Malleable Shared Workspaces to Support Multiple Usage Paradigms

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

ABSTRACT

Several recent systems provide a room-based metaphor to represent shared workspaces that require use of multiple collaborative tools. These systems provide users with a fairly static usage paradigm of room-centered collaboration, requiring users to mold their collaborative activities to the paradigm rather than molding the paradigm to fit the requirements of their collaborative activities. In this paper, we propose a powerful and yet simple event-action based model, augmented with access control and multi-user features, for room-based systems for providing a high degree of malleability so that these systems can be adapted to provide support for a variety of collaborative facilites, such as call centers, mailboxes, shared repositories, and role-based collaboration, including facilities that are not necessarily anticipated by system designers. The model can be used by both system developers as well as by end-users to customize a system to meet the requirements of their group tasks.

Keywords

CSCW toolkits, shared workspaces, room-based systems, reflective systems, flexible groupware, groupware tailorability.

INTRODUCTION

Shared workspaces are a key element of collaborative systems. Several systems, including MUDs [3], Worlds [13, 5, 7], CBE [8], and TeamRooms [10], use the *room* metaphor to represent shared workspaces, especially when participants use multiple tools for collaboration in a shared workspace.

Existing room-based systems provide rooms with fairly standard behavior. In fact, the room-based metaphor on these systems evolved from the observation that real-world collaboration often takes place in physical rooms around shared artifacts; thus it is logical to emulate the behavior of physical rooms in virtual computer-based rooms. Systems vary in the extent to which physical rooms are emulated; MUDs are often text-based, TeamRooms provides a shared window manager in which shared objects can be placed, and CBE provides access-controlled rooms in which users share objects but not the interface.

A key question is whether the room-based collaboration paradigm, as exemplified by these systems, adequately supports the needs of different types of groups and collaborative situations? Can the metaphor support collaboration activities that do not take ordinarily place in physical rooms? If not, what support is needed so that the paradigm can be adopted for a wide variety of uses, some perhaps not even anticipated by system designers? In this paper, we explore answers to these questions.

We show that there are many examples of collaborative situations where the standard room-based metaphor provides inadequate support. On the other hand, we show that, with suitable extensions, the paradigm can be made very powerful and can be used to support a range of collaborative activities, such as standard room-based collaboration, call centers, shared repositories, paper reviewing, voting, email, etc.

The key goal is to provide a high degree of malleability to the room-based metaphor so that it fits the requirements of the application. The malleability can range from simple (e.g., access control) to behavioral (e.g., establish a private session with an available agent in a call center room when a customer enters the call center).

The importance of flexibility and tailorability in CSCW systems has been discussed by several researchers. Bentley and Dourish suggest an approach that views a CSCW system as a medium through which collaborative work occurs, rather than an embodiment of mechanisms representing perceived regularities in collaborative activity [1]. Roseman and Greenberg argue that personalizable groupware must adapt to a wide variety of different group behaviors, including behavior not originally expected by the groupware developer [9]. Our work is focused on considering this problem of tailorability of the environment to fit the characteristics of the work is in the context of room-based systems. We propose concrete design recommendations to achieve *malleability* (a high degree of tailorability), while maintaining simplicity of use, in these systems.

The paper is organized as follows. First, we show various types of shared workspaces. Then, we discuss our model of software architecture to provide flexible shared workspace. Next, we describe an access control mechanism of our system. Then, we discuss how the model can be used to support malleability of workspaces to fit various usage scenarios. the

system follows. Finally, we present conclusions and future work.

TYPES OF SHARED WORKSPACES

What are the various ways in which room-like shared workspaces can be potentially used, but do not conform to the standard room metaphor? We consider the following examples of infrastructures required for various collaborative tasks: *call centers, paper reviewing facility, support for email to a group of users who act as one entity, support for setting up invitations to a group of participants, shared repositories among a group of participants, room where views of the room are synchronized among users* and *transient rooms that automatically disappear after a certain time or when no users are active*.

We briefly describe the characteristics of each of the above. In subsequent sections, we show how these can be supported in a room-based system, after suitable, fairly general, extensions.

