Supporting Multi-User, Multi-Applet Workspaces in CBE

Jang Ho Lee, Atul Prakash, Trent Jaeger, and Gwobaw Wu
Software Systems Research Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
Email: {jangho, aprakash, jaegert, gwu}@eecs.umich.edu

ABSTRACT

Our experience with Internet-based scientific collaboratories indicates that they need to be user-extensible, allow users to add tools and objects dynamically to shared workspaces, permit users to move work dynamically between private and shared workspaces, and be easily accessible over a network. We present the software architecture of an environment, called CBE, for building collaboratories to meet such needs. CBE provides user-extensibility by allowing a collaboratory to be constructed as a coordinated collection of *group-aware applets*. To support dynamic reconfiguration of shared workspaces and to allow access over the Internet, CBE uses the metaphor of *rooms* as the high-level grouping mechanism for applets and users. Rooms may contain applets, users, and arbitrary data objects. Rooms can be used for both asynchronous and synchronous collaboration because their state persists across synchronous sessions. Room participants may have different roles in a room (such as administrator, member and observer), with appropriate access rights. A prototype of the model has been implemented in Java and can be run from a Java-enabled Web browser.

Keywords

Groupware, CSCW toolkits, shared electronic workspaces, Web-based collaboration, group communication, DistView, access control.

INTRODUCTION

We have created and used an experimental multi-institution testbed, called the Upper Atmospheric Research Collaboratory (UARC) [13], to examine issues in supporting synchronous, collaborative scientific work over wide-area networks. NeXT-based prototypes of UARC provide users with access to scientific data, allow them to maintain synchronized data display windows with telepointing using DistView [12], and provide collaborative tools such as *multi-user chat*.

Computer Supported Cooperative Work '96 , Cambridge MA USA
© 1996 ACM 0-89791-765-0/96/11 ..$3.50

The successful usage of the various prototypes of UARC has led us to redesign the facilities used in UARC so that they can be used for building additional collaboration environments. In this paper, we describe solutions used to meet the following needs of collaboration environments:

- support for multiple tools (e.g., domain-specific viewers, multi-user chat, shared whiteboard, etc.) in shared workspaces;
- support for both private and multiple shared workspaces and the ability to dynamically move both tools and data between the workspaces;
- access to collaboration environments over the Web from multiple platforms;
- provide user-extensibility. Users can add collaborative tools and Uniform Resource Locators (URLs) to their suites for collaboration purposes;
- support for user roles, particularly observers, so that open access over the Web to a collaboratory is not intrusive to the scientific work; and
- provide mechanisms for access control, particularly for use over the Web.

The work presented in this paper is a part of the Collaboratory Builder's Environment (CBE), a toolkit for creating extensible collaboration environments. It is being used to build a Java-based version of the UARC system.

To support extensibility, we structure client application software as a set of applets (mini-applications). The applets include domain-specific services (such as data viewers in UARC) and domain-independent tools (such as multi-user chat, whiteboard, etc.) that are useful for collaboration. CBE allows users to dynamically add new applets and URLs to their collaboratories.

CBE uses the metaphor of *rooms* to denote shared workspaces consisting of multiple applets and data (URLs). We describe high-level mechanisms for managing a collection of applets so that both private and shared rooms can be established in a simple way. Also users can dynamically create rooms. They can also move work-in-progress, including applets with their current state and data, between private and

shared workspaces, depending on the extent of sharing they desire with other users and the purpose for which rooms are being used.

CBE is designed to build collaboratories that can be accessed via standard Web browsers. All the client software, i.e., the applets, are written in Java so that it can be executed from a Java-enabled browser.

To provide both scalability (in terms of performance) and to allow users control over access to a room, we provide support for user *roles*. Users can have different roles, such as *observer*, *member*, or *administrator*, in a shared workspace. Users in different roles have different access control rights to a room and can be provided different levels of reliability guarantees in the presence of network failures.

We examine the impact of workspace-level access control features on the design of the lower layers of the communication infrastructure. In current systems, access control is largely based on trust — that users will not bypass workspace-level access controls by directly accessing underlying communication services or file systems. Once collaboratories are made available over the Internet and targeted to support potentially large number of observers, the issue of enforcing access control becomes more relevant.

The rooms in CBE can be used to support both asynchronous and synchronous work. Previously, we proposed a toolkit called DistView to support synchronous, window-level sharing of workspaces [12] in a single application. This paper addresses the issue of workspace sharing in a larger context when a user's workspace consists of multiple applets. We also extended DistView facilities in order to allow state of rooms to persist across collaborative sessions, so that rooms can be used for asynchronous collaboration.

The rest of the paper is organized as follows. We first give an example of a CBE-based collaboratory to illustrate its concepts from a user perspective. Next, we present the CBE's key concepts of rooms, applet-groups, applets, and users, as well as the relationships between them. Then, we discuss the current design and implementation. We also describe how access control can be enforced in insecure environments such as the Internet. Next, we present how the proposed architecture is used to meet various needs in a collaboratory. Finally, we present conclusions and directions for future work.

