms remote operations based on the state of the local copy of the buffer; no locks are required. Both GroupKit [10] and MMConf [11] provide a general-purpose, replication-based infrastructure for running real-time, collaborative conferences; in each, an instance of a collaborative application runs at each collaboration site, and application replicas exchange messages among themselves for replicated state consistency. GroupKit allows customized conference registration procedures whereas MMConf provides a set of lock-based synchronization mechanisms upon which conference-specific synchronization policies can be devised.

The advantages and disadvantages of centralized and replicated approaches are well documented in the literature [7, 11, 12]. Generally speaking, a replicated approach has an advantage over a centralized counterpart where the issues of robustness and user responsiveness are concerned. Because each collaborating process equally assumes the administrative responsibilities of managing shared data, failures of one or more processes are unlikely to bring down the entire collaboration. Furthermore, the user of a process can directly interact with the local copy of the shared data of the process, hence increasing the opportunities for providing performance in interactive responsiveness close to that of single-user application.

Because there are multiple copies of shared data in a replicated architecture, however, it is generally expensive to keep the state of shared data replicas synchronized. The synchronization algorithms in DistEdit and Grove take advantage of characteristics specific to the application domain of editing and hence can be difficult to use in a general class of applications. Algorithms based on the distributed transactions semantics [13, 14] are expensive to implement and may incur significant processing overheads, degrading response times. Indeed, the main advantage of a centralized approach is its ease of synchronization of shared data. Because there exists only a single copy of the shared data, concurrent accesses can be easily serialized by the server process.

Accommodating late-comers to a collaboration is easier with a centralized architecture than with a replicated counterpart in which a complex group membership synchronization protocol needs to be run among all the keepers of replicated data whenever a late-comer is introduced. The server in a centralized architecture can maintain the membership of collaborating processes and notify the existing clients of the new client only when an accommodation protocol is completed. In a replicated architecture, existing clients may have to be suspended while a copy of shared data is transferred to the newcomer. Furthermore, the system may have to ensure that group members agree on the group membership in order to maintain the state of shared data consistent. Of course, replication strategy, though more difficult to implement, has the the potential advantage of providing better response time and fault-tolerance.

B. Size of Shared State

Collaboration tools, such as shared whiteboards and group editors, inherently allow their internal state to be shared, i.e., accessed and/or modified, by multiple users. The size of shared state refers to the amount of shared application data, and depending on collaboration semantics, it may range from the entire application state to a small subset. For example, Microsoft NetMeeting, XTV [4], and SharedX [15] allow single-user applications to be shared among multiple users without any code modifications by multiplexing window manager events. In these systems, the entire state of an application, including its graphical user interface and the locations of the application's windows on the screen, is shared. On the other hand, NSTP [7] allows individual application objects to be shared. It provides a centralized repository, called a Place, in which shared objects, called Things, may be stored. Then collaborating processes acquire remote references to a set of Things and receive update notifications whenever the state of a Thing is changed.

The size of shared state has a significant impact on the types of collaboration that can be supported. For example, in a collaboration environment based on application sharing as in NetMeeting, XTV, and SharedX, security can be compromised if applications can read/write files. Furthermore, because the windows of a shared application may have to be identically located on the screen, participants have little control in management of screen display, which may, in effect, prevent them from doing private work.

In contrast, several toolkits such as NSTP [7], COAST [13], and DistView [16] allow applications to share internal objects. Applications based on these systems are "collaboration-aware" and can be designed to allow users to perform both private and shared work. They can also allow users to control their individual displays, perhaps seeing the shared application data in different ways, and to interact in parallel. Performance is also usually an order-of-magnitude higher because internal application state changes are propagated (usually small) as opposed to user-interface updates (usually larger as they can involve bitmaps and complex graphics operations). At present, applications designed to be "collaboration-aware" are much more scalable to large groups than single-user applications used in application-sharing environments such as NetMeeting.

Another issue is that the size of shared state may have to change at runtime. In addition, depending on collaboration semantics, it may be difficult to determine the proper size in advance. For example, many window-based editors allow simultaneous editing of multiple documents, and a group editor with similar capabilities may support editing of both private and shared documents. However, the group editor cannot determine in advance shared documents to be coauthored. The decision is up to users and is made at runtime. For example, a user may decide to bring a private paper to the attention of the user's colleagues for their comments. Conversely, the owner of a shared document may want to stop the group-editing on the document in order to finalize the document. Providing support for such dynamics in shared state size is important in enhancing the effectiveness of collaboration.

C. Shared Data Management in CBE

The shared data management in CBE is based on the notion of *stateful group communication* [17], in which the underlying communication subsystem directly manages shared application data for its clients. Our experience with groupware systems in wide-area networks is that clients, such as those run from Web browsers, can often crash, be frequently stopped or terminated by users (such as by closing the browser window), or have poor connectivity to the network. Consequently, clients cannot be expected to reliably perform administrative tasks such as state transfer. If a client chosen for state transfer experiences a link failure or crashes, the new client would have to wait until another client is chosen, and so on.

Existing group communication subsystems, such as ISIS [18], Transis [19], and Consul [20], provide important message ordering and consistent membership view guarantees and have been successfully used to support building of robust and replicated services. They have also been used to manage replicated state among clients in groupware systems; for example, DistEdit [21] uses ISIS as the underlying communication mechanism. However, the use of this approach is limited in groupware systems due to the inherent assumption that group members are generally available and that group membership does not frequently change.