Call Center

In a call center, customers wait for service until an agent becomes available. For example, customers may go to an insurance company's web page and, if they want to talk to an agent, click on a "call center" object. When an agent becomes available, a private collaborative "room" is established with the client and the agent as the members, and the customer and the agent can use various collaboration tools to conduct their business in this private room.

Support for this can of course be provided by creating a special type of room called a *call center*. Call center rooms behave differently from standard rooms, where any number of participants simply enter and share artifacts in the room. For a customer, entering a call center room implies waiting until an agent becomes available and then entering a private collaboration with the agent. Unfortunately, that solution of simply extending the infrastructure to add an additional room type is unsatisfactory. It is difficult to anticipate all the potential variations in which call centers may be used by users. For instance, one variation could be that customers are asked to leave their contact information after a certain timeout and agents attempt to contact them later.

Our goal is to provide general facilities that can be used to create customized call centers, even by end-users, and certainly by system administrators.

Group Mail Room

This is a type of user-specific room where the user can check for mail from others. Other users are allowed to post mail to this room.

Only the recipient of the mail should be able to read the mail or delete the mail.

Paper Reviewing

A customized room is needed to support role-based collaborative tasks, such as paper reviewing. Authors of papers can submit papers to this room, but not delete or look at other authors' papers. A reviewer in room should be able to read a submitted paper and comment on them unless the reviewer herself is an author of the paper. To support this type of room, the grouping of users into role-based categories for the purpose of access control is necessary. Also, access rights can be object-specific; a reviewer cannot necessarily review all papers in the room. Authors can only add their own paper in the room, and not list contents of the room (except perhaps their own paper) or delete papers submitted by others.

Room for Invitation

A user can invite people to a room. In case the invited people are in our system, this involves first notifying them. If the invited people are not in the system, external notification can be used, such as by sending email to them. This requires the functionalities for checking the presence of the user in the system as well as for notifying users. Secondly, the access privilege for the invited people should be set accordingly. It is likely that the privileges of these invited people are not permanent. The access rights of these people expire after a certain period of time.

Repository Room

This is a room type that allows people to put objects for use by other users. Users are allowed to add and operate on their own objects but are not allowed to modify others' objects without proper permissions. This requires both room-level and object-level access control.

Synchronized Room

This is a room type that allows people to see synchronized action. For example, a slide object in this synchronized room can provide a synchronized slide show to people inside the room.

Transient Room

This type of room expires after a certain period. This room type requires the attribute associated with a room, which determines life of a room. The time-out after the period, a notification to users should occur as well as deletion of a room. The life of an object in a room is subject to the lifetime of a room.

ADAPTABLE WORKSPACE MODEL

We now present a high-level model that provides facilities to programmers and end-users to do customization of behavior of rooms and objects within rooms so that the metaphor fits the variety of needs described in the previous section. The model proposed in this paper is an extension of the model called Collaboratory Builder's Environment (CBE[1]) presented in [8].

[1]CBE is a software architecture or toolkit for creating extensible collaborative shared workspace

The key elements of the model are entities that include *users*, rooms, *objects in the room*. Users, rooms, and room objects can have a dynamic set of attributes. These entities can be associated with $< event, action >$ pairs. When an event occurs on the entity (e.g., clicking on the object), an associated action is executed. Actions are typically small fragments of code that can read or update attributes of the entities, and can be written in an appropriate scripting language[2]. A role-based access control model provides the ability to constrain operations on entities, or the ability to execute actions on objects, to users in appropriate roles.

We next describe each of these simple elements in more detail. Then we give examples of how these elements provide a powerful mechanism to create highly tailorable rooms, including supporting many of the features of the powerful access control model described in Edwards [4].

Entities

Following are the key types of entities in our shared workspace model: users, applet, group-aware object, non-group-aware object, URL and room.

Users

Users perform operations on the shared workspace, such as entering rooms, deleting rooms, generating events on objects (e.g., clicking on an object), etc. For every user in the system, the system maintains certain attributes, such as their unique system id and *currentRoles*, the current set of *roles* the user is in for the purpose of access control. In addition, other attributes can be added to facilitate collaborative activities such as user name, organization, phone number, etc.