AN EXAMPLE OF CBE USE

Figure 1 illustrates the use of a multi-applet shared workspace in our current Java-based system that uses CBE. A session manager displays a list of the available rooms (top-right corner). The rooms shown include the Uarc room, automatically-created private rooms (one per user), and the Lounge room. The interface provides commands to allow users to join one or more rooms, leave a room, create rooms, add applets or URLs to a room, and to move URLs/applets dynamically from one room to another.

Users can enter rooms to collaborate with other users in the same room. They can also enter a room to simply view URLs or use applets in that room. URLs can be references to arbitrary objects (e.g., files, images, single-user applets, etc.) which can be display using a Web-browser. For URLs, only the reference to URL is shared among users in a room — not the view of the URL.

In Figure 1, the user has entered the Uarc room. The applets/URLs in the room are shown under Uarc. The user is currently using a group-aware applet, `chat` and a URL that instantiates `radar`, a standard Java applet. The room also contains another group-aware applet, `draw`, but that applet is not being used actively.

Rooms are persistent in the sense that they exist beyond a synchronous collaboration. Any URLs in rooms are saved across collaborative sessions. Also, the `chat` applet on bottom-left, provide a `Save Messages` command so that its state can be persistently saved in the room. If the same applet is later restarted, it starts with the last saved state. Thus, rooms can be used to support asynchronous collaboration.

The interface provides some basic group-awareness features, such as the applets available and the users who are present in a room. The list of all the users is also available because a Private room corresponding to each user is created when users login to the collaboratory. Users cannot enter other users' private rooms, but they can use them and the user list to send messages to other users.

The applets can be placed anywhere on the desktop. Applets display the room to which they belong to.

A user can move a group-aware applet, e.g., draw, from a shared room to their private room. This has the no effect on other users; it simply causes the user's applet to stop communication of the changes to its state to other users.

A user can also move a group-aware applet from a private room to a shared room. In such a case, the applet state becomes available to other users in that room. Other users in that room can invoke the applet to jointly interact with the applet's state.

Other interfaces can be built using CBE and in fact the current user interface is still evolving. CBE simply provides the infrastructure for building collaboration environments, and does not dictate a particular user interface choice.

DESIGN CONCEPTS

A shared workspace consists of four major components; *applets*, *users*, *applet-groups*, and *rooms*.

Applets

An *applet* is a small application that typically provides a GUI interface to a particular *user* for viewing/updating some data. A user would typically use multiple applets to collaborate with other users. It can be implemented as a process or thread on the user's desktop. As an example, if users wish to collaborate using a shared whiteboard and a multi-user chat with other users, a whiteboard applet and a chat applet will be created on each user's machine. An applet in our terminology is used in a somewhat different way than the same term in Java.

