Proceedings of the Second European Conference on Computer-Supported Cooperative Work
Bannon, L., Robinson, M. & Schmidt, K. (Editors)
September 25-27, 1991, Amsterdam, The Netherlands

Personalizable groupware: Accommodating individual roles and group differences

Saul Greenberg
Department of Computer Science, University of Calgary, Canada T2N 1N4

Abstract—For groupware to be considered successful, it must be usable and acceptable by most, if not all, members of the group. Yet the differences present between group members—their varying roles, needs, skills—and the differences between groups as a whole are a serious obstacle to achieving uniform acceptance of the groupware product, especially when the product treats all people and groups identically. This paper raises several consequences of not accommodating individual differences, and then offers a possible solution to the problem. First, instances of groupware failure are described: the inability of the group to reach a critical mass; the unequal accessibility of the groupware by participants; the failure to accommodate the different roles participants may play; the failure to balance the work done against the benefits received; and the failure of groupware to evolve with the needs of the group. Second, the notion of *personalizable groupware* is proposed, defined as a system whose behaviour can be altered to match the particular needs of group participants and of each group as a whole. Finally, the paper presents SHARE, a working example of personalizable groupware. SHARE is a shared screen system that offers its users a flexible choice of floor control models to help them mediate their interactions with the shared application.

1. Introduction

Design teams now build single user systems with good interfaces suitable for selling to the mass market. While the product may not be to everyone's tastes, the vendor's goal is to have it acceptable to enough customers to make its production an economically worthwhile venture. Those customers with differing requirements or preferences simply go to another product, or do without.

Designers of groupware face more rigid criteria. Of course, the product must satisfy enough groups to be commercially viable. But unlike single user software, the product chosen by the group should almost by definition be acceptable and usable by almost *all* its members. While this may seem a strong claim, experience has revealed conditions of groupware failure due to its inability to satisfy all its supposed users.

Consider the following issues and instances of groupware failures.

a) *A critical mass of system adopters may not be reached if too many people opt out of using the groupware product.* A new feature-rich asynchronous conferencing system was introduced at a work site to replace an old but still usable one. Although the new product had a strong champion and was used heavily by 20% of the departmental community (predominantly upper management), a good number of the staff resisted switching to it mostly due to the overhead of learning and using the new system's primitive user interface. Conference activity dwindled as contributors realized they were not reaching the majority of the intended audience. The new system was eventually shelved until a better interface could be developed. (See also Markus and Connolly's 1990 discussion of payoff criteria for adopting technology).

b) *Participants who cannot or will not use the technology face the danger of becoming second class citizens within their own group.* Participants of CAPTURE LAB face-to-face meetings can access a large public screen through their personal computers (Mantei 1988). Austin, Liker and McLeod (1990) noticed that CAPTURE LAB participants who rated themselves as less than 25% proficient with its computer technology were unlikely to use it. Those with greater proficiency were equally likely to use or not use it. Austin et al suggest the existence of a "proficiency floor", above which an individual would perceive themselves as having sufficient competency to use the technology. Those below the floor would avoid its use.

   Our similar observation concerned face to face meetings whose members had shared access to a spreadsheet program being projected at the front of the room. Participants who were not familiar with the spreadsheet package or who were not adept typists were inhibited from adding to the model being displayed.

c) *New people joining an established but evolving group must be able to use the system adeptly, otherwise cliques of expertise may evolve.* The initial joining period is critical for new members to assert themselves into the established group. In the spreadsheet case above, we observed that the complex uses made of the spreadsheet package by the already adept but established group made the system almost unreachable by new participants. The newcomers, although familiar to some extent with the technology, were unfamiliar with the ways it had been applied. Unconsciously, the established group became a clique of elite controllers.

d) *Participants in a group may have quite different roles that are not recognized by the groupware product.* Consider a screen sharing package that enforces a pre-emptive floor control protocol (ie anyone can pre-empt control away from anyone else). We have observed one interacting team of a senior and junior person, where the junior person was quite uncomfortable and almost unwilling to take control away from the senior person.

   Another effect of role differences was noticed in the CAPTURE LAB study mentioned above. Austin et al (1990) write that use of the public screen technology was proportionally higher by influential group members when contrasted to members with less influence, and by males when compared to females. They suggest that some group members perceived the public screen as a means of influencing other group members. The CAPTURE LAB technology does

not recognize these effects; unequal use of the technology is neither encouraged nor guarded against.

e) *There is often disparity between who does the work and who gets the benefit when using groupware.* Grudin (1988) argues that groupware applications often fail because they require that some group members do additional work even when they are not the ones who perceive or receive a direct benefit. A familiar example is a group appointments scheduler that requires all members to do the extra work of keeping their on-line calendars up to date for the benefit of the person (usually the manager) who schedules most of the meetings (Grudin 1988; Bullet and Bennett 1990).

f) *Group needs evolve rapidly, not only from meeting to meeting but within the course of a meeting. The Groupware must keep pace.* Users of COGNOTER, a multi-user idea organizer, would often split into multiple sub-groups to work on ideas raised in the brainstorming session (Foster and Stefik 1986; Tatar, Foster and Bobrow 1991). An early COGNOTER design created a formal division of the sub-groups into "rooms" (Stefik et al 1987). However, these formal boundaries did not reflect the evolving sub-group membership or interactions between them. As a result, Stefik et al conjectured that formation and dissolution of subgroups would be inhibited.