Rooms

A room can contain objects that are shared among users, subject to access control and object-specific attributes. It also provides users with facilities for collaboration awareness – system-defined attributes of a room include the set of users who are present in a room. Room also have the following system-defined attributes: *room name*, *owner*, *access permissions*, *the list of objects in the room*, *the list of users in the room*, and *the creation date*. In addition, users can define additional attributes for a given room, subject to access control rules, for modifying the room behavior, such as *termination time* for implementing transient rooms, etc.

A user can be inside multiple rooms at the same time, as far as the system is concerned. Since the rooms are virtual, unlike physical rooms, no such restriction is necessary. In fact, in the use of CBE in the Upper Atmospheric Research Collaboratory, space scientists often use this facility to monitor and participate in several collaborative activities simultaneously. This behavior can, however, be changed by tailoring the rooms using the facilities described in this section.

[2]An obvious question is whether scripting languages are an appropriate interface for end-users. We will discuss some ways of simplifying the interface for customization to end-users in a later section.

Our current implementation of CBE is restricted to a flat organization of rooms. The model, however is intended to be hierarchical. Rooms can contain other rooms or references to other rooms. However, a room cannot be contained in two rooms simultaneously; it could be contained in other rooms as a *reference* (similar to symbolic links in Unix file systems).

Room Objects

Rooms contain named objects. Standard objects known to the system include URLs, data objects, applet objects, and group-aware tools. These objects can be added or deleted dynamically from a room. They can also be copied or moved from one room to another. Common system-defined attributes of room objects include *object owner* (initially set to the object creator), *creation date*, *access control list*, *current room*, and *object type*.

A *URL object* provides the access to URL(Uniform Resource Locator) on the Web. Users can put URLs in a room and make those URLs accessible by other users in the room for opening via their web browsers. Users can use this URLs to share references to objects on the Web. A URL object has the following additional attribute: *the URL value*, which is passed to the browser for display. URLs, in general, can also refer to executable content, such as applets in Java, or plugins.

Data objects are simply a named sequence of bytes that can be read or written to a room. They allow a room to be used as a repository of shared data, just like a file system. Users, for example, can upload files to a room or download files from a room, subject to access control restrictions. Besides the standard attributes such as owner and access permissions, they have the associated attributes *dataType* and *appletName*. The *appletName* is the name of the application that should be started when an attempt is made to display the object. This application is passed a reference to this object. For instance, users could use a shared whiteboard in the room to collaborate with each other, and then save its state to a data object in various formats (e.g, GIF, whiteboard-specific format, HTML). The format name should be stored in the *dataType* attribute. The name of the application (e.g., whiteboard), to be used by default to display the object, should be stored in the *appletName* attribute.

Group-aware objects are a special kind of data objects that come up in shared mode. In that case, the application that is used to display them is passed an additional parameter indicating that the object's state should be shared during editing or modifications. Tools such as Chat and shared whiteboard are implemented as group-aware objects in CBE.

Parameters (e.g., attribute values such as username) can be passed when invoking URLs and group-aware objects, so that appropriate context can be passed from rooms to the contained objects.

Opening a group-aware object causes the invocation of a group-aware applet by downloading the associated group-

aware applet and then initialization of its state with the content of the group-aware object or the current active state of the applet-group [8] if the object been modified by one of the users using the group-aware applet.

An *applet-group* [8] consists of a set of *applets* that are sharing a particular group-aware object. A set of whiteboard applets that are collaboratively used to modify a shared object belong to the same applet-group. From the implementation point of view, applets in the same applet-group are members of the same multicast group. ISIS [2] and Corona [6] take this approach for state coordination purposes.

Object Attributes

As indicated earlier, entities in the system have system-defined as well as user-defined attributes. User-defined attributes, coupled with $< even, action >$ rules, provide the means to change the paradigm behavior to suit the needs of the application.

At present, we consider the attribute values of integers, floats, strings, and lists to be adequate for most tailorability tasks.