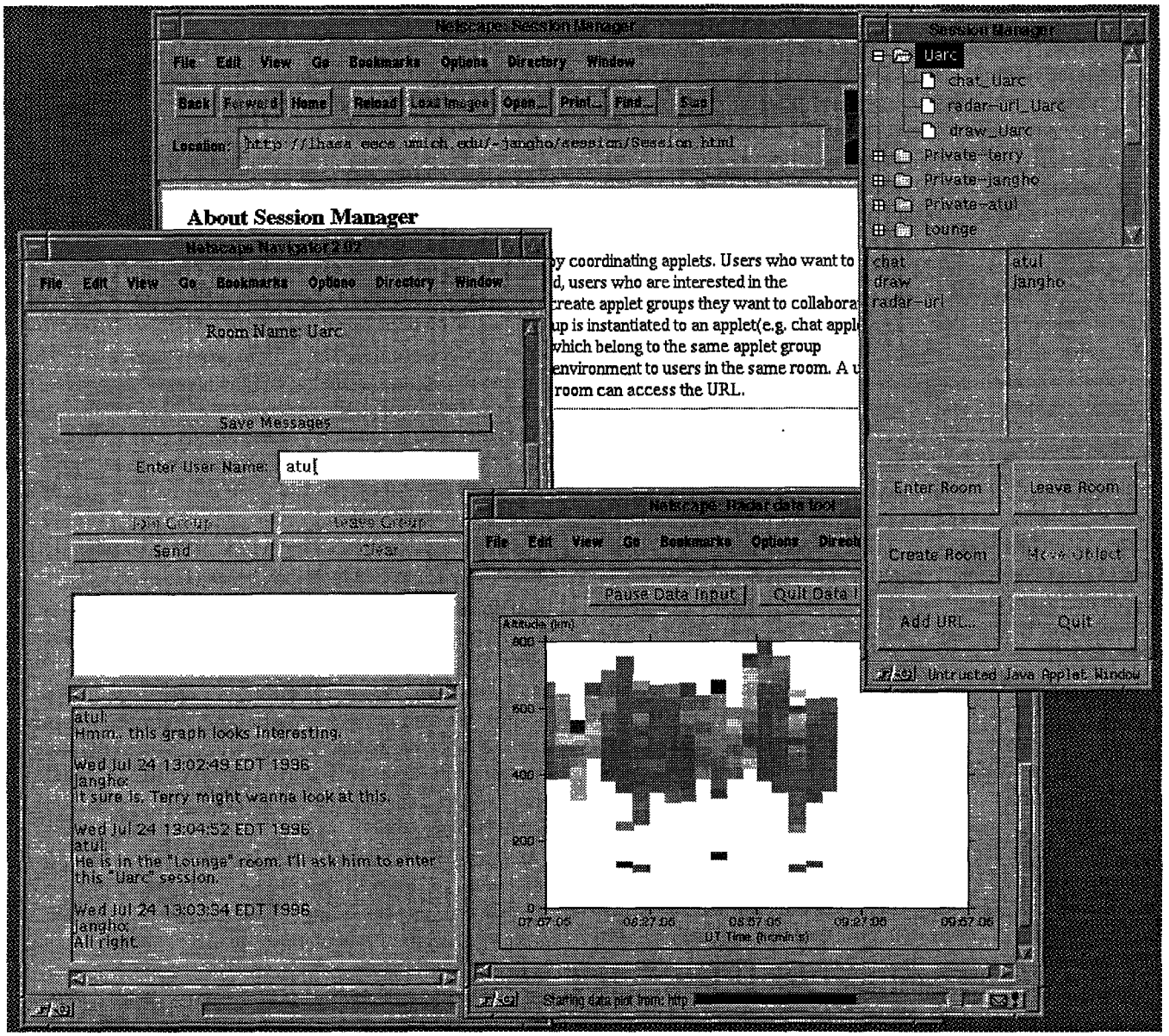

Figure 1: The use of a multi-applet shared workspace. The upper window of the session manager shows a list of five rooms : two public rooms(Uarc and Lounge) and three private rooms(Private-terry, Private-jangho and Private-atul). A private room is automatically created when a user logs in the system. The middle-right window of the session manager shows a list of users (atul and jangho) who are currently in the selected room, Uarc. In the Uarc room, atul and jangho are currently collaborating with each other on the Radar data picture via the chat applet.

The applets in our model can be implemented as either Java applets or Java applications, for example.

Applet-Groups

An *applet-group* consists of a set of *applets* of the same type that provide a shared workspace to users in the room. The whiteboard applets on each user's desktop constitute an applet-group because they keep their displays or data synchronized as they are interacted with by the users. In our implementation, each applet-group member joins the same communication multicast group, similar to those provided by ISIS [1] and Corona [7], for state coordination purposes.

URLs

URLs can be placed in a room and can be accessed by users in the room. This enables users to share references to objects on the Web, like files and images, with other users.

Rooms

A room may contain *URLs* (Uniform Resouce Locators) and *applet-groups*. A room provides shared data objects and tools to users around which they can establish collaborative sessions. A room also provides collaboration awareness features to the users who enter the room, so that they are aware of applets/URLs and other users present in the room.

We use the notion of room as simply a high-level grouping mechanism for identifying applets and objects in a shared workspace. We do not imply the use of a particular user-interface metaphor, such as that described in [8]. Also, the applets within the room can be placed anywhere on the user's desktop. Another alternative would have been to present rooms as URLs in an HTML page. In principle, we leave it up to the user-interface designers of a particular collaboratory to decide how the notion of rooms is presented to end-users.

A shared workspace can consist of multiple rooms. The rooms are virtual in the sense that users can enter multiple rooms simultaneously. Room services may require users to be authenticated before they can enter the room.

Figure 2 shows an example of a room. In this example, two users A and B are collaborating in a room. There are two applet-groups – whiteboard and chat – that are being used in the shared workspace. If now a user T enters the room, the two applet-groups in the room are made available to T; the applets corresponding to the two applet-groups get created on T's desktop and they join their respective applet-groups, in order to synchronously share tools and data within the group. If there were no room facility, the user T would have had to do all the work of establishing a shared workspace manually.

Below, we present semantics of rooms and their relation to users, applet-groups, and other rooms.

Applet-Room Relationship

We allow any number of applet-groups to belong to a room. However, an applet-group can belong to exactly one room. This is not a limiting constraint, because we allow users to be

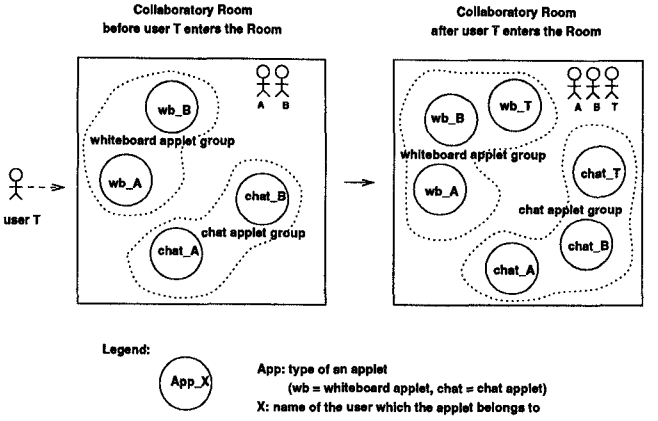

Figure 2: A collaboratory room

members of multiple rooms simultaneously. Thus, if users in different rooms wish to view the same applet, they could do so by moving the applet to a new common room or entering a room that provides the applet.