We believe that a prerequisite to successful groupware is that it must be acceptable by most or all members of the group. This can be accomplished in several ways. First, groupware use can be so generic or transparent that almost anyone can use it. For example, tele-conferencing requires only normal interaction skills of participants, while vanilla electronic mail requires mostly familiarity with an editor of choice. Second, the service provided could be so valuable or so entrenched in the organization (perhaps politically) that all users are effectively "forced" to accommodate to it.

This paper raises a third possibility: that groupware be *personalizable* so that it can accommodate individual roles and group differences.

2. Personalizable Groupware

Personalizable groupware is defined as groupware whose behaviour can be tailored to match the particular needs of group participants (ie each member of the group may observe a different behaviour), and the particular needs of the group as a whole (ie each group may observe a different collective behaviour). Illustrating the first point, suppose that a small group of three people, say two architects and a client, are in a remote real time meeting consulting over blueprints displayed through a shared computer aided design (CAD) package. Depending upon personal needs and tastes, each participant may require a slightly or even completely different style of user interface. For instance, the senior architect may have complete access to the controls in the CAD package, while the apprentice architect may only be able to observe the drawing. The CAD-naive client may still be able to gesture and sketch around the existing drawing through a very simple graphics pencil. Illustrating the latter point about between-group differences, the same groupware tool may be used by a group of contractors to implement the blueprint. In this case, the contractors may only be able to annotate that part of the drawing that they are responsible for, perhaps to indicate deviations they had taken from the design.

Although this paper names and champions the concept of personalized groupware, it is not a novel idea. A handful of groupware systems incorporate some level of personalization. Consider QUILT, a computer-based tool for collaborative document production (Leland, Fish and Kraut 1988). A person's ability to manipulate a document is tailored to one's position in a permission hierarchy. Some positions are readers, commentors and co-authors, each with greater powers of annotation and revision. Another example is INFORMATION LENS (Malone et al 1987), an information manager for mail and news. Here, users can construct their own semi-structured templates representing particular types of mail they wish to compose, can create their own rules to filter incoming information in quite sophisticated ways, and can create custom views that summarizes selected information (see also OBJECT LENS, Lia and Malone 1988). A third example is CRUISER, a video-based "virtual hallway" system that facilitates casual interaction (Root 1988; Fish 1989). People in the CRUISER network can control privacy by setting a variety of personal permissions that limit how others can observe and/or interact with them. Finally, the VIRTUAL LEARNING COMMUNITY (Johnson-Lenz and Johnson-Lenz 1991) is an asynchronous conferencing system that lets a conference facilitator tailor the groupware to support the purpose and the variety of the group's activities. For example, the boundaries that define group membership can be adjusted to either enforce equal participation of all group members, or to allow "lurkers"—people who follow the group's discussion but who never express themselves.

Aside from these four and a few other notable exceptions, personalization is usually ignored—sometimes intentionally—in groupware design. Consider at the extreme the point of view of "groupware as mechanism", where the computer's role is to provide a single well-defined mechanism that incorporates some social model of interaction (Johnson-Lenz and Johnson-Lenz 1991). Here, the social model enforced by the system and imposed on its users is an explicit attempt by the designer to provide methods to help keep the group on task, enforce roles and commitments, and make the group efficient and productive (a common goal of group decision support systems). While such systems certainly have a positive role in some settings, the Johnson-Lenzs argue that inflexible structures may trigger organizational and individual resistance, and that flexible patterns encouraging personal initiative are just as important as well-defined group procedures. The negative outcome of mis-matched groupware as mechanism may well be inflexible systems that force its users to do things in undesirable and unproductive ways, where people must change their behaviour to match the machine's model, rather than vice versa. At their worst, users will perceive such systems as "fascist software" and will avoid their use (for example, see Bair and Gale's 1988 report on the COORDINATOR).

Yet we do not advocate the chaos of a completely customizable and unstructured interface, for these will often leave its users at a lost of what to do (Johnson-Lenz and Johnson-Lenz 1991; Thimbleby 1980; but see Dykstra and Carasik 1991 for another point of view). We see personalizable groupware as a way to soften the negative effects of groupware as mechanism by offering a range of structures that reflect a complementary range of the group's requirements. The groupware designer's job is to determine what parts of the groupware system should remain immutable and what part personalizable, and then to set reasonable constraints on the personalization allowed.

3. A working example of personalizable groupware

3.1 Shared window systems and floor control.

"Collaboration aware" groupware for real-time sharing of work explicitly recognizes the existence of each participant in the collaboration (Lauwers and Lantz, 1990). For example, a collaboration-aware sketchpad can be designed to support what people actually do in collaborative design (Tang 1991) eg gesturing by displaying multiple cursors, and concurrent work by allowing participants to sketch simultaneously into a common shared workspace (Greenberg and Bohnet 1991). It is unlikely, however, that collaboration aware systems will have a major impact on the market in the next few years. Not only are they technically difficult to build, but the prerequisites for design are lacking—we really know very little about how people work together.