Events and Actions

Rooms and room objects can be associated with $< event, action >$ pairs. Events are typically generated in response to a user action (e.g., selecting a menu item after selecting an object or clicking on the object). When an event occurs on the selected entity, an action bound to the event is executed. Events can also be generated from timers, or by actions.

Actions provide the core mechanism to alter the behavior of rooms. Actions can read attribute values, update them, modify access control lists, set named timers attached to specific objects, etc., and generate new events, subject to access-control restrictions.

Actions are typically expected to be small fragments of code that can read or update attributes of the entities, and can be written in an appropriate scripting language, such as a subset of Tcl or JavaScript[3] Often, we expect that end-users will not have to do any programming. They will simply copy existing rooms (e.g., transient rooms) and change their attributes (e.g., terminationTime) to customize their behavior. However, the model permits more radical changes of behavior using scripting.

Actions have an attribute *creator* and an access control list of users who are allowed to execute the action. Actions can check the identity of the user who initiated the action, in order to, say, provide different functionality to different users.

Figure 1 shows an example of the use of the events and actions in the model for implementing part of the functionality of a call center room.

[3]In our current system, actions are coded in Java, and not available to end-users via a scripting language. We consider that to be temporary limitation of our present implementation, rather than a fundamental conceptual limitation of the model.

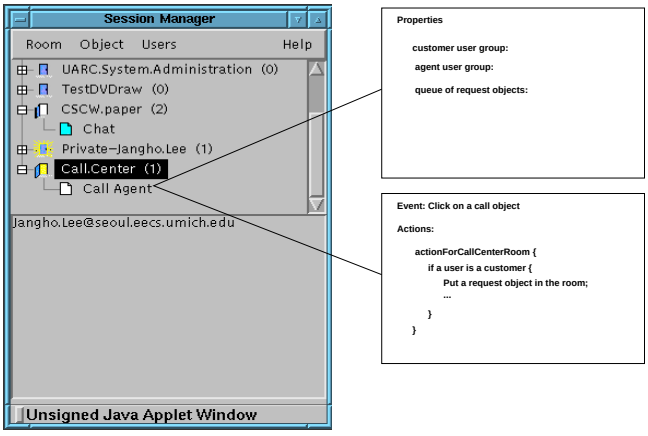

Figure 1: Property, event and action paradigm for the type of call center room

Basic functionalities that a call center should provide include the following. First, an *enter a room* operation allows a customer to enter a call center. When a customer clicks on a call object in a room, a Chat applet is brought up to provide a communication channel between the customer and an agent. To shorten the procedure, a room can be designed so that as soon as the customer walks into the call center a call object is automatically activated. Then, the agent and the customer can have a chat inside the call center room or they can chat in the customer's private room. In the case of the latter, a transient room can be created with *create a room* operation. The *put user* operation will put the customer and the agent in the room. The *add applet* operation will put a chat applet in the room. Then, *invoke applet* will bring up chat applets for them. As described above, the event of walking into a room by a customer triggers the corresponding actions (or operations).

The following shows the outline of of the $< event, action >$ pair for a call center room. Other examples can be found in the later section on "Using the Model to Provide Adaptability".

Event: Click on a call object
Action:

```
public actionForCallCenterRoom {
   if a user is customer {
      Put a request object in the room;
      Send a notification to an agent;
   }
   else if a user is agent and there are request objects
      Create a private room;
      Put an agent and a customer in the room;
      Start Chat applets for the agent and the customer;
      Have the applets join the same chat group;
   }
}
```

ACCESS CONTROL

Access control is a critical element of supporting different types of collaborative activities. Without some form of access control, some collaborative tasks cannot be easily supported — e.g., paper submission and reviews, call centers, etc. If a system supports users with different roles (e.g., customers vs. agents), it almost invariably needs to support some form of access control. Access control can be important both from the perspective of privacy of collaborative activities as well as in guiding users to perform correct actions in a complex shared workspace. Both privacy and simplicity of interaction can enhance collaborative activities.

