Coping with complexity and interference: design issues in multimedia conferencing systems*

M. Angela Sasse
Mark J. Handley
Nermeen M. Ismail

Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK

*This work has been funded by the CAR project of the CEC RACE Programme and Department of Computer Science, UCL

Abstract

The paper recounts the design and usability issues encountered by the authors during a project which built a multimedia conferencing system. These issues tend to arise mainly from two sources: (a) the *complexity* involved in handling the system, and (b) the continual *interference* users experience when interacting with the system and each other. We found that design principles which have been suggested (such as *WYSIWIS* and *seamlessness*) offer only partial solutions to these problems. Generally, complexity and interference experienced by users can be reduced by imposing certain constraints on the way in which users interact with the system and each other. Imposing any of these constraints, however, results in reduced flexibility of user-system interaction and more rigid protocols on user-user interaction, and is bound to make the system less acceptable for some groups of users and some tasks. We therefore suggest that such constraints should be agreed and implemented by the user group, prior to undertaking a collaborative project, to suit user and task characteristics, rather than be imposed on users by including such constraints in the design of the system interface.

1 Introduction

Once the concept of CSCW tools was established, many software developers responded enthusiastically, and quite a number of groupware applications were built and released in the space of a few years. Few of these applications, however, were taken up with the same enthusiasm by their intended users. Grudin (1990) studied several of the failed projects and detected a high level of resistance amongst users against adopting these applications. He attributed this resistance to inadequacies in the design of groupware applications, resulting from the developers' failure to appreciate that technology which aims to support collaborative work groups has to address issues and concerns which are fundamentally different from those encountered in the development of other application software. In Grudin's view, approaches developed in human-computer interaction (HCI) and human factors (HF) were ill-equipped to tackle these problems, since they were based on experiences and observations of individual users interacting with stand-alone systems. Consequently, they focussed on optimising screen designs and interaction styles. Developing effective and successful groupware applications, however, required approaches which provided an in-depth understanding of the collaborative activities which these applications were trying to support.

Following this approach, the developers of Freestyle [1], a multimedia groupware system, dedicated resources to in-depth studies of office work and discussing functionality with prospective users (e.g. Ehrlich, 1987) prior to developing the system. In a recent report on the post-installation phase of Freestyle, Francik et al. (1991) found that the system was not adopted by many users, simply because adoption required a change in organisational work patterns. They suggest that, in addition to studies on the nature of work, such systems need to develop an *adoption strategy* in order to be successful. They predict that this will be particularly relevant for multimedia and desktop conferencing systems which are currently being developed. Such an adoption strategy includes:

- that the system addresses a pressing business problem in the prospective user organisation(s);
- helping users to identify uses of the system in their current work environment; and
- close collaboration during development with a pilot user work group.

When designing a multimedia conferencing system for the automotive industry under the RACE CAR [2] project, the design team followed a rationale which closely resembles an adoption strategy as suggested by Francik et al. (1991):

- the system addresses a pressing business problem in the automotive industry (the need to shorten the design cycle for new models);
- the team of developers held trials and workshops involving users (car designers and productions engineers) from the target client organisations to help them identify uses of multimedia conferencing in their work; and
- the team of developers collaborated with pilot user groups throughout the project.

The result of this adoption strategy was a pilot system whose functionality closely matches the requirements of the intended users. In addition, the semi-formal trials with members of the pilot