An alternate approach stems from the old idea of taking a single-user application and sharing it between participants of an on-line meeting through a "shared screen" or "shared window" (Engelbart and English 1968; see Greenberg 1990 for a survey). Each participant would have an identical view of the running application and an opportunity to interact with it. Special "view-sharing" software would allow *any* unaltered single-user application to be brought into a meeting; the application itself would have no awareness that more than one person was using it. This scheme is usually implemented by merging all participants' inputs into a single stream sent to the application, and by sending a copy of the application's output stream to every participant's workstation[1]. While limited in power, shared views are a logical and reasonable "stepping stone" to true collaboration aware systems (Johansen 1989).

A catch of sharing single-user applications is that users must take turns; attempts at simultaneous activity would have the input to the application garbled (eg two people typing at the same time would have their input merged into a nonsense sentence; simultaneous attempts to move the single cursor would result in "cursor wars"). Consequentially, most shared view systems enforce serial turntaking through some type of explicit floor control mechanism (see Table 1 for a brief summary of several floor control protocols and some systems that implement them). For example, the CAPTURE LAB, the face-to-face meeting room that allows participants to share a single large screen, forces a *pre-emptive protocol* where one can pre-empt the floor away at any time from the current floor holder (Mantei 1988). The commercial TIMBUKTU product offers a *free floor*, where any participant can enter any input at any time—turntaking must be mediated out-of-band (Farallon 1988). In contrast, CANTATA uses a first in, first out *queue with explicit release*, where the current floor holder must release the floor before the next person in line gets it (Chang, Kasperski and Copping 1987).

While existing shared view systems usually offer one particular style of protocol for floor control, no literature provides justification as to why that style was chosen. Is there, in fact, a "best" general floor control policy? We believe there is not, for

[1]While simple to do in shared terminal systems (eg by using pseudo-tty filters to trap i/o in UNIX), the technology of sharing windows is far more complex and is fraught with many technical issues and difficulties (Lauwers, Joseph, Lantz and Romanow 1990; Greenberg 1990).

groups will differ in how its members interact with each other. As Lauwers writes (the designer of the technically sophisticated DIALOGO view sharing system) "...the only certainty [about floor control] is that no one policy will suffice for all groups, in all situations" (Lauwers 1990 p97).

We can easily envision situations where groups desire different policies. A small group of practised collaborators may prefer the free floor, choosing to mediate interaction by voice alone, while a larger cooperating group may employ pre-emptive control to avoid accidental input merging. As a case in point, programmers using the SharedX shared window system (Garfinkle, Gust, Lemon and Lowder 1989) reported the need to alternate between free floor when brainstorming to system-controlled one-person-at-a-time when wanting to make sure a particular piece of code was coded correctly (reported in Lauwers 1990). In distance education, a seminar presenter or teacher may use a "central moderator" approach to hand off and take back control from members of the audience who are posing questions. In a formal meeting context, a group decision support system may enforce a round-robin or queue policy.

Protocol	Description	Where implemented
Free floor	Any participant can enter input at any time, with floor control mediated out of band usually through a voice channel. Accidental mixing of multiple input streams is possible.	TIMBUKTU (Farallon 1988) SHARE (Greenberg 1990)
Pre-emptive	Any participant can pre-empt control away at any time from the floor holder.	CAPTURE LAB (Mantei 1988) DIALOGO (Lauwers 1990) SHARE (Greenberg 1990)
Explicit release	The floor holder must explicitly release the floor before another participant may claim it.	CANTATA (Chang et al 1987) SHARE (Greenberg 1990)
First in, first out queue with explicit release.	Participants line up to take turns, where the floor, once explicitly released by the floor holder, is given to the person at the front of the line.	CANTATA (Chang et al 1987) VCONF (Lauwers 1990)
Central moderator	A moderator oversees all activity and decides who should hold the floor, usually by monitoring requests for the floor by other participants.	RTCAL (Sarin & Greif 1985) SHARE (Greenberg 1990)
Pause detection	The floor is made available to any participant only after the system detects a suitable pause of activity by the floor holder.	EMCE (Garcia-Luna-Aceves et al 1988) SHARE (Greenberg 1990)

Table 1. Some floor control protocols that have been implemented in view-sharing systems

Lauwers (1990) suggests that even aspects of the operating environment—the availability and quality of an audio/video channel, the length of communication delays—will also influence the choice of policy. For example, a group preferring to use out-of-band traditional social protocol (ie voice, gestures) to mediate a free floor may suffer increasing accidental input collisions as a function of lengthening the communication delays and degrading the audio/video channel.

3.2 Personalizable floor control in SHARE

Lauwers (1990) recommends that an ideal shared window system should "support a broad range of [floor control] policies... in an architecture capable of switching between different policies depending on user preferences and the operating environment". We have taken up this challenge. Based on the belief that no single policy can address adequately all groups, we have designed and implemented SHARE, a view-sharing system that supports personalizable floor control.