In contrast, stateful group communication method has the underlying communication subsystem manage shared application data for its groups in order to provide reliable join and state transfer. Of course, like clients, the communication servers may experience network failures or crash. However, the assumption is that as a service provider, the servers can be made more reliable than clients and that their runtime environment can be better controlled, e.g., resources can be dedicated. Furthermore, other measures can be taken to counter server crashes, e.g., message logging and client rejoin support [17]. Message logging can minimize the amount of data loss in case of a crash, whereas

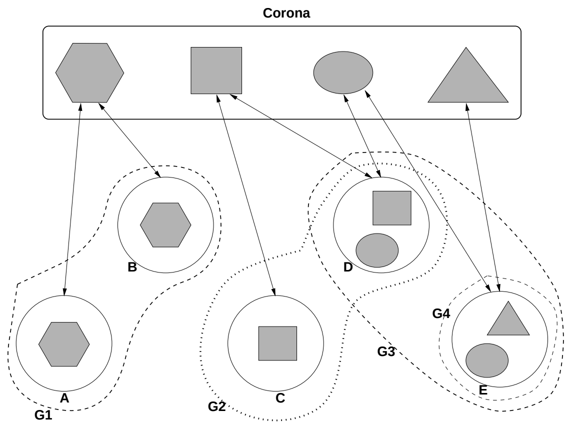

Fig. 3. Groups and Shared States in Corona. Circles represent collaborating processes, dotted lines depict groups, and shapes represent shared states. In the figure, Processes A and B belong to Group G1. Process D belongs to both Group G2 and Group G3, and Process E belong to Group G3 and Group G4. Group G4 is a singleton group in that it only has one member. Note that Corona as well as client processes have copies of the shared state of a group.

client rejoin support allows clients to rejoin their collaboration groups without having to start over from scratch, which helps reduce usage overhead.

In CBE, stateful group communication is provided by Corona [22], a group communication server. Corona models a set of collaborating processes as a group. A group can define and update shared application state at Corona by broadcasting *state update messages*. A state update message can contain an incremental state update for a shared object or the object's complete state. Corona logs state update messages as they are broadcast. Logging is based on the semantics of state update message; for example, a state update message with the complete state of a shared object replaces any logged messages that contain incremental state updates and old state of the object. Note that Corona does not (or cannot) decode message contents; the interpretation of messages contents is up to clients. This client-based semantics [7] of Corona's shared state management is designed to support a wide variety of collaborative applications.

Figure 3 graphically illustrates the management of shared state in Corona. Note that in addition to Corona, each client has a copy of shared state. This is due to Corona's state transfer service; when a client joins a group, a snapshot of logged messages is sent to the new client. If new messages are broadcast, Corona forwards the messages to the client. This join and state transfer protocol is both reliable and unobtrusive in that the new client can receive its group's shared state even when the other clients are unavailable due to network failures or crashes and that existing clients can continue to work while a new client is still joining.

That clients have a local copy of shared state provides low response times for local operations, an important factor in interactive collaborative environments. Corona also provides locks to those clients that need exclusive access to shared objects. The Corona services are open in that clients can choose only the services they need. For example, a group working on a shared whiteboard may not need to synchronize concurrent accesses to the whiteboard. Such a group does not need to subscribe to Corona's lock services.

D. Size of Shared State in CBE

In CBE, the concept of *selective window sharing* is employed to allow the size of shared state to be tailored to collaboration need. Selective window sharing allows on-demand sharing of application windows. Today's applications have sophisticated graphical user interface that consists of multiple windows, and a window graphically displays part of application state. With selective window sharing, users decide which windows to share at runtime. When a window is shared, the application state displayed in the window is also shared. The shared window then provides a synchronized view of the shared state to collaborating users. Shared windows can co-exist with "un-shared" windows, which allow users to perform private work.

Because users decide which windows to share at runtime, the size of shared state can be tailored to collaboration need. For example, if the sharing of entire application interface is desired, as in XTV, all the application windows can be shared. A finer granularity can be provided by sharing only the windows that display application state that need to be shared.

Figure 4 graphically illustrates selective window sharing. In order to share a window, the user first *exports* the window to a public repository, i.e., Corona in CBE. Subsequently, the user's coworkers can import the window from the repository. Once the window is imported, it maintains a synchronized view of shared data at all times. In order to provide low response times, an imported window is replicated at import time and is functionally-equivalent to the exported window. The user can interact with an imported window as he or she would with regular windows. An imported window can be positioned on screen independently of the exported window, can be iconified, etc.

Selective window sharing is incorporated in DistView [16]. DistView is an object-oriented toolkit that provides a library of *group-aware* objects. A group-aware object can be used both for private and collaborative work. When shared, it is replicated at collaboration sites and maintains a consistent state by exchanging state synchronization messages with its replicas. The application model in DistView is based on the well-known programming paradigm, called Model-View-Controller or MVC [23]. In MVC, an application consists of three types of objects: models, views, and controllers. A model manages application state. A model updates its state in response to user input and notifies one or more views. A view is associated with a model and is responsible for graphically displaying its model's state on the screen. A controller is associated with a model and is responsible for notifying its model upon sensing user input, e.g., mouse clicks and keyboard entries. Views can contain other views and are organized in a hierarchy. A window