There are several models of access control, including general-purpose models such as access-control lists used in file systems, and models designed for collaborative systems such as the inheritance-based model by Shen and Dewan [11], the object-specific model in SOL [12], and the rule-based model by Edwards [4]. What model is appropriate to support the various uses of the room-based metaphor? We feel that there should be two guiding principles in choosing an access-control model for a specific situation: (1) the model should be as simple as possible so that it is easy to understand, easy to use by *end-users* and *administrators* of the system, and relatively easy to verify if access-control violations are a concern; and (2) it should provide the necessary flexibility for supporting a variety of potential uses. In particular, we felt it is better to sacrifice some generality in the access-control model (e.g., applicability to workflow-based collaboration as well as synchronous collaboration) if it leads to a simpler solution for our specific situation of applying the room-based metaphor to a variety of uses.

The model we feel satisfies our guiding principles for use in flexible room-based collaborative systems is similar to the access-control list model used in file systems such as AFS, adapted to make it appropriate for room-based systems. The model has several appealing characteristics. First, it is likely to be familiar to end-users if they have used standard network file systems. Second, it is an *end-user-oriented* model rather than a *programmer-oriented* model. Models such as those in [11] may require understanding of inheritance; inheritance is a nice concept from a programming perspective, but not something usually exposed to end-users. Such models are more likely to be better suited for programming access control in complex groupware systems.

Our access control model specifies access control on operations on three kinds of objects: *rooms*, *room objects*, and *actions* associated with events on rooms or room objects. The operations on rooms that can be permitted or denied to users include standard room operations such as *enter a room*, *leave a room*, *list* objects inside a room, and *add* objects to the room. In addition, standard access rights (read, modify, create, delete) can be specified on room attributes.

Permissions can be specified in terms of users or in terms of *roles*. Users can belong to one or more roles at a given time (specified as an attribute of the users). Users also have a a maximal permitted set of roles that they can become member of; at any given time, users belong to a subset of roles that they are permitted to join.

The following are the privileges at the level of objects.

read allows a user to open and read the object

write allows a user to overwrite the object

add attribute allows a user to add an attribute to the object

read attribute allows a user to read an attribute value

delete attribute allows a user to delete an attribute value

delete allows a user to delete the object from a room

see allows a user to list the object. Otherwise, the object is invisible to the user.

change access control list Changes access control list of the object

Attributes of an object are subject to access control. By default, only the creator of a object can add, read, or delete an object attribute. System-defined attributes cannot be deleted. They also cannot be modified, for obvious security reasons, except via requests to the system.

We feel that access control rules, as suggested in Edwards [4] are indeed useful (e.g., allowing a user access to a room during 9-5 only). However, we do not need to necessarily put such rules in the access control model. In our design, we can implement such rules at the event-action level. For instance, the rule about limiting access to a room between 9-5 can be encoded as follows:

(a) set timer events to be generated at 9 am and 5 pm on the given room.
(b) On 9 am timer event, take the limitAccess() action.
(c) On 5 pm timer event, take the permitAccess() action.

```
limitAccess() {
  /* save current ACL as an attribute of the room */
  currentRoom.saveACL = current ACL;
  currentACL = null;
}

permitAccess() {
  /* restore access */
  currentACL = currentRoom.saveACL;
}
```

USING THE MODEL TO PROVIDE ADAPTABILITY

Table 1 shows many of the key operations that are available to be executed from actions that are associated with events. These operations include facilities for creating or deleting rooms, adding or deleting users from rooms, adding URLs/applets to a room, moving or copying objects from one room to another, setting access control, notifying users, defining new attributes, etc.