The following operations may be performed on rooms to change the binding between applets and rooms:

- *Put an applet in a room*: If a user creates or put an applet in a room, a new applet-group is created and the applet joins that applet-group implicitly. This applet-group becomes available for users in the room to join.
- *Create an applet corresponding to an applet-group in the room:* This operation creates an applet in a user's workspace of the same type as the applets in the applet-group. This applet joins the applet-group, normally leading to a synchronization of state and displays associated with the applets in the applet-group.
- *Delete an applet:* This operation destroys an applet in the room from the user's workspace. Applets of other users may continue to exist, even if they were part of the same applet-group as that of the deleted applet.
- *Delete an applet-group:* This operation destroys the communication group to which members of an applet-group belonged. The applets in the applet-group receive a notification that they are no longer part of the group, and will typically be designed to destroy themselves or become private applets. For instance, if the whiteboard applet-group is deleted from a room, all the corresponding whiteboard applets either will be destroyed or will become private applets of their users.
- *Move an applet from one room to another:* As a result of this operation, the applet will leave the applet-group it had belonged to. Then, it will be put in the destination room. As a consequence, a new applet-group will be created in the destination room. Applets can be made private by moving them to a private room.

- *Move an applet-group from one room to another:* This is equivalent to moving an applet belonging to the applet-group to the destination room and destroying the applet-group in the source room.

Room-Room Relationship

Nesting of rooms within one another is not currently allowed. To facilitate navigation from rooms to related rooms, *links* to other rooms can be easily implemented using the existing model. Links can be provided by a *link applet* that maintains a collection of references to other rooms. At the present time, users of the collaboratory have not indicated a need for nesting of rooms. We may consider such support in the future, if a need arises. In such a case, rooms most likely will be modeled after directories in file systems such as in the Andrew File System.

User-Room Relationship

A user can be inside any number of rooms simultaneously. We see no reason at present for the underlying architecture to restrict users to a single room, and indeed see several potential benefits for users such as being able to better utilize their time, transfer results of collaboration from one room to another, etc.

Each user is implicitly assigned a *private room* which only that user may enter. A private room serves as a placeholder for applets that are not currently shared with other users.

When a user successfully enters a room, the applet-groups within the room become accessible to the user's session manager. For each such applet-group, a session manager will typically instantiate an applet of the appropriate type and send a request to the applet to join the applet-group, thus making available the shared context of the room to the user.

When a user leaves a room, user's applets that are part of the room implicitly leave their applet-groups.

To control the access of users to a room, users are given roles with respect to a room, which is described in the following section in detail.

Persistence of Applets, Applet-groups, and Rooms

Our experience with the work in collaboratories indicates that support for asynchronous collaboration can be essential because not all participants may be available at the same time due to scheduling conflicts. We therefore provide features for persistence of state of a collaboration across synchronous collaboration sessions.

In our current system, rooms continue to persist even if there are no users within the room. Thus rooms can be used as depositories of information that is being shared among users. In the future implementations, we plan to allow users to declare a timeout value for a room so that room is automatically deleted after a timeout if it is not being actively used.

To support asynchronous collaboration, applets also need to be designed so that their state can be saved across sessions. We use DistView's (Java-based version) mechanisms to save the applet identifier, its applet-group name, and its state with the Room Manager, upon request of a user. DistView requires applets to provide access to their state anyway for replication purposes when new applets are added to an applet-group. For persistence, the same state is saved with the room manager.

If a user enters a room with no other users later, the applet-group, along with the last saved state, is still available. The user can instantiate applets that join the saved applet-group, initializing themselves to the last saved state.

User Roles in a Room

For the purpose of room access control, a user may assume one of the following roles:

Administrators who are allowed to change the access control on a room, create/delete applet-groups from the room, and interact with the applets in the room.

Members who have all the rights of administrators, except that they cannot modify access controls on the room.

Observers who are only allowed to observe the state of group-aware applets in a room but not allowed to affect the state of these applets in anyway.

Restricted who are not allowed to enter the room.

Each room is associated with its own Access Control List (ACL). There are five levels of privilege in the ACL corresponding to each of the roles. Table 1 shows the operations allowed for each privilege level.

A model with finer levels of access control, say, along the line of Andrew File System, could have been used, but the above model appears adequate for the time being and is simple to understand and enforce.

Rooms are created by users (in our implementation, by the session manager, on their behalf). The creator of the room becomes a permanent administrator of the room (to ensure that a room always has at least one administrator).