SHARE is a "policy free" view-sharing system whose kernel supports primitives upon which one can build a broad range of policies to manage floor control (Greenberg 1990). Its architecture is comprised of four entities (Figure 1).

a. The *Registrar* is a daemon process responsible for initiating the shared view conference set up and tear down, the selective entry and departure of participants while the conference is in progress, and feedback of the conference's current status. One or more conferences may be established via the Registrar, and participants may join as many of the running conferences as they wish. There is one Registrar daemon for the entire network
b. The *View Manager* is the technical heart of the system, a process responsible for synchronizing and transmitting the shared views between participants. There is one View Manager per conference. In the current implementation, the View Manager provides only rudimentary shared views of a text-based terminal window running UNIX applications (eg UNIX SHELL, the GMACS editor, etc), and cannot share views of graphical mouse-based applications. While this limits the true usability of SHARE, we avoided the extremely time-consuming implementation of a graphics-based window-sharing system[2], allowing us to concentrate on other design aspects such as floor control.
c. The *Chair Manager* process is responsible for interpreting (but not setting!) a floor control policy. It receives directions from the Turntaker (see next point) on what *observe* and *write* permissions it should set on each participant's view into the application. There is one Chair Manager per conference.
d. The *Turntaker* process sets a particular floor control policy and presents the interface to it. User interactions are interpreted and translated into protocol primitives, which is sent to the Chair Manager. There is usually one Turntaker per participant.

When a person asks for a new shared view meeting, the Registrar will create instances of the View Manager and Chair Manager processes. A fully interactive and sharable UNIX SHELL window will then appear on the person's workstation.

---

[2]SHARE was up and running within two man-months of work. In contrast, the technically sophisticated SHAREDX (Garfinkel et al 1989) and DIALOGO (Lauwers 1990) systems required several man-years to implement.

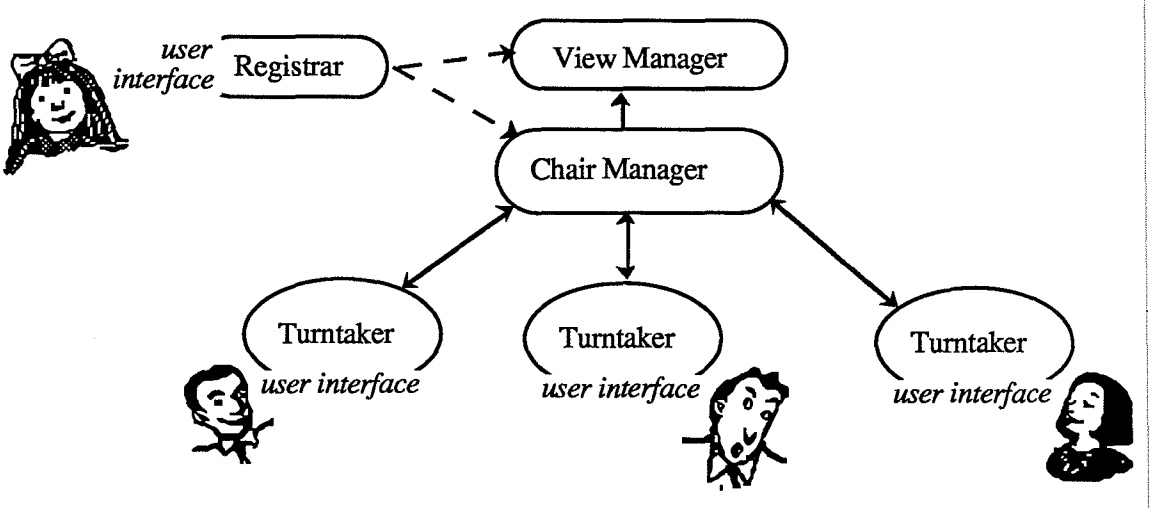
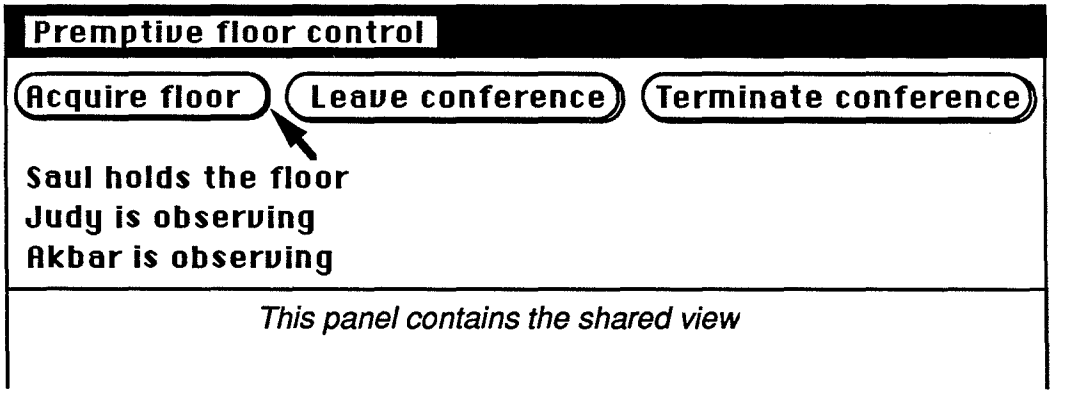


Figure 1. Main architectural components of *Share*




```
ForEach participant
    if participant[i].id = self
        SendToChairManager (SETFLOORPERMISSIONS, participant[i].id, "Write")
    else
        SendToChairManager (SETFLOORPERMISSIONS, participant[i].id, "Observe")
```

Figure 2 Pre-emptive floor control: the interface and the protocol sent for pre-empting control

Other people may now join the meeting, which will cause a copy of the UNIX window to appear on their workstations.