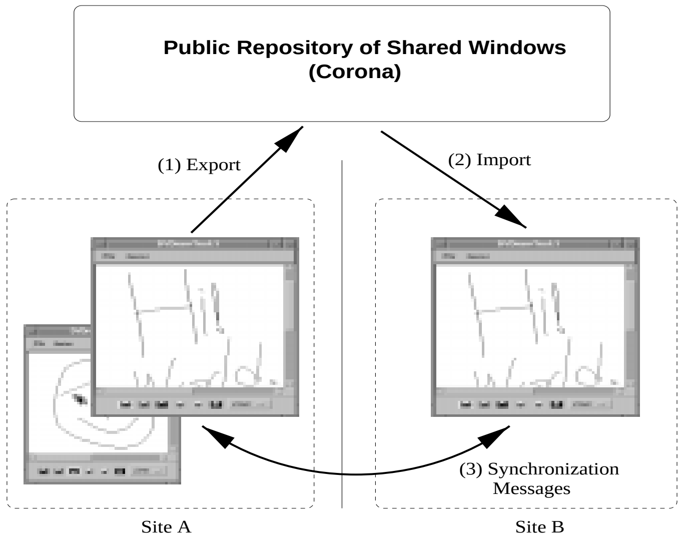

Fig. 4. Selective Window Sharing. A shared window is first exported to the public repository and then imported. An imported window is replicated and functionally-equivalent to the exported window. The shared window displays a synchronized view of shared application data.

is a top-level view in a view containment hierarchy. When a window is shared, DistView replicates the views in the window's hierarchy and their associated models.

DistView currently requires programmers to write code that acquires appropriate locks to ensure consistency of replicas. All the locks should be acquired together before a user action because no explicit support is provided for aborting a partially-executed action if some of the locks cannot be acquired. The COAST system [13], based on Smalltalk, addresses this problem by modifying the Smalltalk virtual machine to support optimistic transactions and aborts of transactions. The DECAF system [14], like COAST, also uses optimistic transactions, but hides the complexity of optimistic transactions from programmers by giving a standard set of objects, such as Integers, Strings, etc., that have rollback capability built in. All these systems are relatively new and more experience is needed with them to evaluate the best programming model for building CSCW systems.

V. Reducing Cognitive Overhead

An infrastructure for general-purpose, large-scale collaboration over a wide-area network, such as CBE, needs to support multiple groups of users who have different goals and need to perform a wide range of activities. In order to facilitate such collaborative work, a system should address the following issues:

• **Information Overload:** A group of users should ideally be presented only with information relevant to their work. Any other information should be "hidden" from the group's view of the system, unless otherwise needed. Blindly presenting all the data in the system, irrespective of the needs of different groups, would unnecessarily overwhelm users and degrade the usefulness of the system as a general-purpose collaboration infrastructure.

• **Scalable Collaboration Environment:** The collaboration environment provided by the system should be scalable. In particular, it should allow a group of users to simultaneously pursue different goals, form sub-groups, and bring in the tools they need at runtime. Further, users should be able to arrange their desktop displays according to individual needs. Scalability can also be facilitated by the support for asynchronous collaboration, such as the session record-and-replay capability [24]. In a large-scale collaboration where participants may be distributed across different time zones and/or continents, it is likely that participants sometimes join an ongoing group session or entirely miss a session. In such cases, asynchronous collaboration support such as the ability to record the minutes of the session and replay them at a later time would greatly facilitate those who were absent to quickly catch up with the current status of group work.

In this section, we describe various approaches to these issues in existing collaborative systems.

A. Dealing with Information Overload

The basic approach to solving the problem of *information overload* is to first evaluate incoming streams of information against users' needs before delivery. Then unwanted information may be filtered out and/or the information that does reach users may get prioritized according to users' specifications. Many approaches are possible to specifying filtering criteria. For example, a user may specify keywords or phrases to look for in information contents [25]. CLUES [26] relies on users' everyday work environments that may include electronic calendar, log of telephone calls, and rolodex. For example, if a user has made a phone call to an area with a certain area code, CLUES will assign a high priority to any subsequent electronic mail from people whose telephone numbers in the user' rolodex have the same area code.

Collaborative filtering [27] involves *humans* in the evaluation of incoming data streams. For example, the filtering process in Tapestry [27], an electronic mail system, takes into account *human reactions* to received mail messages. The reactions are captured in *annotations* to mail messages, and rule-based filters may be created based on the contents of annotations as well as the contents of mail messages. In Group Lens [28], a news reader, users assign numeric scores to each news article they read. The scores of different users are correlated with each other in order to find users who share similar taste. Before sending a news article to a user, GroupLens first finds the existing scores on the article and computes a score that predicts how much the recipient would like the article. The computation is based on a heuristic that "people who agreed in their subjective evaluation of past articles are likely to agree again in the future."

A major drawback of collaborative filtering is that it currently requires a substantial amount of pre-planning on the part of users and extensive set-up time. In Tapestry, when writing a filter, a user has to know in advance whose annotations are to be included in the query. Hence, there needs to be a prior agreement among a group of users on types of annotations and group membership. However, determining the exact membership of the group in advance may be difficult as group work may require different expertise at run-time. GroupLens is learning-based in that the more article ratings it has, the better its predictions. Therefore, it may take substantial time before GroupLens produces useful predictions.

In CBE, the *Session Manager* [1] allows a group of users to organize their shared workspace into *rooms* that can be used for organizing their work and filtering out unwanted information. Figure 2 shows the user interface to the Session Manager. A room represents a workspace into which users may put arbitrary *objects* that represent users' work artifacts. A room may be either *public* or *private*. A public room is a shared workspace for a group of users who collectively work on the objects in the room. A private room is a private workspace in which a user can work in private without being exposed to others' public work.

Rooms may be organized as a hierarchy, in which topmost rooms represent different user communities, the rooms at one level down represent organizations or groups within a community and so on. Such a room hierarchy can assist users in organizing their joint work and filtering out events that do not pertain to their work.

CBE currently does not provide much filtering of information within a room. Little work has been done on this yet – most room-based systems such as MUDs attempt to provide all information to users. The concept of collaborative filtering may be applicable there, for instance to ensure that important chat messages are highlighted in the chat window (any user could perhaps be allowed mark a chat message as important), or to indicate which windows in the shared workspace are the focus of attention.