room_create(rname)	Create a room. Initialize room structure.
room_enter(rid)	Enter a room. Put a user inside of a room
room_monitor(monitor)	Set up a monitor (callback) function for receiving notifications of state changes of rooms. The notifications are multicast using the atomic ordered multicast mechanism.
room_delete(rid)	Delete a room. Delete the room entry along with users and applets inside the room.
room_leave(rid)	Leave a room. Put a user outside of a room.
room_lookup(rid, lookup_list)	Look up a room. Return a room status
room_create_app(rid, gid, appname)	Create a new appropriate applet for the specified group. The new applet joins the group implicitly.
room_put_app(rid, gname, appname)	Put an applet in a room. The applet creates a new group and joins the new group.
room_put_grp(rid, gname)	Put an applet-group in a room.
room_move_app(src_rid, dst_rid, appid)	Move an applet from one room to another. The applet leaves its previous group and joins a new group.
room_remove_app(rid, appid)	Remove an applet from a room. The applet leaves its group.
room_remove_grp(rid, gid)	Remove an applet-group from a room.
room_get_acl(rid, uid)	Get a user's privilege level in a room.
room_set_acl(rid, uid, acl)	Set a user's privilege level in a room.
put_user(rid, uid)	Put a specified user in the room
list_object(rid, object_type)	Returns a list of room objects in a room.
read_object(rid, object_name, object_type)	Reads the content of the room object in a room
write_object(rid, object_name, object_type, object_content)	Writes the content of object in a room.
add_object(rid, object_name, object_type)	Add an object entry in a room.
delete_object(rid, object_name, object_type)	Delete an object entry from a room.
invoke_applet(room, user, appletType, objectName)	Start an applet on a user's desktop
lookup_users(rid)	Lookup users in a room.
find_user(uid)	Find a specified user in the system
notify_user(uid, message)	Send a notification to a user
add_atribute(attributeName)	Add a new user-defined attribute
set_attribute(attributeName, value)	Set the attribute value
get_attribute(attributeName)	Get the the specified attribute's value

Table 1: Examples of System-provided Operations on Rooms and Objects

We now show how to use these operations to adapt the behavior of rooms to various needs.

Call Center

We earlier gave an example of a an $< event, action >$ rule for call center. Now we consider the interaction of that role with access control. If we apply the design described above to *call center* room, two roles can be defined: *customers* and *agents*. At the level of a room, the customer have the access right of *add* that allows the customer to put a call object in the room. The *delete* right can also be given to the customer who is the owner of the object. On the other hand, the agents have the access right of *read*, *write*, and *delete*. The *call center* room can also be set in such a way that people in the room are not seen by other people in the same room. (*userlookup* attribute can be used for this matter)

As an another example, *group mail room* involves both room-level and object-level access controls. The *group mail room* allows all the members of the room to have the *add* privilege so that users can put (or add) email to others in the room. However, *read* access right of the object should also be enabled for the recipient so that only the recipient can read that message.

Paper Review Room

As described in earlier section, in the paper review room, a reviewer of a paper can read the paper and comment on it unless the reviewer is the author of the paper. We can model this type of room with some attributes and event/action methods associated with a paper object. First, the basic attributes should be defined both for a room and a paper object. First, a paper object should have attributes of user groups of *authors* and *reviewers*. Secondly, a room attribute of deadline is required to disallow applicants to add paper objects to a room after the deadline.

The following shows a pseudo code of the action for the event of *click on a paper object* for reading or reviewing a paper. When the applet is invoked by the invoke_applet, it should call *read_object(currentRoom, paper, commentApplet or readerApplet)* to get the paper object.

Event: Click on a paper object
Action:

```
actionForReadPaper {
  if (currentUser is in paper.Reviewers but not in paper.Writers)
    invoke_applet(currentRoom, currentUser, commentApplet,
    paper);
  else if (currentUser is in paper.Writers)
    invoke_applet(currentRoom, currentUser, readerApplet,
    paper);
}
```

Group Mail Room

Group mail room is a room where users can post mail to others and check their own mail. A mail object for this type of room need to have the attribute of recipient. The following shows the event *click on a mail object* and its corresponding action of reading mail. In the action, it invokes the reader applet only if the currentUser is the recipient of mail. When the reader applet is invoked, it gets the content of the mail by calling read_object(currentRoom, mail, readerApplet).

Event: Click on a mail object
Action:

```
actionForReadMail {
  if (currentUser is mail.recipient)
    invoke_applet(currRoom, currentUser, readerApplet, mail);
}
```

Transient Room

The transient room is a room which exists only for a certain period of time. The basic attribute of this type of room should include the expiration time. The following shows the event of time-out and the action performed for the event. In this example, the time-out event triggers the notification to all the users in a room and deletion of the room therby deleting all the objects in the room.