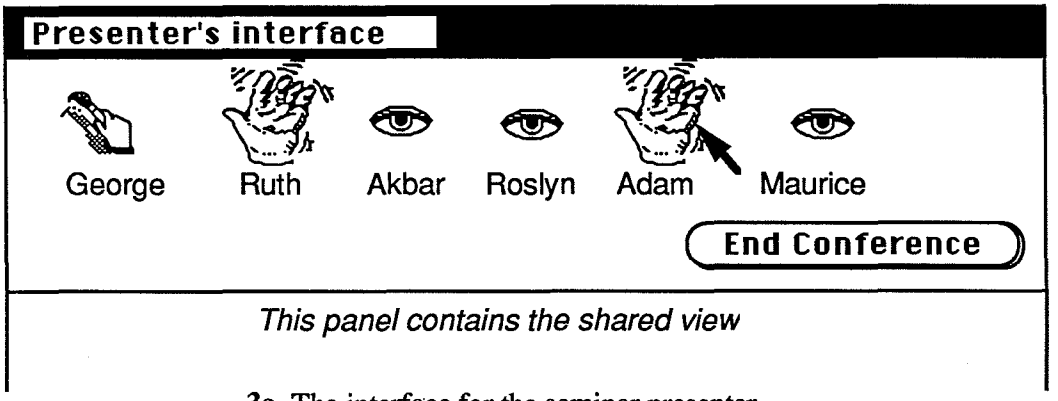
At this point, no floor control policy has been specified. The Chair Manager will by default allow only the original meeting creator to enter input into the shared view. Others can observe but not interact with the application. The actual floor control policy resides in the Turntaker processes—one activated for each participant—that presents its users with an interface to a particular floor control scheme and converts a user's request into a set of primitive messages sent to the Chair Manager (as listed in the Appendix). These primitives include asking the Chair Manager: to set any participant's *observe* and *write* status; to get information about other participants; and to forward messages to other Turntakers. The result is that different floor control policies are easy to implement. Each participant's Turntaker and its interface may be specialized to reflect one's specific political role in the meeting, and floor control policies can even be switched on the fly. The Appendix gives detail on the sequence of events that occurs between the Turntaker and the Chair Manager.

Figure 2, for example, illustrates a simple pre-emptive floor control interface supported by SHARE. Here, all participants have invoked the Turntaker process that enforces the pre-emptive policy. When a user selects the "Acquire floor" button, the Turntaker requests the Chair Manager to assign *write* permission for that person, and *observe-only* status to all other meeting participants (bottom of Figure 2, Appendix). The Turntaker also tells its user who is in the meeting and who currently holds the floor.

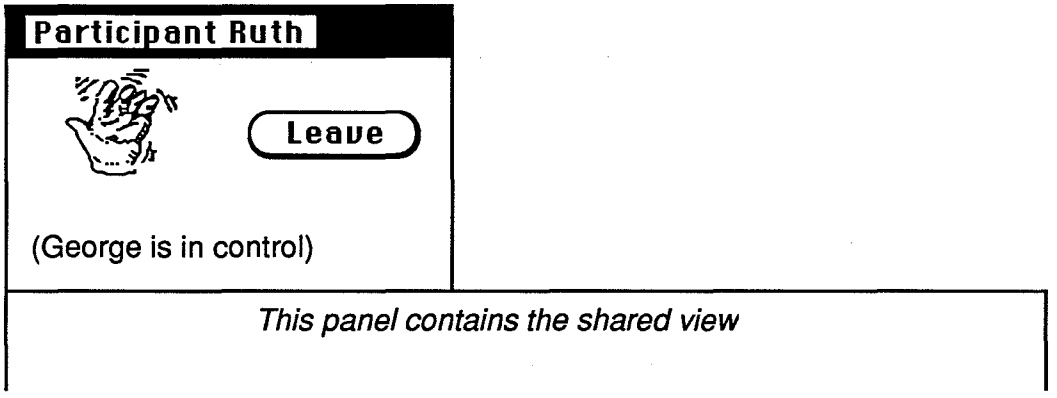
In contrast, Figures 3a and 3b illustrate the more complex "central moderator" protocol (Table 1) used by a seminar presenter and by the audience respectively. This scheme requires two different (but coordinated) styles of Turntaker processes representing the roles of the presenter and of the seminar participants. The single presenter would invoke the Presenter Turntaker, while each audience member would invoke their own copy of the Participant Turntaker. In Figure 3a the presenter sees: a list of all participants who desire the floor (the raised hands); who currently holds the floor (the writing hand); and who is just observing (the eyes). The presenter can assign or take away interaction permission by selecting the icon portraying the chosen participant. In Figure 3b participant Ruth has requested the floor by selecting her single icon (selecting it again will put her back to observer status). She also sees that participant George is the current floor holder. It is worth noting that the presenter and the participant also have different controls affecting conference departure. While the presenter can terminate the conference for all participants by pressing the "end conference" button, participants can only leave (which does not affect any other participants).