B. Scalable Collaboration Environment

The Session Manager in CBE is an example of a system that aims to be scalable to the dynamic demands of collaborative work, in terms of types of user groups, types of objects on which collaboration occurs, and the types of tools used for collaboration. Below, we describe some of its features that facilitate scalability. We also compare its approach to other key systems such as TeamWave [29] (previously known as TeamRooms).

CBE allows one user to be present in multiple rooms simultaneously. This allows users to participate in multiple collaborations, making better use of their time. In UARC, scientists use rooms to partition their work by data sources, and participate in multiple rooms simultaneously so that they can observe the activity in various rooms.

Objects may be added/deleted to and from rooms as needed. Objects can be of different types and invoked from the session manager to view or modify them. For example, if the object is a URL, then on invoking that object, a web browser starts up and displays the page that the URL refers to[1]. If the object represents a shared document, a CBE-compatible group editor starts up and displays the current contents of the document. Note that the users in a room are not required to run all the objects in the room; instead, they run only the objects of their interests. This supports the different needs of individual users within the same collaboration group. For example, some members of the group may want to work on a shared paper, whereas the others are jointly creating a design diagram on a shared whiteboard. In order to focus on their group-editing, the co-authors may not want to participate in the drawing session and vice versa.

CBE supports persistence of state across sessions to allow participants who miss a session to catch-up on missed activity and collaborate asynchronously. Rooms and objects persist in the sense that they outlive group sessions. When a group of users leave a group session and reconvene at a later time, the rooms for their work still exist in the server, and the objects are in the same state as when whey last left them. This efficiently allows groups' work to span multiple sessions.

An object in a room can also be a *shared window*. If several users open a shared window object in a room, their windows are guaranteed to be synchronized. All operations, such as scrolling, window resizing, etc., are propagated among the copies of a shared windows. Shared windows in CBE are implemented via the model-view sharing capability of the DistView toolkit discussed in Section IV-D.

In order to facilitate data sharing between different activities and/or different groups of users, objects may be either copied or moved between rooms. When an object is copied to a room, a new instance of the copied object is added to the room. The new instance has the same state as the original at the time of copy, but the subsequent updates on it have no effect on the original. Moving an object is similar to copying an object except that the object is removed from the room from which it is moved.

TeamWave [29] is another system that uses the room metaphor for representing a shared workspace. In TeamWave, a group of users is given a shared window manager in which the windows of individual collaboration tools used for their group work are displayed. Figure 5 shows an example of such a shared window manager. Users may independently view different parts of the shared window manager while working on different aspects of their work. New tools may also be added and removed as needed.

The main advantage of the TeamWave's approach is that a shared window manager increases the sense of working together for a group of users. Because the shared window manager is shared only by the members of the group, the users feel tightly connected with each other. However, the shared window manager reduces flexibility in individually arranging users' desktops as it does not allow users to independently place collaboration tools on the screen. When a user participates in multiple groups, the user's desktop may become too crowded with multiple shared window man-

[1]The Session Manager currently does not support the collaborative viewing of URL pages

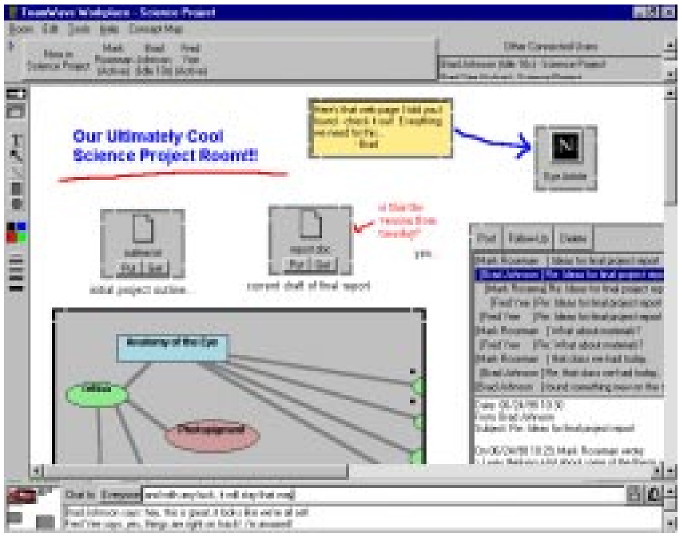

Fig. 5. A Shared Window Manager in TeamWave.

agers. Thus there is a tradeoff between extent of view synchronization of rooms and the control users have over their desktop. Further experience with these systems is required to determine the best compromise between these competing needs.

VI. Access Control

Proper access control is critical to the successful adaptation of the CSCW technology. Collaborators should be able to work without unauthorized users prying into their interactions, and their physical as well as intellectual properties should be protected from unauthorized access. Furthermore, doing so should not incur excessive overheads.

Traditional approaches typically bind user names to a set of data objects through a set of access rights. A access control list, for example, associates a set of tuples (S, R) to a given object, O, where S is a list of user names, and R is a list of access rights, each of which specifies the access right of the corresponding user in S on O.

However, an open issue is whether traditional approaches are adequate for collaborative systems. Effective collaboration is often accomplished by collaborators spontaneously reacting to each other's actions [30]. As such, the participants in a collaborative activity should not be restricted to a single role, as doing so would limit their contributions [31]. For example, if a user is limited to a role of observer in a meeting, the user would/could not express his or her ideas at appropriate moments. Furthermore, things often do not work out as planned, and collaborators may have to perform unexpected tasks as a collaboration evolves over time. For example, the unexpected early departure of a secretary in a meeting would require the delegation of the responsibilities of the secretary to one of the meeting participants on the fly, and the selected person would have to perform tasks of both the secretary and the participant thereafter.

