Revealing Delay in Collaborative Environments

Carl Gutwin[1], Steve Benford[2], Jeff Dyck[1], Mike Fraser[2], Ivan Vaghi[2], and Chris Greenhalgh[2]

[1]Department of Computer Science
University of Saskatchewan
57 Campus Drive, Saskatoon, Canada, S7N 5A9
{carl.gutwin, jeff.dyck}@usask.ca

[2]The Mixed Reality Laboratory,
The University of Nottingham
Nottingham, NG8 1BB, UK
{sdb, mcf, irv, cmg}@cs.nott.ac.uk

ABSTRACT

Delay is an unavoidable reality in collaborative environments. We propose an approach to dealing with delay in which 'decorators' are introduced into the interface. Decorators show the presence, magnitude and effects of delay so that participants can better understand its consequences and adopt their own natural coping strategies. Two experiments with different decorators show that this approach can significantly reduce errors in specific collaborative activities. We conclude that revealing delays is one way in which groupware can benefit from accepting and working with the reality of distributed systems, rather than trying to maintain the illusion of copresent interaction.

Categories & Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—User interfaces; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—CSCW.

General Terms: Design, Experimentation.

Keywords: Collaborative environments, groupware, network delay, latency, jitter, shared workspaces

INTRODUCTION

It is widely recognised that delay is a significant issue for collaborative applications. Previous research has examined the effects of delays in a number of situations, including multimedia transmission [14], conversation [23] and coordination of action in collaborative systems [9, 17, 26]. Research into strategies for dealing with delay, however, has primarily been conducted in the networking and multimedia communities, and has been oriented around schemes that reduce delays at the network level [1, 12].

This paper explores a different and complimentary approach in which the characteristics and effects of delay are revealed to users in order to support them in adopting their own natural coping strategies. This approach has been proposed in previous research [5]. Here we develop it in two ways. First, we introduce different families of delay-related information that might be revealed to users. Second, we report on two experiments to assess the effectiveness of revealing delays in collaborative tasks, the results of which show that the revealing delays can significantly reduce coordination errors.

THE NATURE AND CONSEQUENCES OF DELAY

Delay is an unavoidable fact of life in distributed applications. Delays result from two main sources: the network used to transmit messages, and the processing of those messages at the endpoints. Network delays arise from a combination of transmission delay, switching delay, queuing delay, and retransmission delay, and their magnitude varies greatly according to the type of network involved and changing network conditions. Processing delays result from processing information at the sender, receiver, and servers (if present).

There are two key aspects to delay: latency and jitter. *Latency* is the amount of time between when an event occurs and when it is received by another system in the group. This results in a slower pace of communication, with actions seen sometime after they actually occur. *Jitter* is the variation in latency due to changing network traffic conditions and processing loads. This causes a remote user's actions (e.g. moving a telepointer) to appear jerky, with the result that they may become difficult to predict.

Delays can have severe effects on collaboration – on feedback, coordination, communication, and understanding of the shared situation. Feedback may become delayed so that users cannot relate consequences to previous actions or may believe that their actions have failed when they are in fact delayed. Depending on the underlying control mechanism that is used, it can become difficult to negotiate turn taking: unpredictable communication may hinder social locking protocols, single master locking mechanisms may suffer from reduced availability, and there is more likelihood of inconsistent updates where local replicas are used, with the confusing possibility of having to roll back local actions to some previously agreed state. Delay may also cause users to disagree over the timing or simultaneity of key events. People may experience different orderings of events with implications for causality, such as missed causal links or wrongly inferred dependencies. Finally, applications may suffer from faulty physics, especially

those that rely on the metaphor of a consistent shared space such as Collaborative Virtual Environments (CVEs).

One approach to dealing with delay is to reduce it. This may involve choosing appropriate distribution architectures [20] and consistency mechanisms [6]. Peer-to-peer, client-server and hybrid architectures trade communication speed against simplicity and the ability to enforce common orderings on events, and different consistency mechanisms trade immediate response time against the possibility of inconsistent updates. Collaborative applications may also draw on various multimedia networking techniques including using improved network protocols [25], adapting to meet quality of service (QoS) requirements [2], error correction techniques [1, 19] application layer multicast [18], sending information over multiple network paths [15], load balancing [13] and prioritising communication between users with high mutual awareness [7]. Although HCI research is beginning to explore the potential impact of such techniques [4, 16], the bottom line is that delays and their effects on users are here to stay.