Another floor control policy we have implemented is pause detection (Table 1). If there is a pause in input activity of the current floor holder for several seconds, the floor becomes free. The floor is then automatically assigned to the next participant that attempts to interact with the application (eg by typing). Because a short pause is required before the floor is freed (we use a delta of 2 seconds), the accidental overlap of input commonly seen with a free floor policy is eliminated. We have found this method effective for general use by small cooperating groups as it reflects a person's natural and implicit way of mediating turntaking in conversation, unlike the other methods mentioned here that require one to explicitly request a turn.

Some shared view architectures have similarities to SHARE. We are aware of two other systems—DIALOGO (Lauwers and Lantz, 1990) and SHAREDX (Garfinkle,



3a. The interface for the seminar presenter



3b. The interface for a participant who can only request the floor

Figure 3. Two roles for participants in a centralized floor control interface.

Gust, Lemon and Lowder 1989)—that have the same potential in its architecture for flexible floor control. What is novel is that we have striven to give the power of this flexibility directly to the end users. Additionally, SHARE has an extensible open architecture. Given the protocol primitives understood by the Chair Manager (as mentioned in the Appendix), a programmer should have little trouble designing new Turntakers. Access to SHARE's source is not required, and the kernel does not need recompilation[3]. In the current version of SHARE, people and groups personalize their system by selecting the desired floor control policy from a library of policies—the library is extended only by programming new Turntakers. We foresee providing end users with the power to construct and/or extend a floor control policy through a prototyping tool or through a scripting language.

4. Summary

In spite of individual differences and preferences between group members and between groups, most groupware now available requires its users to conform to a single model of use. As a consequence, people may opt out of using the product, which seriously threatens the potential benefit the system can offer to the group as a whole. This paper argued that personalizable groupware can lead to wider acceptance of the product by offering a system that conforms to the individual needs of participants and of groups.

We have presented SHARE, a shared view system, as a simple example of personalizable groupware. As these systems allow only serial interaction with the running application, floor control must be mediated. SHARE supports between group differences by providing an extensible library of floor control policies for groups to chose from. Within-group differences are managed as well, for a particular policy may provide several roles appropriate to a participant's position within the group (as in the seminar presenter/student case).

Programming the differet policies proved both easy and quick. What is missing is an end user evaluation. Although we have our own successful experiences using SHARE's various floor control policies, it remains to be tested in an unbiased environment, preferably by tracking its long term use by people requiring desktop conferencing capabilities in real settings. This will be difficult to do in practice, for it demands the costly process of bringing Share to near-product functionality and reliability.

Personalizable groupware has a long way to go. On the social side, we must understand how group participants vary and how groups differ. This is fundamental if we are to supply appropriate flexibility to handle that diversity. On the technical side, we must provide not only architectures appropriate to personalization (an interesting possibility is a self-adaptive interface; Greenberg 1985), but also the means to allow the participant and the group to select the method the best fits their needs.

---

[3]We recently discovered that the ASPECTS, a commercial groupware product, allows a single group mediator to select three floor control levels: free floor, serial, and central moderator (Group Technologies 1991). Unlike SHARE, however, ASPECTS is a closed system. Policies are hard wired and can only be extended by altering the source code.

Appendix. Protocol primitives used between the Turntaker and the Chair Manager

This appendix describes the protocol primitives transmitted between the Turntaker and Chair Manager processes and gives examples of their use. The Chair Manager and the View Manager form the kernel of SHARE, with participants creating, entering and leaving conferences through the Registrar. The Turntaker process, which implements a particular floor control policy, may be created and destroyed by the participant at any time. When a Turntaker is created, it establishes a UNIX socket connection with the Chair Manager and then presents an interface to the user (as in Figures 2 and 3). The Turntaker then embodies the particular floor control policy by sending message primitives to the Chair Manager and to other Turntaker processes. The list below describes most of the primitives exchanged. The <id> field indicates the identification of the participant. The '|' is used as a field delimiter.

Protocol	Explanation
TURNTAKERREGISTER <id>	The Turntaker registers with the Chair Manager. If the participant id is specified, the Chair Manager will signal the Turntaker when that participant leaves the conference, upon which the Turntaker will usually destroy itself.
REGISTERUSER <id>	The Chair Manager informs the Turntaker that a new participant has connected to the conference.
UNREGISTERUSER <id>|<id>|...	The Turntaker requests the Chair Manager to unregister the participants specified by their id's from the conference. This will delete that user's shared view from their workstation.
ENDMEETING	The Turntaker requests the Chair Manager to terminate the entire conference. This will unregister all participants and destroy the meeting's Chair and View Manager processes.
SETFLOORPERMISSIONS <id><permission>| <id><permission>|...	The Turntaker requests the Chair Manager to assign *observe-only* or *write* permission to the participants listed.
SETMETAFIELD <id>|message|<id>|message|...	The Turntaker requests the Chair Manager to attach a message to a participant's id and then forward it as a STATUSCHANGED message to all other Turntaker processes. Usually used to define a protocol between Turntakers.
GETINFORMATION <bit mask>|<id>|<id>|...	Turntaker requests information on some or all participants (this information is stored by the Chair Manager). The bit mask indicates the information desired, which includes: •the participants login name, •the host and port number of the participant's machine, •the participant's pseudo-terminal containing the shared view •the message (meta) field associated with the participant •the current *observe/write* status of the participant.
STATUSCHANGED <bit mask>|<id>|message|...	When a participant's status information is changed, the Chair Manager broadcasts the particular change to all Turntakers. The bit mask is as noted in GETINFORMATION.
TIMERCHANGE <seconds>	Turntaker specifies a time delay value for pause detection.