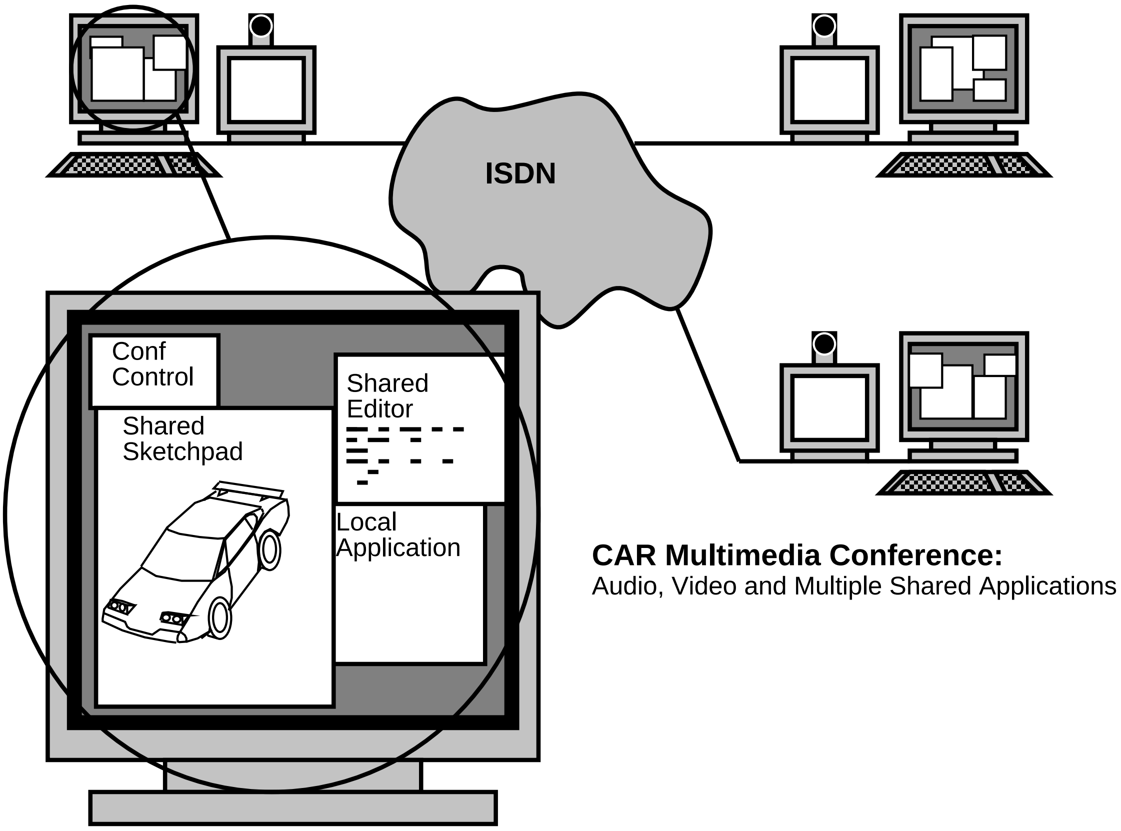
[1] Freestyle is a trademark of Wang Laboratories

[2] CAD/CAM for the Automotive Industry under RACE (Research for Advanced Communications in Europe)

user groups helped the developers to identify usability issues specific to multimedia conferencing systems which could pose an obstacle to adoption of the system. The paper reports explains these usability issues and discusses how they can be resolved.

The CAR Multimedia Conferencing System

The CAR multimedia conferencing system was developed in the Department of Computer Science at University College London, as part of the RACE CAR project. Its goal was to develop a multimedia conferencing system that would help shorten the design cycle for new cars. It was aimed at assisting car designers at different sites to interactively discuss design decisions - something that currently involves much travel and courier-transport of drawings and documents. The idea was that designer could have design sessions without leaving their desks, via their workstation.



The design of the system is based on providing a conferencing platform that can support both applications written specifically for the platform (*collaboration aware* applications), and existing applications written as stand-alone software (*collaboration transparent* applications). As an example of collaborative aware application, a distributed SketchPad Application was developed by one of the CAR partners that has been fully integrated with the conferencing system. CAD packages running under the X windows are an example of collaboration transparent applications that can be included in the conferencing system. This is done by using a shared window system, which is based on Shared-X developed by Digital. The conferencing system also encompasses video and audio communication. Typically, there is one video channel for a conference, which is controlled by the floor holder, while the audio channel is an open one independent on the floor mechanisms. Figure 1 gives an overview of a CAR multimedia conference.

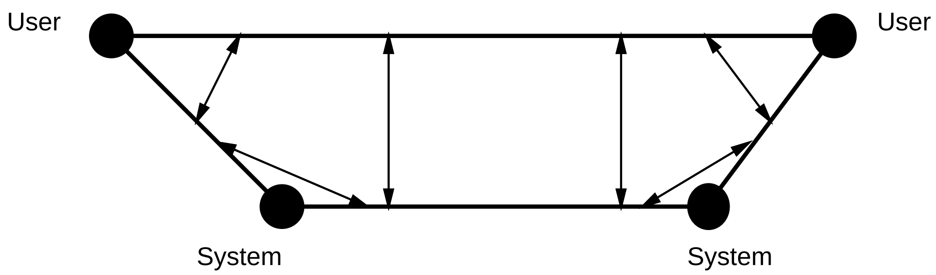
Most of the observations reported in this paper took place during a week-long semi-formal evaluation of the system with a group of 16 users, car designers and production engineers from an automotive design company and two car manufacturers. The users were introduced to the functionality of the system in a two hour training session. They were then asked, two users at a time, to complete a series of interactive design tasks using the system. They were also free to experiment with the system. The sessions were recorded on videotape, and the users were de-briefed for about one hour each following the session. Additional observations were made by members of the project team when using the CAR system for a collaborative writing project (Baydere et al., 1993).