DESIGN AND IMPLEMENTATION

Our architecture uses a *session manager* to coordinate the various applets on a user's desktop, as shown in Figure 3. One session manager exists per user. The session manager presents a common interface to the various applets and provides facilities for querying about the status of the collaboratory. The session manager and the applets may communicate with each other; the session manager can send commands to the applets, such as requesting them to quit, and provide them information on collaboratory status for group-awareness purposes.

Users can access a collaboratory's home page on the World-Wide Web and download a session manager to a platform with a Java-enabled Web browser (e.g., Netscape). The session manager downloads group-aware Java applets (e.g., chat applet, draw applet) using the Web browser to establish a shared workspace for users.

Operations	Privileges			
	ADMIN	MEMBER	OBSERVER	RESTRICTED
Enter a room	Y	Y	Y	N
Look up a room	Y	Y	Y	N
Leave a room	Y	Y	Y	N
Destroy a room	Y	N	N	N
Join applet-group	Y	Y	Y	N
Add applet	Y	Y	N	N
Move applet-group	Y	Y	N	N
Delete local applet	Y	Y	Y	N
Delete applet-group	Y	Y	N	N
Get room ACLs	Y	Y	Y	N
Set room ACLs	Y	N	N	N

Table 1: Operations allowed for each privilege level

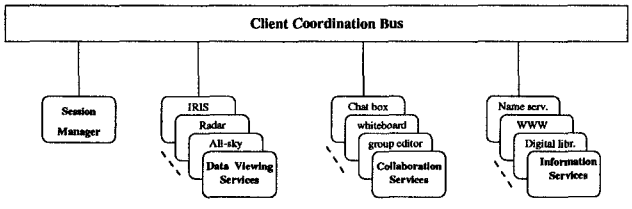

Figure 3: Client Architecture – clients consist of applets and a session manager

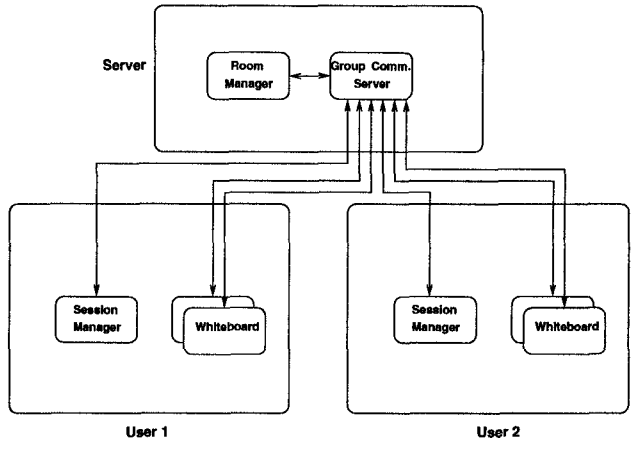

Figure 4: Run-time communication architecture in a collaboratory

Run-Time Communication Architecture

Figure 4 shows the run-time communication architecture of a collaboratory. Each user's workspace consists of one or more applets (small, individual applications, such as whiteboard, chatbox, and scientific data viewers). Corresponding applets (e.g., whiteboard applets of different users) communicate with each other through group communication server, as users perform actions in order to maintain a common interface to the group.

The room manager and the group group communication are implemented as Java applications and run on a server machine. The session manager is implemented as a Java applet that runs on each user's machine. Through the HTTP server, `httpd`, also running on the same server machine, users can access a Collaboratory's home page on the World-Wide Web and download the session manager to their desktop with a Java-enabled Web browser (e.g., Netscape).

Figure 5 shows the actions that occur when a group-aware applet (e.g., chat applet) needs to be started on a user's desktop. The numbers near the arrows show the sequencing of messages. First, the session manager gives a URL to the browser to request the downloading of the applet from the HTTP server. The URL contains the name of the name of the Java-applet class, the unique id to be assigned to the Java-applet, the unique id assigned to the session manager, and the address with port number of the server where the group communication is running. Next, the `httpd` server starts a CGI script on the server which, in turn, produces a virtual

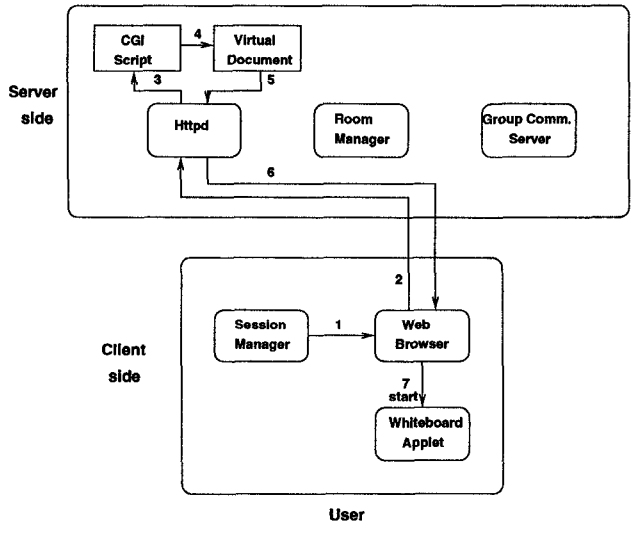

Figure 5: Starting an applet in a collaboratory

HTML document (a document created on the fly) that contains HTML tags to start the applet and parameter tags, to pass the applet id, session manager id, and the group communication server's port, to the applet. Then, This HTML document is returned to the browser.