Consider a meeting implementing the simple pre-emptive floor control policy shown in Figure 2. When the Turntaker process is started, it does the following.

1. Register with the Chair Manager via the TURNTAKERREGISTER request.
2. Ask the Chair Manager for all it knows about the other conference participants via the GETINFORMATION request; this will include who is in the conference, who holds the floor, and so on.
3. Present the user interface, listing the current status of other participants (eg "Judy is observing").
4. When a message is received from the Chair Manager indicating a change of status of any of the participants (the STATUSCHANGED message), then update the status information on the display.
5. When the user selects the 'Acquire Floor' button, tell the Chair Manager to change the permissions to *write* for self, and *observe* for all others via the SETFLOORPERMISSIONS message. The Chair Manager acknowledges via a STATUSCHANGED message.
6. If the 'Leave Conference' button is pressed, the Turntaker notifies the Chair Manager via the UNREGISTERUSER message. When the Chair Manager acknowledges the request, the Turntaker will destroy itself.
7. Alternatively, if the Terminate Conference button is pressed, the Turntaker will send the ENDMEETING message to the Chair Manager.

Changing this floor control policy to explicit release (explained in Table 1) is fairly straight forward. Substituting for step 5 above:

5a. Alter the "Acquire Floor" button so that it is enabled only when no participants hold the floor (ie have *write* permission), and dimmed otherwise. When enabled and selected, the Turntaker requests the Chair Manager to set *write* permission for self. The button's label is then changed to 'Release Floor'. Other Turntakers will be informed of the change in status and will dim their 'Acquire Floor' buttons.

5b. When 'Release Floor' is pressed, the Turntaker tells the Chair Manager to change the permission of self from *write* to *observe*. A status message indicating that floor permissions have changed is sent automatically by the Chair Manager to all Turntakers, who in this case react by enabling their "Acquire Floor" button.

A more complex example is the centralized floor control interface shown in Figure 3, where a participant may request the floor from the presenter. As the Chair Manager has no primitive that directly supports a 'floor request', this must be implemented as a protocol between cooperating Turntakers. In our implementation, Turntakers attach a "raised hand" and "lowered hand" message to a participant id and transmit status changes to each other. To illustrate, when the participant requests the floor by selecting the icon (Figure 3b), the Turntaker sends the Chair Manager the SETMETAFIELD primitive along with the participant's id and the message "raised hand". This is then forwarded by the Chair Manager to the presenter's Turntaker (Figure 3a), which will interpret the message and change the appropriate icon on the display. The important point here is that this extended protocol is implemented completely within the Turntakers; no change had to be made to the code in the Chair Manager. The rest of the centralized floor control interface is straight forward. When the presenter assigns the floor to a participant, the appropriate permission fields are set and sent via the SetFloorPermissions message. When the Turntaker of the participant chosen receives its StatusChanged message, it will change the icon being displayed to a writing pen.

As a final twist, we can implement a selective free floor policy in the above centralized floor control scheme. All that is required is to set write permission for the presenter, which is maintained even when a student has write permission.

Acknowledgements. The Alberta Research Council sponsored part of this research during my position there as a National Science and Engineering Research Council Industrial Research Fellow. Many thanks to Ralph Bohnet, who was instrumental in implementing SHARE.

References

Austin, L. C., Liker, J. K. and McLeod, P. L. (1990) "Determinants and patterns of control over technology in a computerized meeting room." In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, p39-52, Los Angeles, California, October 7-10, ACM Press.

Bair, J. H. and Gale, S. (1988) "An investigation of the Coordinator as an example of computer supported cooperative work." Hewlett Packard Laboratories, California, Unpublished.

Bullen, C. V. and Bennett, J. L. (1990) "Learning from user experience with groupware." In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, Los Angeles, California, October 7-10, ACM Press.

Chang, E., Kasperski, R. and Copping, T. (1987) "Group co-ordination in participant systems." Technical report, Department of Advanced Computing and Engineering, Alberta Research Council, Calgary, Alberta, Canada, September.

Dykstra, E.A. and Carasik, R.P. (1991) "Structure and support in cooperative environments: The Amsterdam Conversation Environment." In S. Greenberg (ed.):*Computer Supported Cooperative Work and Groupware*, Academic Press, London. Originally published in *Int J Man Machine Studies*, **34**(3), March.

Engelbart, D. and English, W. (1968) "A research center for augmenting human intellect." In *Proceedings of the Fall Joint Computing Conference*. Montvale, NY, Fall, AFIPS Press.

Farallon (1988) "Timbuktu user's guide." Manual, Farallon Computing Inc., Berkely, California,

Fish, R. S. (1989) "Cruiser: A multi-media system for social browsing." *The ACM SIGGRAPH Video Review Supplement to Computer Graphics*, **45**(6). ACM Press, Baltimore, MD. Videotape.