Capability or access control lists do not provide any provisions for dealing with such dynamism and spontaneity inherent in collaborative activities. As a result, collaborators have to assume the responsibilities of updating access control lists depending on the collaboration context. For example, when a user assumes a new role, an authorized user may have to manually update a capability or access list so that the user can access data objects needed for the new role. Such overheads involving the administrative aspects of collaboration, rather than collaboration itself, disrupt the fluid interactions among collaborators and can significantly degrade the effectiveness of collaboration.

In short, an access control mechanism for collaborative systems should be able to incorporate the dynamism inherent in collaboration. At the minimum, it should:

- allow collaborators to effortlessly switch between roles. Role switching should be easy and should not incur too much overheads.
- allow collaborators to assume multiple roles simultaneously.

In this section, we discuss approaches found in current collaborative systems.

A. Inheritance and Dynamic Roles

Suite [32] is a general framework for building collaborative applications. It allows dynamically defining and configuring a wide range of collaborative interactions by providing an access control framework that allows fine-grained access right specifications. The framework extends the classic access control matrix model by Lampson [33] by structuring each of the *subject*, *access*, and *object* dimensions of the model as an inheritance-based hierarchy. In Suite, a subject in an access control list may be either a user name or a *role*.

Role-based access control (RBAC) has been advocated in several systems [34, 35]. The basic concept of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. Roles correspond to job characteristics in an organization and users are given roles according to their responsibilities and qualifications.

Access authorization on objects can then be specified for roles without the need for specifying each user's access to the objects.

Examples of roles in literature include *author*, *commentor*, *principal*, and *observer*. Because roles are assumed to have semantics well understood in collaborators' communities or organizations, the collaboration context can be deduced from the participant roles present in a collaboration. For example, in a group-editing session, the presence of multiple *author* roles effectively conveys that a paper is being concurrently edited by multiple users.

Suite organizes roles as a hierarchy in which child roles have more specific definitions than their parents' roles. If a user takes a role, then he or she inherits any rights associated with the parent of the role. A child role may also have rights that override the rights of its parent role. Users may take multiple roles and change roles dynamically. If the multiple roles of a user have conflicting rights, the access specification of whichever role that appears first in the access control list is used. Suite users may extend the standard role hierarchy by defining custom roles that reflect the particular collaboration context.

Intermezzo by Edwards [30] provides access control based on a richer collaboration context than is supported by existing systems. Edwards argues that the traditional role-based mechanism does not provide enough flexibility and responsiveness to dynamic collaboration environments. In particular, roles in many systems are *static* in that a set of roles are pre-defined prior to collaboration and that role membership is specified in terms of user names. As a result, static roles require users to anticipate various situational variables in their collaboration, a difficult task.

Intermezzo supports *dynamic roles* in addition to static roles [2]. Instead of a list of user names, a dynamic role is specified with attributes that embody situational clues about collaboration context. The attributes are incorporated in a *predicate function* associated with the dynamic role. Whether or not a particular role has access rights to some data objects is not established at the access control specification time. Rather, the predicate function associated with the role is dynamically evaluated when an access request is made.

Both Suite and Intermezzo are likely to require significant administrative overheads. Suite can require substantial set-up efforts before collaboration can actually commence. Defining the role hierarchies in an organization, the set of users, mapping between users and roles and the access right of each role can be nontrivial.

In Intermezzo, it can be difficult to define dynamic roles and write associated predicate functions for all possible collaboration contexts. Verifying the correctness of the predicate functions can also be problematic.

B. Access Control in CBE

In CBE, each room is associated with its access control list. The access right specification is role-based, and the supported user roles include *administrator*, *member* and *observer*. Each user role is mapped to its own access rights to rooms with a different level of service in collaboration. In order to support the spontaneity of group work, the system allows users to freely create rooms. Our policy allows any user to create rooms in order to provide a free collaboration environment to scientist users. However, the permission to create rooms can be specified in user roles by the system administrator if more strict access control is needed in an open environment. By default, a user who creates a room becomes the room's administrator. A room administrator decides who can enter the room and specifies access rights of users in the room. A member user may participate in a collaborative activity by running an object and make updates on the shared data used in the activity. A member user is not allowed to update the access rights nor delete a room, which are the privilege of the administrator of the room. An observer can also run an object (or run an application with the content of the object), but may only observe the updates on the shared data of the activity.

[2]The static roles are supported by means of access control lists

Since the observers cannot disrupt the collaborative activity in the system, member users feel comfortable with the idea of making the system openly accessible over the Internet. This model can be extended to specify access control on operations on objects which include rooms, room objects and actions where actions are associated with events on rooms or room objects [36]. Examples of operations on rooms are *enter a room*, *leave a room* and *list objects in a room*. The model allows user roles to be defined based on the semantics of the collaboration. Since the model is similar to the access control in Andrew File System (AFS), adapted for use in collaborative systems, users are likely to be familiar to the model. As a result, the model is simple to understand as well as easy to use. However, it lacks the richness of the functionality and flexibility found in Suite and Intermezzo.

VII. Collaboration Awareness

Collaboration awareness refers to information that enables collaborators to know what others are doing. The awareness information makes a collaboration more efficient by allowing participants to avoid the duplication of work and reduce conflicts. Providing adequate facilities for the awareness information is essential to effective computer-supported collaboration, as failing to do so may have users guessing what others are doing, leading to wasting their time on unproductive tasks.