Upon receiving the HTML document containing the reference to the applet, the browser displays the HTML document, causing the applet to be downloaded from the HTTP server, if it is not available locally. The applet sends a message to the session manager on the user's machine (via the group communication server because of the current security restrictions on Java applets), gets the the applet-group id to join from the session manager, and joins the applet group, thereby establishing a shared workspace with other applets belonging to that group.

An Example of Run-Time Communication

Figure 6 illustrates the message communication among an applet, a session manager and the room manager, when a user enters a room.

When a user requests the session manager to *enter a room*, the following sequence of events occur.

1. The session manager performs the operation of *enter a room* which sends a message to the room manager.
2. The room manager checks the access control of the room in order to verify the privilege level of the user associated with the room. If the user is allowed to enter the room, the room manager updates the room structure to put the user inside the room and sends the result back to the session manager.
3. When the user gets inside the room, the session manager performs the operation of *looking up a room* and gets the available applet groups in the room from the room manager.
4. For each applet group the user has selected, the session manager instantiates an applet which belongs to the applet-group.
5. Each applet created by the session manager according to the user's input sends a message to the session manager that says it has started.
6. Upon receiving this message, the session manager tells the applet which applet-group the applet should join.
7. When the applet receives this message, the applet calls the *joinGroup* primitive [7] to join the applet group and then sends the group id back to the session manager to let it know that it has successfully joined the applet-group.
8. Upon receiving this message from the applet, the session manager performs the operation of *add applet* to tell the room manager to add the applet entry in the applet-group in its room data structure.

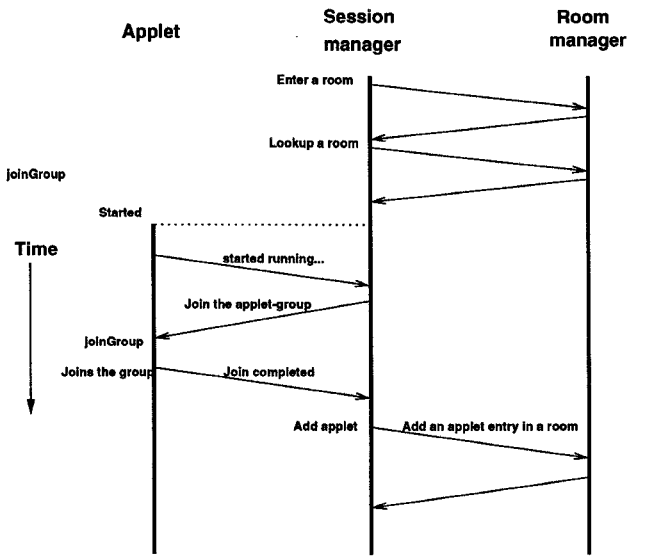

Figure 6: Interprocess communication among applet, session manager, and room manager when a user enters a room

9. The room manager checks the privilege level of the user with respect to the room. If the user is allowed to create an applet in the room, the room manager updates its room data structure and multicasts the changes to the room view to all the session managers. Upon receiving these messages, the session managers update their room views.

The mechanism of *leave a room* is straightforward. Basically, it involves removing the applet entry from its applet group in the room structure and making the applet leave its applet group.

A preliminary prototype of the model has been implemented in Java. Primitives except for access control have been implemented and the implementation of access control is currently in progress.

Enforcing Access Control

We next show the extensions to the above protocol of entering a room in order to enforce access control in an unsecure environment. In this protocol, we assume that the room manager is a principal trusted by all users. The protocol is designed for a public key cryptosystem [6], so we assume that all users can securely obtain each other's and the room manager's public keys and that they and the room manager possess trusted cryptographic software. All principals keep their private keys secret.

The secure *enter a room* protocol is a variation of the Needham-Schroeder authentication protocol [10]. This protocol uses a trusted authority (in this case, the room manager) to assign keys for a pair of users to interact. We modify the protocol to enable a group of users with different access rights to interact. Because of the variation in privileges, asymmetric secrets, such as private/public key pair, are used to au-

thenticate a principal. Symmetric keys can also be used (with greater performance improvement), but the distribution protocol is more complex [9].

In step 1, the session manager sends a message to the room manager to *enter a room*. This message contains the collaborator's name, the room name, and a random number. The user's session manager creates a digital signature (private key transformation) of this message using his private key and encrypts it using the room manager's public key. Therefore, only the room manager can read the message.