2 Usability problems in multimedia conferencing systems

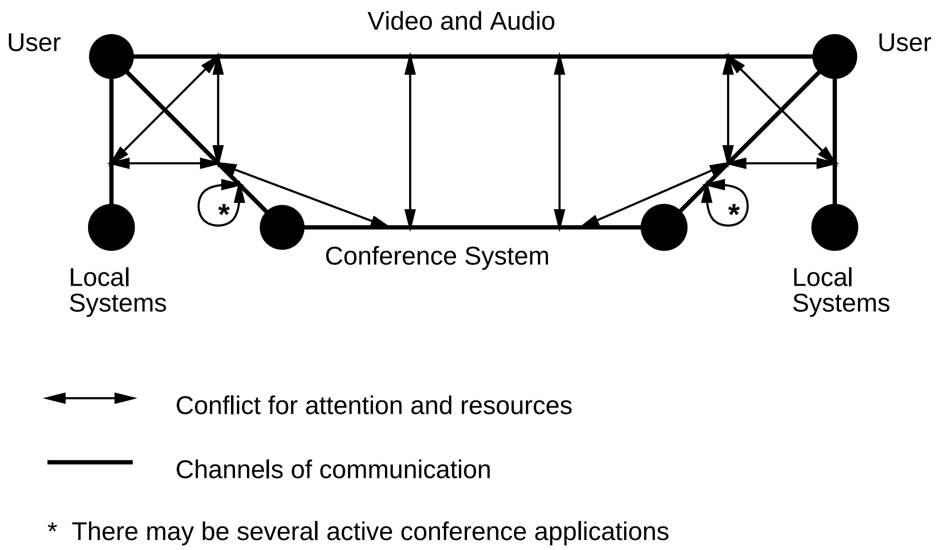
The usability problems observed in the CAR user trials stemmed in the main from two sources. The first is overwhelming *complexity* of this sort of multiuser system: users of multimedia conferencing systems tend to work with more than one application, and use multiple workspaces. In applications such as collaborative drawing tools, more than one layer is generated for each image (for a detailed description of such an application see Ishii & Miyake, 1991). Multimedia conferencing systems offer a wealth of functionality, and much of users' cognitive efforts is spent on determining and locating the appropriate tool or function for the task they are trying to perform. Green (1990) has captured this the idea of cognitive capacity being taken away from the actual task in the concept of *viscosity*. In addition to being highly viscous, conferencing environments tend to generate many notifications and messages, which disrupt users' actions or their planning of actions. These messages may or may not require action to be taken, so users are often left in a state of uncertainty as to whether and how they should respond to these events.

The second source is continual *interference*. Performing a task, such as starting up an application, is often (unintentionally) interrupted by an action (e.g. a question) from another user. Similarly, a discussion between users is often interrupted by a system message or notification (e.g. that another user has joined). In multi-user systems such as CAR, users interact with each other via their systems. This means that their are 3 channels of communication: between the users, between a user and her system, and between the system components. (The idea of taking users and systems as dialogue partners, and distinguishing possible interactions into user-user, system-system, and user-system interaction is taking from a paper by Gaines & Shaw, 1986).

Interfering Channels of Communication (General Model)



Interfering Channels of Communication in the Conferencing System



Which design principles could be employed to avoid these problems? Two principles which have been proposed for CSCW applications are WYSIWIS and seamlessness.

WYSIWIS

WYSIWIS (What You See Is What I See) has been proposed as a principle for displaying shared information CSCW applications. Shared information presented to all users in a system should be presented to all users in the same way (Elwart-Keys et al., 1990). This means that users cannot, for instance, resize or iconise a the window in which shared information is displayed. The main reason is to avoid referencing problems (e.g. "on that second spike from the top"). These could cause distortion of information and subsequent breakdown in communication between users.

But even with strict application of WYSIWIS, distortion and breakdowns still occur. Two examples:

- What You See Is What I See - but does it mean to you what it means to me? Displaying the same information to all users does not guarantee that what is displayed is interpreted in the same way by everyone. Variation in factors such as a user's knowledge and background, vigilance, focus of attention etc. can lead to different interpretation.
- What You See Is Only Part Of The Action. Users looking at shared information share the display being changed by another users, but not the actions which caused the change. They might, for instance, see an error message, but they did not see which button/keys the other user pressed prior to the message.

Seamlessness

Seamlessness was proposed as a design principle by Mark Stefik (in Foster & Tatar, 1988) as a way of overcoming user resistance to groupware applications. The idea is that group software tools (groupware) must merge with individual tools. Whilst not having to learn additional tools certainly cuts cognitive overheads and lowers resistance to some degree, there are doubts whether seamlessness can be achieved or is desirable for all collaborative tasks. Two examples:

- *A "mindshift" might be essential.* Collaboratively working on a task is very different from working on your own task, even when the same software tools are employed. Strategies which might be effective for individual work might be less effective in a group context.
- *Security and privacy concerns.* Users can be expected to be wary of a system which would allow them to accidentally "drift" into a collaborative situation.

Applying WYSIWIS and seamlessness as design principles can solve some problems, but create others in the process.