Awareness is concerned with various aspects of collaboration. In particular, it can provide information about *participants* in a collaboration. Collaborative systems typically display a list of users in a collaboration and updates the list as users join or leave the collaboration. In addition to information identifying individual users, e.g., their names, the user list may convey pertinent information such as users' roles and participation status, e.g., the time of their last activity. The list also often includes contact information from their business cards, which, for instance, could be used to send electronic mail to individual participants. The shared workspace may also serve as a source of awareness [10]. For example, in group editing, the knowledge about the part of a shared paper each author is viewing may provide clues as to what each author is currently working on.

Awareness information should be seamlessly incorporated into users' work environments and be easily accessible. The awareness information should be provided in such a manner that it is relevant to collaboration activities at hand and is easily distinguishable from actual work contents. The awareness information should be automatically gathered and distributed whenever possible. Entirely delegating the responsibility of generating the awareness information to collaborators tends to make it unlikely that collaborators will generate/use the information and will render awareness features counter-productive. To this end, Dourish and Bellotti [37] argue for the *shared feedback* approach. Rather than relying on external channels for communicating awareness information, such as electronic mail or telephone, the awareness information should be "presented in the same shared workspace as the object of collaboration." The common feature of group editors that allows users to see the texts of remote writers in real time is an example of the shared feedback approach.

A. Awareness Widgets

Groupkit [10] is a toolkit for building a general class of collaborative applications and includes a number of *awareness widgets* for use in GroupKit-based applications. An awareness widget is a user interface object, such as windows and scrollbars, that is specifically designed to provide awareness information. Examples of such widgets in Groupkit include *radar view*, *multi-user scrollbar*, and *telepointer*. In Groupkit, the contents of a shared workspace are displayed in a loosely-synchronized shared window, somewhat like the root window in a windowing system. The main goal of the GroupKit awareness widgets is to help collaborators locate each other in the shared workspace.

A radar view shows collaborators' locations in the shared workspace by representing their views as rectangles in a miniaturized version of the shared workspace. The rectangles are of different colors, each corresponding to a collaborator. A radar view is often used with telepointers. A telepointer is a trace of a remote mouse cursor movements, and each rectangle in a radar view may use a telepointer to show the mouse movements of a user whose view of the shared workspace is represented by the rectangle. A multi-user scrollbar also identifies users' locations in a shared workspace. Instead of using a miniaturized shared workspace, however, the multi-user scrollbar is directly incorporated into the shared window that displays the shared workspace.

The study [38] conducted by the Groupkit designers indicates that users heavily rely on the awareness information provided by awareness widgets, especially the radar view, to coordinate their work with the others. Because a radar view effectively indicates data artifacts that each collaborator is presently accessing, collaborators can establish the context of their colleagues' work and plan their own work around accessed artifacts in order to avoid conflicts. The study also reports that the information provided by most of the awareness widgets is not distracting to their work and is worth the screen real-estate the widgets take up. In short, the study finds that the awareness information is effective when provided appropriately.

B. Awareness in CBE

The Session Manager provides a list of all the users in the system as well as a list of users in a room. The lists specifies user names and locations, i.e., the names of host machines they are on. The list of entire users is provided on demand; a user has to click a "who" button on the Session Manager, at which point a separate window containing the list is displayed. If the user leaves the user list window open, the list is automatically updated as users enter and leave the system. A user may find the current users in a room by simply clicking on the icon for the room in the Session Manager. The list of the users in the room are displayed within the Session Manager window, and this list is also automatically updated as users enter and leave the room. The logic behind the different supports is that the awareness of the presence of the entire users in the system may not always be needed for a group's work and hence is provided on the need-to-know basis, whereas the awareness of the current group membership is relatively more important to the group's work and thus is incorporated into the main Session Manager window.

When a room is created, it is named. Because a room name is an arbitrary string, it may contain any pertinent information that represents the purpose of the room. For example, a room named "CBE Software Development" indicates that the room is primarily for CBE application developers. In this way, users can be aware of what others are doing in each room throughout the system. Also, a search for the room of the particular interest can be done by using awareness information like room name.

The awareness information provided by the Session Manager can be extended to include the list of rooms that a user is in and the list of objects a user is running. Furthermore, users can assign different colors to themselves. Then a user could ask the Session Manager to find the rooms or objects another user is in or running respectively, and the Session Manager shows appropriate icons in the color of the other user. The Session Manager could also provide a radar view that shows clusters of rooms and object icons with each cluster representing the rooms or objects a user is in or running respectively. Such a radar view would provide an instant overview of a user's location in the Session Manager and/or the user's current activities. In the current implementation, the Session Manager can only display information about a single user at a time because of the practical limitation that an icon cannot show multiple colors. However, this approach does not demand extra screen real estate, which is an important consideration in the UARC's use of CBE, where scientists literally have tens of data viewer windows open simultaneously.

The shared windows, introduced in Section V-B may be viewed as a collaboration widget. Because all the physical attributes of a shared window as well as the view of data displayed in the window are kept synchronized, the shared window may be considered as a workspace widget in which users are always located in the same place and look at the same thing. Thus the shared window helps reduce the need for additional collaboration widgets such as radar views or multi-user scrollbars.

VIII. Conclusions

In order to enhance the effectiveness of computer-supported collaboration, collaborative systems should address distributed systems issues in such a way that the solutions are context-sensitive to the joint work of a group of users. In this paper, we have identified data management issues critical to CSCW and discussed example approaches to these issues found in collaborative systems. The issues include: 1) the architecture of collaborative systems, a crucial goal of which is to allow users to access shared data without having to pay significant performance penalties, even when they interact over a wide-area network, 2) selectively delivering and/or prioritizing data according to users' needs, and 3) enabling users to know the activities of their colleagues in order to avoid conflicts and duplication of work.