Event: Time-out
Action:

```
actionForTransientRoom {
  for each user in currentRoom
      notify_user(user, notificationMessage);
  room_delete(currentRoom);
}
```

CONCLUSIONS

We propose ways of supporting tailorability to room-based systems so that they can support a variety of collaborative applications that do not conform to a standard room-based metaphor that is typical in existing systems. The proposed design can be used both system builders as well as end-users to customize the behavior of room-based workspaces to meet the needs of a variety of collaborative applications. Examples of such applications include call centers, shared repositories, mailboxes, transient rooms, synchronized rooms, etc. In each case, variations are possible that perhaps may not even be anticipated by system designers.

To provide tailorability, we propose a model of room-based systems with the following features: (1) objects and rooms can have user-defined attributes; (2) a set of basic operations are provided to manipulate objects and rooms; (2) user operations on objects and rooms generate events; (3) end users can define actions that take place when events occur, where events can examine properties, examine or update attributes, or invoke operations on rooms or objects; and (4) a role-based access control model is defined to facilitate collaboration in situations where privacy or access control is required. We show that the model provides sufficient power to allow

tailorability of rooms to meet the needs of a variety of collaboration needs; yet the model is simple enough that it can potentially be exposed to end users in desired ways or via a simple scripting language.

We are currently providing support for the model in the Collaboratory Builder's Environment. Currently, the model can be used by system developers by programming in Java to build customized rooms. Future work includes exposing the customization facility to end-users via a simple scripting language. Work is also needed to better understand the implications of providing such a powerful customization facility to end-users; in general, there is a potential for benefit as well as a potential for mistakes. We hope to get a better handle on these issues by gathering experience with the facility in the context of on-going Upper Atmoshperic Research Collaboratory project at the University of Michigan as well as the use of Collaboratory Builder's Environment to support teaching activities in our classes.

ACKNOWLEDGMENTS

We thank Daniel Atkins, Robert Clauer, Farnam Jahanian, Gary Olson, Terry Weymou th 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 Richard Bentley and Paul Dourish. Medium versus mechanism: Supporting collaboration through customization. In *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work*, pages 133–148, September 1995.

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

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 W. Edwards. Policies and Roles in Collaborative Applications. In *Proceedings of the Sixth Conference on Computer Supported Cooperative Work*, pages 11–20, November 1996.

5 G. Fitzpatrick, W. Tolone, and S. Kaplan. Work, locales and distributed social worlds. In *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work*, pages 1–16, September 1995.

6 R. W. Hall, A. Mathur, F. Jahanian, and A.Prakash an C. Rasmussen. Corona: A communication service for scalable, reliable group collaboration systems. In *Proceedings of the Sixth Conference on Computer Supported Cooperative Work*, pages 140–149, Boston, Massachusetts, November 1996.

7 S. Kaplan, W. Tolone, and D. Bogia C. Bignoli. Flexible, active support for collaborative work with ConversationBuilder. In *Proceedings of the Fourth Conference on Computer Supported Cooperative Work*, October 1992.

8 J.H. Lee, A. Prakash, T. Jaeger, and G. Wu. Supporting Multi-User, Multi-Applet Workspaces in CBE. In *Proceedings of the Sixth Conference on Computer Supported Cooperative Work*, pages 344–353, Boston, Massachusetts, November 1996.

9 M. Roseman and S. Greenberg. Building flexible groupware through open protocols. In *Proceedings of the 1993 Conference on Organizaional Computing Systems*, pages 55–65, August 1995.

10 M. Roseman and S. Greenberg. TeamRooms: Network Places for Collaboration. In *Proceedings of the Sixth Conference on Computer Supported Cooperative Work*, pages 325–333, Boston, Massachusetts, November 1996.

11 H. Shen and P. Dewan. Access Control for Collaborative Environments. In *Proceedings of the Fourth Conference on Computer Supported Cooperative Work*, pages 51–58, October 1992.

12 G. Smith and T. Rodden. Sol: A shared object toolkit for cooperating interfaces, technical report cseg/7/1995. Technical report, Lancaster University, 1995.

13 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.