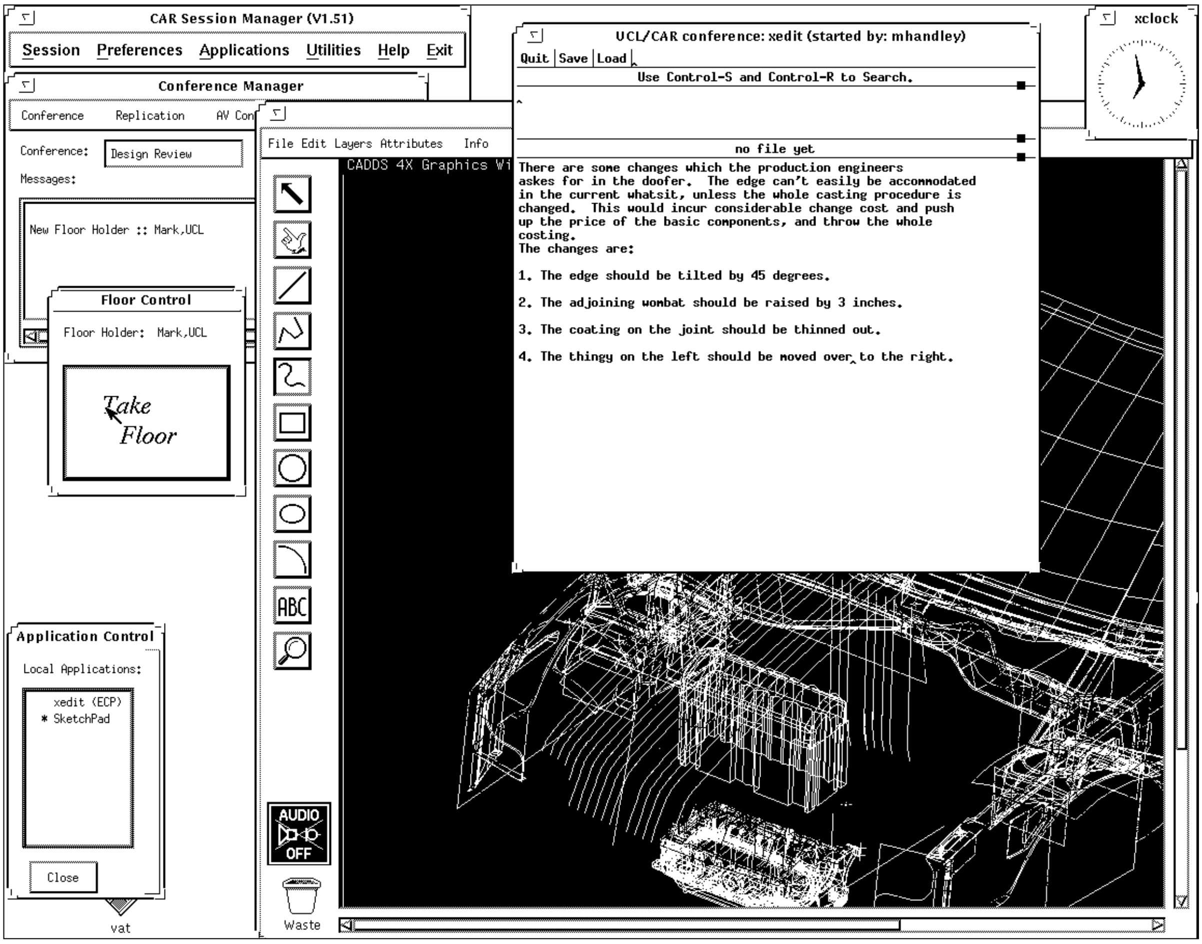
3 Interaction problems in multimedia conferencing systems

In a conventional multi-tasking windows environment (such as SunView or XWindows), a user has many channels of communication with the system using many applications. Several of these applications may be active simultaneously, but typically only one holds the users attention at any time. The user provides input to a single application, and that application produces a reply - all the window system is used for is a simple way to switch applications and/or context. Any system response is a result of the user's interaction with it.

A multimedia conference windows environment also provides many channels of communication with the system, but these channels now also communicate with multiple users. A user's focus of attention is now not only determined by the desired interaction with the system, but also by unexpected events caused by other conferees. Rather than being able to focus on one channel of communication (with the system), the user communicates with other users at the same time. In addition to receiving replies from her own application to known input, the user receives replies to unknown input from other users. This can lead to problems with screen clutter, handling of events, notifications, floor control, and pointers. The problems are described in this section, and available design solutions and their - potentially undesirable - side effects are discussed.

3.1 Clutter

In a single user environment, a user has all the necessary information to perform screen management. In a shared environment, users try to keep all applications visible on screen at once, as they feel that iconising or hiding them will lead to confusion, as outlined below. They rapidly run out of screen space, the display becomes cluttered and information which is needed to manage the interaction with the system successfully (e.g. the conference manager) becomes difficult to locate because windows cover or overlap with other windows (see Figure 3 for an example screen).