The discussed approaches are evaluated in the context of the Collaboratory Builder's Environment (CBE), a general infrastructure for supporting large-scale collaborations over a wide-area network. We find that although the basic ideas behind these approaches were useful in CBE, the direct application of any of these approaches is difficult. The reason is that while the approaches do solve targeted problems, they also make implicit assumptions that do not hold in specific applications. For example, writing filters in Tapestry requires prior knowledge about the membership of a collaboration group, which, in practice, may not be known until collaboration is well under way. The filters are also written in an SQL-like query language, which average computer users are not familiar with and thus make dynamically creating filters in response to changing collaboration needs difficult in practice. This shows that a significant gap remains in addressing the requirements of collaborative systems and that designers have to make design tradeoffs that are often difficult to evaluate.

Current research issues include increasing the usability and scalability of CSCW systems. For example, in CBE, we are examining issues in providing better support for tailoring the Session Manager based on the needs of individual groups. That is, users should be able to identify their needs so that they do not have to deal with rooms and objects other than the ones relevant to their joint work. However, specifying the needs should not incur much overhead, and the specifications should be allowed to change dynamically as the group's work evolves over time. These facilities would increase the focus of a group's work and allow users to freely interact with each other.

To improve usability, collecting and providing data so that *closure in dialog* can be achieved in discussions among users is another important, open issue in CSCW systems. Often a user is not aware that other users are paying attention, even if they are displayed in the user list of a system. In a meeting where participants are in the same physical room, eye contact and verbal cues often serve to tell users that they can start speaking, that others are listening, and when to stop, etc. Without those cues, in a distributed CSCW system, users often end up getting frustrated (e.g., if a user asks a question but no one responds, it may not be obvious whether no one has read the question or no one wants to respond). We feel that the infrastructure can provide some support for achieving closure, by a judicious use of additional media (audio, video) and by tracking activity level of users so that a user knows if other participants are paying attention. This, however, has obvious tradeoffs with privacy issues.

Scalability of the communication infrastructure for CSCW systems is another important aspect. The current implementation of CBE employs a centralized server, thus limiting the scalability of the communication infrastructure to medium-size groups [3]. Further, in a wide-area network, the centralized implementation may not be able to provide "equal" performance for all users. For example, if the server is running in Michigan, the users in California may experience longer delays or more frequent failures in received updates than those in Michigan. Replication of the server would help, but that can degrade the interactive responsive times and latencies, as messages may have to go through multiple instances of the server, leading to users experiencing longer delays in observing each other's actions. Hence, the distributed design of the server should strike a balance between the scalability and responsiveness of the system. In addition, it should be designed to support both synchronous and asynchronous work and not be dependent on the reliability of individual clients.

Scalability of the user-interface is another important aspect. It is not clear how to convey all the shared data and awareness information to users. Scientists using the CBE have noted that they would like to be able to work in multiple rooms simultaneously and be aware of activities of other users in those rooms, but have problems with screen real-estate. For making effective use of screen real-estate, providing flexibility to users in configuring the windows on their screens appears to be useful. On the other hand, that makes providing awareness information about other users' work more challenging because different users are looking at potentially different windows. Use of multimedia channels, such as audio, may need to be investigated as additional means to convey awareness information, so as to keep demands on screen real-estate low.

Acknowledgments

Several colleagues from the UARC project influenced the ideas described in this paper, including Dan Atkins, Tom Finholt, Farnam Jahanian, Robert Hall, Tim Killeen, Radu Litiu, Jason Breslau, Gary Olson, Sushila Subramanian, Terry Weymouth, and Gwobaw Wu. We also thank many domain scientists who have provided feedback on the use of the UARC system for supporting team science. This work is supported in part by the National Science Foundation under cooperative agreement IRI-9216848, the National Science Foundation grant ECS-94-22701, IBM Partnership Award, and Intel.

References

[1] J.H. Lee, A. Prakash, T. Jaeger, and G. Wu, "Supporting Multi-user, Multi-applet Workspaces in CBE," in *Proc. of the Sixth ACM Conference on Computer-Supported Cooperative Work.* Nov. 1996, pp. 344–353, ACM Press.

[2] S. Greenberg and D. Marwood, "Real-Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface," in *Proc. of the Fifth Conf. on Computer-Supported Cooperative Work*, Chapel Hill, North Carolina, 1994, pp. 207–217.

[3] C. R. Clauer et al, "A Prototype Upper Atmospheric Research Collaboratory (UARC)," *EOS, Trans. Amer. Geophys. Union*, vol. 74, 1993.

[3] Our experiments show that the current implementation can accommodate about 40-50 users, whereas scientists have indicated that they would like the system to support an order of magnitude higher.

[4] G. Chung, K. Jeffay, and H. Abdel-Wahab, "Dynamic participation in computer-based conferencing system," *Journal of Computer Communications*, vol. 17, no. 1, pp. 7–16, January 1994.

[5] D. Nichols, P. Curtis, M. Dixon, and J. Lamping, "High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System," in *Proceedings of UIST '95*, Pittsburgh, PA, 1995, pp. 111–120.

[6] P. Curtis, "Mudding: Social Phenomena in Text-Based Virtual Realities," in *Proceedings of 1992 Conference on Directions and Implications of Advanced Computing*, Berkeley, Ca, 1992.

[7] J. F. Patterson, M. Day, and J. Kucan, "Notification Servers for Synchronous Groupware," in *Proc. of the Sixth ACM Conference on Computer-Supported Cooperative Work*. Nov. 1996, pp. 122–129, ACM Press.

[8] M. Knister and A. Prakash, "DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors," in *Proc. of the Third Conference on Computer-Supported Cooperative Work*, Oct. 1990, pp. 343–355.