In step 2, the room manager decrypts the message and verifies the signature of the message and checks the access control list of the room. If the signature is verified and the user is permitted to enter the room, then the room manager determines a short-term private/public key pair for the user in the room (called the *private room key* and the *public user-room key*, respectively). The room manager creates a return message containing the room id, the key pair, and the user's random number. The room manager adds its digital signature of that message and encrypts it with the user's public key, so that only the user's session maanger can obtain the key pairs. Only the room manager could have decrypted the step 1 message to obtain the random number, so the user's session manager can safely assume that it is interacting with the room manager.

In a step 3, the user's session manager sends a message to request the room status by invoking the operation to *lookup* a room. The session manager's message simply contains the user name and the room id. The session manager sends the message and its digital signature created using his new private room key to the room manager.

In step 4, the room manager verifies the signature. If the signature is verified, then the room manager can add the user to the room. The room structure is modified to add an entry containing the user's name, role, and the public user-room key. Other users obtain this public key from the room manager (in a message signed by the room manager) when they want to verify a signed message from that user. By using the room manager to store the public keys, the well-known replay attack against the Needham-Schroeder protocol is avoided [4].

MEETING THE NEEDS OF A COLLABORATORY

Below, we discuss several ways in which the model described can support different needs of collaboratories:

- *User-extensibility:* The architecture allows users to add their own collaborative applets as long as they provide an interface required by the session manager. In our implementation, an abstract class is provided that specifies the methods that each applet should support. Examples of methods include request by the session manager to the applet to join a particular applet-group, to leave a group, to iconify itself, to kill itself, etc.

- *Group-awareness:* The session manager currently manages information about users and objects in a room. It also provides an API to applets to allow them to query about the state of the collaboratory, such as information about the rooms they belong to, users who are in a room, etc. Applets can use that information to provide additional context to users.

- *Link applets:* To allow users to move from one room to another in an hypertext (or Web-like) manner, as discussed earlier, a link applet that maintain a list of related rooms can be used.

- *Common utility applets:* Such applets include chat, whiteboard, mail, and audio/video conferencing tools. In our implementation at present, we provide chat, whiteboard, mail (via the browser), and some UARC-specific applets.

- *Support for asynchronous collaboration:* Persistent rooms that contain persistent applet-groups can be used to support asynchronous collaboration. In our current prototype, rooms and applets such as multi-user chat and whiteboard support persistence.

- *More open, scalable collaboratories:* The access control model encourages more open collaboratories. It allows scientists to be receptive to the idea of making the collaboratory openly accessible over the Internet since they can work in rooms that can be made reasonably secure and support non-obtrusive observers.

There are certainly other challenging issues that need to be addressed in supporting the model described in this paper for large-scale collaboratories to be practical. For instance, scalable communication protocols are needed so that large number of observers can be supported, and performance perceived by members of a room can be given higher priority than the performance perceived by observers in a room. The work on the Corona system [7] aims to address that problem.

RELATED WORK

Some important examples of collaboration systems include Suite [5], DistView [12], Rendezvous [11] and MMConf [2]. They provide generic facilities for establishing a shared workspace and implementing various floor control policies. In these systems, facilities for conference management are usually tightly integrated with the application. Multi-applet workspaces, like CBE, require that the session management service be outside any individual applet and be usable by multiple applets simultaneously.

Another use of the room concept has been described in [8] with the goal of dividing the user's desktop into a suite of virtual workspaces, similar to the rooms in the real-world. In [8], rooms are primarily meant to facilitate management of one's workspace. In our case, rooms are used as a means for providing a shared context to a group of users.

Multi-User Dungeons (MUDs) provide a notion of a place for collaboration [3]. Traditional MUDs are text-based systems in which users use text-based commands to modify objects in

a room. Recent projects, such as the Jupiter project at Xerox Parc have augmented MUDs to add support for collaboration using video and audio.

The wOrlds [16] system and the TeamRooms [15] system organize applets into "locales" and "team rooms" respectively, which are similar to rooms in this paper. Both wOrlds and TeamRooms, like CBE, support shared workspaces with multiple applets, users, and data objects. One difference in terms of goals is that our system is intended to support Internet-based collaboratories. Thus, we focussed in more detail on specifying constraints on relationships between users, applets, and rooms, as well as on mechanisms for access control so that collaboratories can be made more widely accessible over the Internet. In terms of implementation, The wOrlds system currently uses a distributed version of Smalltalk and TeamRooms uses Tcl/Tk. We decided to use Java, so that clients can access a collaboration environment over the Web from multiple platforms.

In TeamRooms [15] (which uses GroupKit [14]), all applets in a room are displayed in one larger window, which effectively then needs to provide its own window management functions. Thus, TeamRooms also provides a GUI metaphor for representing rooms. In CBE, the display of applets in a room is a decision that is left up to the session manager. The session manager in our prototype does not restrict the applets in a room to a single window. This has the advantage that users can use their standard, familiar window manager to control the placement of applets on their desktop.