Traditional window managers offer three basic methods for to controlling screen clutter in single-user environments:

- Resize windows to fit.
- Iconise windows not currently being used.
- Hide windows not currently being used behind those that are.

In a conferencing windows environment, the display contains windows which can be controlled by other users. These users' actions have to be taken into account before trying to reduce the complexity of the screen. Here conventional screen management tools such as those provided by window managers do not help and can easily complicate the situation further.

Resizing a local copy of a shared window, for instance, can lead to problems because the application is then not strictly WYSIWIS. This is a problem with both collaboration transparent applications and collaboration aware applications, but is much easier to deal with if the application is aware of the existence of the conference. Resizing a window can result in missing some part of another's user drawing. This can occur without users noticing: users assume that they are looking at the same application, but in fact what they see depends on the size of the window.

Iconising and *hiding* windows are essentially two ways to perform the same task - temporarily remove an application from your workspace. Users are reluctant to do either, since if another user revives an unused application, they might not notice. They usually attempt to tile the screen so they can see as much of the shared applications as possible, even if this involves covering up applications such as conference control. Clearly this is inefficient use of screen space, which is a scarce resource. Broadly speaking there are three main strategies for addressing this problem:

- *Notify the user when someone starts to modify a hidden or iconised window.* This inevitably leads to more notification messages (and hence interference) or more implied symbolic meaning (and hence complexity) and more distractions (though they may be very relevant). This may be a good solution for experienced users who can cope with extra feedback, but for inexperienced users there may already be too much going on at once, so they will still avoid hiding shared windows.
- *Let the system take control.* A "clean screen" policy can be enforced in which applications that have not been used for a certain period of time are automatically iconised, and are de-iconised when they become active again. This might result in the user experiencing a certain lack of control. There is also the additional problem that the system cannot tell if the user is only using an application for reference (ie. just looking at it). The latter problem can be solved if the user tells the system they are interested in an application (c.f. OpenLook pushpins), but this requires extra user input for what should be a passive activity.
- *Out of band communication - let the users sort it out verbally.* This can work quite well if only one or two applications are involved and the conference is very small - user holding the floor can get verbal feedback on any problems. This approach becomes increasingly ineffective as conferences become larger.

3.2 Unexpected Events

In a single-user windows environment, events should be determined by the users actions, and thus should for the most part be predictable, or at least self explanatory. Even if the user makes a mistake, the result should explain what that mistake was, but most importantly, the error message is in response to her known input or request.

In a shared windows environment, this is often no longer the case. If a mistake is made, it may be another user that makes it, and so any warnings or messages are viewed out of context. This causes the user to become confused, and wonder what she did wrong. Other conferees manipulating applications cause events that the user does not expect - such as excluding an application from the conference. This leads to a feeling of a "lack of control", especially if one user is less familiar with the system than the other conferees.

In fact this problem does not only apply to errors and error messages - perfectly legitimate requests such as to de-replicate an application you have introduced can cause similar problems. Such events should be accompanied by a notification message saying what has happened, but if you are considering an application when someone else terminates it, this is of little consolation. This problem could be avoided if everyone has to agree to an event, but such rigidity would greatly interfere with the intended purpose of the conference - communication; such administrative tasks should be kept to a minimum if conferencing systems are to become "natural" to use. The simplest way of avoiding problems of this sort is to leave it to a "human protocol"; ie. let the users notify each other verbally (see also Baydere et al., 1993). If this is to be the case, it must be stressed during training that such protocols have to be agreed and followed.

3.3 Notification of Events

In a multimedia conference, events will occur that users should be informed about. If a new conferee joins the conference, all the existing conferees should be notified immediately that this has occurred, for it may affect the way they behave. This is a high priority event, and so the user should be distracted sufficiently from whatever she is doing to take notice of the event. The CAR system does this by sounding an audible warning, and popping-up a new window containing the information.

This sort of warning proves too distracting for users if it is used for lower priority notifications, such as change of floor holder (On the other hand, it will be fairly high priority warning if you were the old floor holder.) The CAR system has an area set aside for these notifications. We experienced situation in the trials where users covered this with applications and then get confused.

Sometimes implicit notification can be used. An example of this is the video communication channel in the CAR system which switches to show the floor holder when the floor changes. However this default can be overridden, and thus confusion can easily be created.

A example of a low priority event is the resizing of a shared window by the floor holder. Here the notification received needs to be ignored until it becomes relevant - ie when the floor holder starts drawing in an area that can't be seen. Clearly, there is a need to provide communication between conference sites of this information, but irrespective of whether this is done using a shared window manager, or aware applications, the presentation of this information to the user presents some problems. In order to lessen the uncertainty of a user and thus her confusion, more information needs to be given, which may result in increasing her confusion. The question is how to present this information in a way that will reduce the complexity, not increase it.