Foster, G. and Stefik, M. (1986) "Cognoter: Theory and practice of a Colab-orative tool." In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '86)*, p7-15, Austin, Texas, December 3-5, ACM Press.

Garcia-Luna-Aceves, J.J., Craighill, E.J. and Lang, R. (1988) "An open-systems model for computer-supported collaboration." In *Proceedings of the IEEE Conference on Computer Workstations*, p40-51, March.

Garfinkel, D., Gust, P., Lemon, M. and Lowder, S. (1989) "The SharedX multi-user interface user's guide, version 2.0." Research report STL-TM-89-07, Hewlett-Packard Laboratories, Palo Alto, California, March.

Greenberg, S. (1990) "Sharing views and interactions with single-user applications." In *COIS '90: Proceedings of the Conference on Office Information Systems*, Boston, April.

Greenberg, S. and Bohnet, R. (1991) "GroupSketch: A multi-user sketchpad for geographically-distributed small groups." In *Proceedings of Graphics Interface '91*, Calgary, Alberta, June 5-7. Also available as Reseach report 90/414/38, Dept of Computer Science, University of Calgary, Alberta, Canada.

Greenberg, S. and Witten, I. H. (1985) "Adaptive personalized interfaces -- a question of viability." *Behaviour and Information Technology*, **4**(1), pp. 31-45, January.

Group Technologies (1991) "Aspects: The first simultaneous conference software for the Macintosh, Version 1." Manual, Group Technologies Inc, Arlington, Virginia.

Grudin, J. (1988) "Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces." In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pp. 85-93, Portland, Oregon, September 26-28, ACM Press.

Johansen, R. (1989) "User approaches to computer-supported teams." In M.H. Olson (ed.): *Technological Support for Work Group Collaborations*, p1-32, Hillsdale, New Jersey, Lawrence Erlbaum Associates.

Johnson-Lenz, P. and Johnson-Lenz, T. (1991) "Post-mechanistic groupware primitives: Rhythms, boundaries and containers." In S. Greenberg (ed.): *Computer Supported Cooperative Work and Groupware*, Academic Press, London. Originally published in *Int J Man Machine Studies*, **34**(3), March.

Lauwers, J. C. (1990) "Collaboration transparency in desktop teleconferencing environments." PhD Thesis, Available as Technical Report CSL-TR-90-435, Stanford University, Computer Systems Laboratory, Stanford, CA, July.

Lauwers, J. C. and Lantz, K. A. (1990) "Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems." In *Proceedings of the ACM/SIGCHI Conference on Human factors in Computing*, Seattle, April, ACM Press.

Lauwers, J. C., Joseph, T. A., Lantz, K. A. and Romanow, A. L. (1990) "Replicated architectures for shared window systems: A critique." In *Proceedings of the Conference on Office Information Systems*, p249-260, Boston, April 25-27.

Leland, M. D. P., Fish, R. S. and Kraut, R. E. (1988) "Collaborative document production using Quilt." In *Proceedings of the Conference on Computer-Supported Cooperative Work*, p. 206-215, Portland, Oregon, September 26-28, ACM Press.

Lia, K.-Y. and Malone, T. W. (1988) "Object Lens: A 'spreadsheet' for cooperative work." In *Proceedings of the Conference on Computer-Supported Cooperative Work*, p. 115-124, Portland, Oregon, September 26-28, ACM Press.

Malone, T. W., Grant, K. R., Lai, K.-Y., Rao, R. and Rosenblitt, D. (1987) "Semi-structured messages are surprisingly useful for computer-supported coordination." *ACM Trans Office Information Systems*, **5**(2), p115-131, April.

Mantei, M. (1988) "Capturing the Capture concepts: A case study in the design of computer-supported meeting Environments." In *Proceedings of the Conference on Computer-Supported Cooperative Work*, 257-270, Portland, Oregon, September 26-28, ACM Press.

Markus, M. L. and Connolly, T. (1990) "Why CSCW applications fail: Problems in the adoption of interdependent work tools." In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, Los Angeles, California, October 7-10, ACM Press.

Root, W. R. (1988) "Design of a multi-media vehicle for social browsing." In *Proceedings of the Conference on Computer-Supported Cooperative Work*, p. 25-38, Portland, Oregon, September 26-28, ACM Press.

Sarin, S. and Greif, I, (1985) "Computer based real-time conferencing systems." *IEEE Computer*, **18**(10), p33-45.

Stefik, M., Bobrow, D.G., Foster, G., Lanning, S. and Tatar, D. (1987) "WYSIWIS revised: Early experiences with multiuser interfaces." *ACM Trans Office Information Systems*, **5**(2), 147–167, April.

Tang, J. C. (1991) "Findings from observational studies of collaborative work." In S. Greenberg (ed.): *Computer Supported Cooperative Work and Groupware*, Academic Press, London. Originally published in *Int J Man Machine Studies*, **34**(2), February.

Tatar, D. G., Foster, G. and Bobrow, D. G. (1991) "Design for conversation: Lessons from Cognoter." In S. Greenberg (ed.): *Computer Supported Cooperative Work and Groupware*, Academic Press, London. Originally published in *Int J Man Machine Studies*, **34**(2), February.

Thimbleby, H. (1980) "Dialogue determination." *Int J Man Machine Studies*, **13**.