A potential advantage of displaying all the applets in a single window, as in TeamRooms, is to allow a user better control on the display of rooms on the desktop, if the user is active in a large number of rooms simultaneously. However, in our experience, most users actively use only one or two rooms at a time, with each room containing a large number of applets, though a larger number of rooms may be available to them. We therefore decided that it was better to allow users to use a familiar window manager to control placement of applets, rather than providing structuring mechanisms to control display of a large number of rooms. If better control over applets in an individual room is desired, the session manager can be used to send control commands (such as quit, iconify, etc.) to all the applets in a room. A session manager that displays all the applets corresponding to a room in a single window could of course be implemented in CBE, since the session manager design is a policy decision.

Habanero at NCSA, like CBE, uses Java for building group-aware applications. It provides support for "sessions" that can contain URLs and group-aware applications. A Habanero session has similarities to a room, but does not allow applications to be dynamically added during a session. Furthermore, CBE allows active applets to be dynamically moved between private and shared workspaces and provides server support for saving the state of a room across sessions to facilitate asynchronous collaboration.

CONCLUSIONS

In this paper, we described an architecture for supporting collaboratories and shared workspaces. The architecture is based on the notion of rooms, users, applets, and applet-groups. We proposed semantics of these concepts, described their inter-relationships, and presented a simple model of access control to rooms.

We also investigated the issue of mapping these higher-level concepts on lower levels of the infrastructure. In particular, we showed room-level grouping mechanism which is mapped to operations on communication-level groups. We also presented a solution outline for enforcing access control on rooms by utilizing digital signatures in group communication protocols.

We showed how these concepts can be used to support a variety of needs in collaboratories using special applets designed to support a variety of tasks such as navigation around rooms, session recording, provide group-awareness, etc.

Prototype versions of the model with several applets, such as multi-user chat, shared whiteboard, and UARC-specific displays have been implemented in Java and can be downloaded to run with a Java-enabled browser. Ongoing work includes building a critical-mass of applets for the UARC collaboratory, refining the user-interface provided by the session manager, and providing scalable, fault-tolerant versions of the room manager.

ACKNOWLEDGEMENTS

We thank Daniel Atkins, Michael Burek, Robert Clauer, Robert W. Hall, Farnam Jahanian, Amit Mathur, Gary Olson, Hyong Sop Shim, Terry Weymouth and other members of the UARC team for their constructive comments and suggestions. We also thank the reviewers for their invaluable feedback. Support for this work has been provided by the National Science Foundation through the cooperative agreement IRI-9216848.

REFERENCES

1 K. Birman. The process group approach to reliable distributed computing. *Communication of ACM*, 36(12):37–53, December 1993.

2 T. Crowley, P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson. MMConf: An infrastructure for building shared multimedia applications. In *Proceedings of the Third Conference on Computer Supported Cooperative Work*, pages 329–342, Los Angeles, California, October 1990.

3 P. Curtis and D. Nichols. MUDs grow up: social virtual reality in the real world. In *Proceedings of the Third International Conference on Cyberspace*, 1993.

4 D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

5 P. Dewan and R. Choudhary. A flexible and high-level framework for implementing multi-user user interfaces. *ACM Transactions on Information Systems*, 10(4):345–380, October 1992.

6 W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

7 R. W. Hall, A. Mathur, F. Jahanian, and A.Prakash an C. Rasmussen. Corona: A communication service for scalable, reliable group collaboration systems. In *ACM Conference on Computer-Supported Cooperative Work*. ACM Press, 1996.

8 D.A. Henderson and S.K. Card. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. Graph*, 5(3):211–243, July 1986.

9 T. Jaeger, A. Rubin, and A. Prakash. Building systems that flexibly control downloaded executable content. In *Proceedings of the 6th USENIX Security Symposium*, pages 71–88, July 1996.

10 R. Needham and M. Schroeder. Using encryption for authentication in large networks. *Communications of ACM*, 21(12):993–999, December 1978.

11 J.F. Patterson, R.D. Hill, S.L. Rohall, and W.S. Meeks. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of the Third Conference on Computer Supported Cooperative Work*, pages 317–328, Los Angeles, California, October 1990.

12 A. Prakash and H. Shim. DistView:support for building efficient collaborative applications using replicated objects. In *Proceedings of the Fifth Conference on Computer Supported Cooperative Work*, Toronto, Canada, October 1994.

13 R. Clauer et. al. UARC: A prototype upper atmospheric research collaboratory. *EOS Transactions on American Geophysical Union*, 74, 1993.

14 M. Roseman and S. Greenberg. GroupKit: A groupware toolkit for building real-time conferencing applications. In *Proceedings of the Fourth Conference on Computer Supported Cooperative Work*, pages 43–50, Toronto, Canada, October 1992.

15 M. Roseman and S. Greenberg. TeamRooms: Network places for collaboration. In *ACM Conference on Computer-Supported Cooperative Work*. ACM Press, 1996.

16 W. Tolone, S. Kaplan, and G. Fitzpatrick. Specifying dynamic support for collaborative work within wOrlds. In *Proceedings of the 1995 Conference on Organizaional Computing Systems*, pages 55–65, August 1995.