Many asynchronous events are known about by the system, but a user is usually concentrating on one of the shared applications. If the user is not to be confused later, the system should notify the user of the event, but if it does this in an obtrusive way, users become irritated by the notifications as they cause distraction.

Which notification method is used for which event should depend on the user. High priority events should always be intrusive, but lower priority events should depend on the user's level of expertise.

In simple cases it is possible to avoid presenting low-priority information to the users at all. We tried this method recently when a team of experienced users (the authors were among them) used the CAR system to write a paper collaboratively. Though there was some confusion from time to time, the task was accomplished very smoothly. However most of the time there was no more than two simple shared applications, controlled by one active user (floor holder). As conference complexity increases or with less experienced users, this is a less satisfactory solution.

Some users may not wish to spend time to learn how to use the system, or just use it on a one off basis. Such a user may just want to "sit back and watch". In this case she may require active windows to raise themselves and resize themselves in line with the floor holder's, but would not want to be explicitly told of application inclusion or change of floor as the screen is already changing out of her control.

If a user is skilled in the use of the system, she will want to be in control of her environment. Events such as window resizing by the floor holder should be flagged visibly, *but as simply as possible*. For example changing the border colour of the window may imply it has been resized. The environment is already complex, and so complicating it further with unnecessary labeling is undesirable. However merely changing border colour has no visible indication of it's meaning. It may work well at a subconscious level, but only with experienced uses.

In between these skill levels come inexperienced users who use the system occasionally, but for whom it it not yet intuitive. It is these people that will have most difficulty. Simple visible flags are only enough to inform such a user that some unknown event occurred, and so need to be accompanied by an associated icon or label. However the appearance of a label can be distracting in itself - any change in state of the system that is not triggered by the user should be self explanatory. Care should be taken that any notification can be recognised immediately, but ignored until we want to act on it. Simply changing a label will be noticed by the user, but she may not notice exactly what it was that changed, and so wastes time checking.

	Low Priority	High Priority
"Sit back & watch" (Novice)	None No response required	Depends on user No response required
Inexperienced	None No response required	Obtrusive Active response required (eq intrusive window appears)
Intermediate	Intrusive but explained * Passive response	Obtrusive Active response
Expert	Intrusive but symbolic * Passive response	Intrusive Configurable response

* These should be intrusive in as much as the user should notice what has happened, but they should not inferfere with existing tasks,

A classification of notification messages according to user experience

3.4 Floor control

In a conferencing environment where more than one person can interact with the shared applications, floor control mechanisms are needed in order to organise these interactions. These mechanisms can vary from a totally open floor mechanism in which all users can interact with all applications at the same time to a strict mechanism in which only one user can interact with all the applications at any time. “Open floor” gives users more control and flexibility but increases the uncertainty of what is happening and who is doing what. The amount of information the user needs to know in order to reduce this uncertainty is huge and displaying all this information increases the complexity of the system. While a single floor is less flexible as it prevents concurrent activities from taking place, it does not provide the same degree of uncertainty, and accordingly it is not as confusing.

One can think of many other schemes of floor control between these two extremes. In all cases the more people that can hold the floor simultaneously, the more communication channels can be created between users and thus more parallel events can happen. It is a trade off between flexibility (number of concurrent activities) and complexity of use (uncertainty).

Different floor mechanisms can be assigned for different applications. Though this concept can exploit the different natures of different applications, it can complicate the user's model of the system. It is not possible to design a single floor control mechanism that can suit all users and all tasks. It is desirable to give users the ability to tailor the floor control mechanism according to their needs and experience as well as to allow them to switch between different mechanisms easily. A simple floor control mechanism can be used for conferences including inexperienced users. Once more experience is gained another more open (more flexible) mechanism can be used, but it is important that users can find out who is doing what without disrupting the ongoing of the real task.

3.5 Pointers

Enabling users to share applications in a conference is the first step to enable meaningful interactive collaborative work. Still users do need other means of communication in order to help them share these applications on a conceptual level as well as on a visual level. One of these tools is a mechanism that enables users to point to objects of interest in a way everyone else can see.

There are many different policies for designing such a tool. These policies can be categorised along four dimensions.

What to echo?

There are three main possibilities :

- To echo the local pointer of the floor holder on all other users workstations. This policy is very intuitive as the floor holder does not have to do or use anything additional. However, it can be distracting for the other users as the local pointer is used not only for collaborative work but also for other local purposes. The system cannot filter out these local movements from the ones the user intends to communicate.

- To echo the local pointer on other users screens if the user explicitly decided to do so. This method is used by MMconf (Crowley et al, 1990). A user has to click on the middle button of the mouse to cause the pointer to be echoed to the other users. The pointer then changes shape to a big arrow indicating that this is now a shared pointer. A problem with this method is that once the user releases the mouse button his pointer disappears and then cannot be referred to whilst further tasks progress.