[9] C. Ellis, S.J. Gibbs, and G. Rein, "Design and Use of a Group Editor," in *Engineering for Human-Computer Interaction*, G. Cockton, Ed., pp. 13–25. North-Holland, Amsterdam, September 1988.

[10] M. Roseman and S. Greenberg, "Building real time groupware with GroupKit, a groupware toolkit," *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 1, pp. 66–106, March 1996.

[11] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson, "MMConf: An Infrastructure for Building Shared Multimedia Applications," in *Proc. of ACM Conference on Computer Supported Cooperative Work*, October 1990, pp. 329–342.

[12] J. Lauwers, T. Joseph, K. Lantz, and A. Romanow, "Replicated Architectures for Shared Window Systems: A Critique," in *Proceedings of ACM Conference on Office Information Systems*, March 1990, pp. 249–260.

[13] C. Schuckmann, L. Kirchner, J. Schummer, and J. M. Haake, "Designing Object-Oriented Synchronous Groupware with COAST," in *Proc. of the Sixth ACM Conference on Computer-Supported Cooperative Work*. Nov. 1996, pp. 30–38, ACM Press.

[14] R. Strom, G. Banavar, K. Miller, A. Prakash, and M. Ward, "Concurrency Control and View Notification Algorithms for Collaborative Replicated Objects," in *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems (ICDCS 97)*, Baltimore, Maryland, May 1997, pp. 194–203.

[15] D. Garfinkel, B.C. Welti, and T.W. Yip, "HP Shared X: a tool for real-time collaboration," *HP Journal*, vol. 45, no. 4, pp. 26–33, April 1994.

[16] A. Prakash and H. Shim, "DistView: Support for Building Efficient Collaborative Applications Using Replicated Objects," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*. 1994, pp. 153–164, ACM Press.

[17] H. S. Shim and A.Prakash, "Tolerating Client and Communication Failures in Distributed Groupware Systems," in *Proc. of Symp. on Reliable Distributed Systems*. Oct. 1998, To appear.

[18] K. P. Birman and T. A. Joseph, "Exploiting Virtual Synchrony in Distributed Systems," in *Proc. of 11th ACM Symp. on Operating Systems Principles*, Austin, TX, Nov. 1987, pp. 123–138.

[19] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "Transis: A Communication Sub-System for High Availability," Tech. Rep. TR CS91-13, Computer Science Dept., Hebrew University, April 1992.

[20] S. Mishra, L. L. Peterson, and R. D. Schlichting, "Consul: A Communication Substrate for Fault-Tolerant Distributed Programs," *Distributed Systems Engineering Journal*, vol. 1, no. 2, pp. 87–103, Dec. 1993.

[21] M. Knister and A. Prakash, "Issues in the Design of a Toolkit for Supporting Multiple Group Editors," *Computing Systems – The Journal of the Usenix Association*, vol. 6, no. 2, pp. 135–166, Spring 1993.

[22] H. Shim, R. Hall, A. Prakash, and F. Jahanian, "Providing Flexible Services for Managing Shared State in Collaborative Systems," in *Proceedings of the ECSCW European Conference on Computer Supported Cooperative Work*. 1997, pp. 237–252, Kluwer Academic Publishers.

[23] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk," *Journal on Object Oriented Programming*, pp. 26–49, August/September 1988.

[24] N.R. Manohar and A. Prakash, "The Session Capture and Replay Paradigm for Asynchronous Collaboration," in *Proceedings of the European Conference on Computer Supported Cooperative Work*. 1995, pp. 149–164, Klewer Press.

[25] T.W. Malone, K.R. Grant, F.A. Turbak, S.A. Brobst, and M.D. Cohen, "Intelligent Information Sharing Systems," *Communications of the ACM*, pp. 390–402, 1987.

[26] M. Marx and C. Schmandt, "CLUES: Dynamic Personalized Message Filtering," in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, 1996, pp. 113–121.

[27] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry," *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, December 1992.

[28] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*. 1994, pp. 175–186, ACM Press.

[29] M. Roseman and S. Greenberg, "TeamRooms: Network Places for Collaboration," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, October 1996, pp. 325–333.

[30] K. Edwards, "Policies and Roles in Collaborative Applications," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*. 1996, pp. 11–20, ACM Press.

[31] C.M. Neuwirth, D.S. Kaufer, R. Chandhok, and J.H. Morris, "Issues in the design of computer support for co-authoring and commenting," in *Proceedings of the Third Conference on Computer-Supported Cooperative Work*, Los Angeles, California, October 1990, pp. 183–195.

[32] H. Shen and P. Dewan, "Access Control for Collaborative Environments," in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, 1992, pp. 51–58.

[33] B.W. Lampson, "Protection," *ACM Oper. Syst. Rev.*, vol. 8, no. 1, pp. 18–24, 1974.

[34] D. Ferraiolo and R. Kuhn, "Role-Based Access Controls," in *Proc. of the 15th NIST-NCSC National Computer Security Conference*, Baltimore, MD, October 1992, pp. 554–563.

[35] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, February 1996.

[36] J. H. Lee and A. Prakash, "Malleable Shared Workspaces to Support Multiple Usage Paradigms," Tech. Rep. CSE-TR-370-98, Department of EECS, University of Michigan, August 1998.

[37] P. Dourish and V. Bellotti, "Awareness and Coordination in Shared Work Spaces," in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, November 1992, pp. 107–114.

[38] C. Gutwin, M. Roseman, and S. Greenberg, "A Usability Study of Awareness Widgets in a Shared Workspace Groupware System," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, October 1996, pp. 258–267.