- To create a separate shared pointer that can be grabbed by the local pointer, moved and then released. This is the policy used in the CAR system. A user can move the pointer to a certain position leave it and refer to it later.

During trials, users sometimes subconsciously assumed that their pointer was echoed on all other users screens. They tended to use it to point to things while explaining them verbally, even though there was a real shared pointer facility.

How many pointers?

One per conference. Only one user (usually the floor holder) can move the pointer at a time. This tends to tie discussions to the floor mechanism. This is rather unnatural and can be frustrating at times. It interrupts the flow of discussion, as you have to take the floor to reply. It has the advantage that there is just one pointer echoed to each users screen which is simple and unconfusing. Having one pointer that anyone can grab solves some of these problems, but introduces the problem of identifying who has control of the pointer.

One per user. Having a pointer per user gives the users the ability to move their pointers free from any floor mechanism. This gives a greater flexibility and more natural feeling especially during discussions. However having several pointers visible at once can get confusing (too many channels of communication). Pointers need to be identified with users, either by colour coding or by labelling them.

When to echo a pointer?

Continuously. Leaving pointers echoed at all times can cause confusion and distraction especially if there is one pointer per user. It also deprives users from controlling their own screen which is contradictory to what usually happens in stand alone environments. Simplest for inexperienced users as they don't need to do anything.

Let the viewer decide. This can cause problems when a user is moving a pointer that another user has made invisible. However pointing is usually done in conjunction with verbal communication, so users should be able to make the relevant pointer visible in due time. Still this can be a problem with inexperienced users, but if they echo continuously, it allows a good mix between users with different experience levels.

Let the person pointing decide. This is probably the most natural approach, and is the approach taken by CAR. However we still found problems when inexperienced users attempted to point with a local pointer.

As always, the rule is that there is no right answer, so it should be possible to configure the system depending on the users and the task.

Who controls a pointer?

If we only have one pointer per conference it makes sense to tie the pointer to the floor mechanism. This way everyone knows who has control. However this means that continuous changes of floor are required for a discussion.

If we have the pointer allocated on demand, then interactive discussion is facilitated, but at the cost of requiring the users to know about extra state information - who has control.

There are many more issues involved such as what happens when a user moves a pointer across a local window? One strategy adapted by the CAR system is to change the pointer shape on the users workstation to an empty hand whenever the pointer is moving across a local space. Another issue is what to do when the pointer is moving on a shared window that is hidden or iconised on a certain user's workstation. Should the system inform the user that a pointer is out of sight because this window is iconised, small or hidden?

4 Discussion: Design principles for multimedia conferencing

As we have outlined in the previous section, the trade-off between simplicity of use and flexibility of communication between users is a recurrent issue. We can provide a system that is simple to understand, but cumbersome to use, or we can provide a system that is powerful and flexible, but inexperienced users will be overwhelmed with information from many parallel communication channels. Our experience shows that even simple systems can cause users many problems because they are not used to working in this way. Clearly the lesson to be learned is that *system designers must not fix these design decisions, but should allow the users to reconfigure their system depending on their task and their expertise.*

It is desirable to support both collaboration aware applications and collaboration transparent applications. Whichever approach is taken (and to support both is desirable), some degree of information on what the other users have done is required. Strict WYSIWIS is generally too restrictive and the resulting events can cause a user to become confused because her local environment can change (windows resize, raise etc) outside her control. If strict WYSIWIS is not implemented, many events triggered by other conferees must be communicated to a user.

From our observations, we conclude that the following set of design heuristics should be considered:

- If the task is simple (few users, few applications), human protocols (verbal communication) may be sufficient, and less intrusive. If the task is complex, human protocols will not suffice *on their own.*
- A set of options should be provided for notification display - not all users can cope with the same level of complexity.
- The notification set should be consistent - so users can progress from one level to another gracefully as they learn.
- As users become more expert, more abstract notifications can be used. The environment is already complex. *If the user can cope*, reduce the display complexity as far as possible.
- Users will notice that even small changes have occurred on screen, but if the changes are subtle they will spent time trying to figure out what has happened - Make notifications obvious enough to be noticed and recognised so the user can get on with their task.
- Unless a notification is really important, it should not be intrusive (for example windows that pop-up). It is worth considering reserving a small area of screen for notifications that do not

apply to a specific window. Ideally it should not be possible to cover up such an area.

- Only very high priority notifications should require acknowledgement from the user. Users attention may not be constant, so high priority notifications should usually remain until acknowledged.
- Users should only be notified of something that is relevant to them. For example, another user resizing a window is not relevant unless that user is the floor holder.
- The appearance and behaviour of pointers (and related tools) should be chosen by the users so to best support the task in hand. The method of control does not have to be uniform throughout the working group.
- Generally speaking, the larger the working group, the simpler the method of floor control that should be employed.
- If strict WYSIWIS is not implemented, users will not have a constant frame of reference. They should be made aware of this, and a way of synchronising views (within restrictions imposed by the hardware) with other users, and particularly with the floor holder, should be provided.

This last point is part of a wider issue - who controls what you see? Two extremes are possible:

- Centralised : The floor holder controls what the application does and how you see it.
- Distributed : Every user can control the application, and their own view of it. (usually with collaboration aware applications)

Most of the time neither of these extremes is desirable. Often is it meaningless to have more than one input to an application. Usually you wish to be able to control your own workstation environment. If a single compromise has to be chosen, then you allow users control over their view, but with a method of synchronising with other users. However it would be much more desirable to allow users the possibility of imposing their view on other users *when required*. It should be emphasised that the authors think this feature should be used with restraint, as it will tend to lead to a conflict of power, and confusion when contexts change without warning. Whether you allow free-for-all control of an application must depend on the application itself, but becomes increasingly undesirable as conference size and number of shared applications increases.

5 Conclusions

There is no comprehensive set of design principles which would lead to a satisfactory solution of all the problems we have observed for all users and tasks. Recommendations can be made for individual features such as shared pointers and floor control depending on the nature of the task and experience of the user group. At this stage we recommend that the following points should be observed:

- Designers of multimedia conferencing systems should not determine the nature of interaction between users, but should support many possible interaction styles and let the users decide.

- The decisions about style of interaction should made in line with individual user and task characteristics.
- System designers should be aware that some interaction styles should conjunction with set of appropriate human protocols.
- A critical part of system design is choosing a set of defaults suitable for novice users.
- Supported interaction styles should support "graceful" growth of expertise.
- A "shorthand" system should be provided for experienced users not just keyboard accelerators, but also for style of notification.
- A user should be kept aware of all relevant events that can affect her, but this must be done in sympathy with her level of expertise.

6 Acknowledgements

The CAR project at UCL was initiated and is managed by Prof Steve Wilbur, to whom we owe many thanks. We would like to thank all our colleagues on the CAR project, and especially Sue Joyner and Caroline Parker from HUSAT, and users from Ford, IAD and Peugeot who participated in the trials.

This work has been funded by the CAR project of the CEC RACE Programme and the Department of Computer Science, UCL.

References

[1] Sebnem Baydere, Tom Casey, Shaw Chuang, Mark Handley, Nermeen Ismail & Angela Sasse (1993): Multimedia Conferencing for Collaborative Writing: A Case Study. In M. Sharples: *Computer Supported Collaborative Writing.* Springer.

[2] Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick & Raymond Tomlinson (1990): MMConf: An Infrastructure for Building Shared Multimedia Applications. *Proccedings of the Conference on CSCW.* October 7-10, 1990 Los Angeles, CA, pp 329-341.

[3] S.F. Ehrlich (1987): Strategies for encouraging successful adoption of office communication systems. *ACM Transactions of Office Information Systems*, 5, 4 (Oct 1987), 340-357

[4] Gregg Foster and Deborah Tartar (1988): Experiments in computer support for teamwork - Colab (Video). Xerox PARC, 1988.

[5] Ellen Francik, Susan Ehrlich Rudman, Donna Cooper & Stephen Levine (1991): Putting Innovation to Work: Multimedia Communication Systems. *Communications of the ACM*, Vol.34, No.12, 53-63.

[6] Brian R. Gaines & Mildred L. G. Shaw (1986): Foundations of dialog engineering: the development of human-computer interaction. Part II. *International Journal of Man-Machine Studies*, 24, 101-123.

[7] Thomas R. G. Green (1990): The Cognitive Dimension of Viscosity: A Sticky Problem for HCI. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel (Eds.): *Human-Computer Interaction - INTERACT '90*. Proceedings of the IFIPT TC 13 Third International Conference on Human-Computer Interaction. Cambridge, UK, 27-31 August. Elsevier.

[8] Jonathan Grudin (1990): Why Groupware Applications Fail. In Laurel, B. [Ed.]: *The Art of Human Computer Interaction*. Wiley.

[9] Hiroshi Ishii & Naomi Miyake (1991): Toward an Open Shared Workspace: Computer and Video Fusion Approach of Teamworkstation. *Communications of the ACM*, Vol.34, No.12, 37-50.

[10] Mary Elwart-Keys, David Halonen, Marjorie Horton, Robert Kass & Paul Scott (1990): User Interface requirements For Face To Face Groupware. In *Proceedings of CHI, 1990* (Seattle, Washington, April 1-5, 1990), pp. 295-302. ACM.

[11] Chris J. Lauwers & Keith A. Lantz (1990): Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window Systems. In *Proceedings of CHI, 1990* (Seattle, Washington, April 1-5, 1990), pp. 303- 312. ACM.