THE APPROACH OF REVEALING DELAY

Given these observations, this paper explores a different approach to dealing with delay in which its presence, magnitude and potential effects are revealed to users so that they can adopt natural coping strategies. This approach is not an alternative to reducing delay, but rather a method for mitigating the effects of delay on the user, and can be combined with the strategies described above.

This approach arose from previous empirical work that explored the effects of delay on users in a CVE [27]. Pairs of users were asked to play a simple tennis-like ball game over a simulated wide area network connection that was systematically subjected to increasing delay. Analysis of video recordings supported by semi-structured interviews showed that some users attributed their interactional difficulties to delay, and that some of these adopted natural coping strategies such as predicting the trajectory of the ball and moving to intercept it at an estimated future position, playing from the back of the court in order to create time to judge the trajectory of the ball, and deliberately trying to slow the game down by bouncing the ball off of walls in order to buy yet more time.

This experiment inspired a new approach to dealing with delay – making users more aware of its presence and characteristics so as to further encourage the adoption of natural coping strategies. Philosophically, this approach treats delay as a natural feature of networked media and considers its effects to be 'delay induced phenomena' rather than problems to be swept under the carpet (an approach that has its roots in a broader discussion of breaking down the transparency of distributed systems in order to better support cooperative work [22]). In short, rather than slavishly following the metaphor of copresent physical space, distributed shared spaces should be treated as their own medium with their own defining characteristics.

DELAY DECORATORS

Although the idea of revealing delay-induced phenomena is straightforward, the practice is not, due to the complex nature of delays and their various potential effects on collaboration. This paper therefore undertakes a deeper exploration of the nature of delay induced phenomena and the ways in which they might be revealed to users. Our main mechanism for revealing delay is what we call a *decorator*. This is a visual ornament that is added to the representation of an object in the user interface in order to enhance a user's understanding of an associated delay-induced phenomenon.

As we shall discuss later, there are potentially many different kinds of decorator to deal with the varied consequences of delay for different applications. For the time being however, we focus on the two specific families of decorators that have provided the basis for our early experimental work: *magnitude of delay* decorators and *past and future state continuum* decorators.

Magnitude of delay decorators

Our first family of decorators communicates the most basic underlying information about delay – its presence and magnitude. We introduce four categories of magnitude of delay decorators: roundtrip time, jitter, one-way temporal distance, and third-party delay decorators. We will use the network scenario shown in Figure 1 as the context for our descriptions of these various decorators.

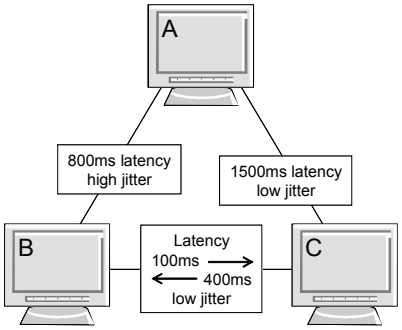

Figure 1. Networked computers running a shared environment with different delays between them.

Roundtrip abstract decorator. This is the simplest example of a decorator and shows the overall round trip time for communication with a remote object (that is, the total delay involved in sending a message and receiving an immediate response). This allows a user to reason about the minimum time that they can expect to wait before observing the effects of acting on an object or, in social situations, the minimum time before they could reasonably expect any response from a remote user. Of course, this time might be longer in practice if the remote object has to process the message. Referring to Figure 1, the roundtrip time between A and B is 1600 milliseconds (ms), between B and C is 500 ms, and between A and C is 3000 ms. As well as representing the magnitude of the delay.

There are many potential ways in which this information might be visualized depending upon the nature of the application and the experience of the users, for example using numbers, translucence or color. One notable possibility is to employ rhythmic animation, for example introducing objects that pulse or oscillate.

Jitter decorator. This shows the variation in delay over time rather than the raw magnitude of delay, as this can be an important factor in predicting the movements of objects [9]. As part of our first experiment (see below), we have implemented an example jitter decorator in which the color of an object (a telepointer) changes according to the amount of time elapsed between receiving position updates.

Temporal distance decorator. Figure 1 reminds us that the delay between two objects need not be symmetrical, for example due to differences between downlink and uplink bandwidth, or due to messages in each direction taking a different route through the underlying network. In some circumstances it may be important to show the delay associated with a given direction. This decorator therefore changes its state to reveal the magnitude of the delay in one direction, from the local user to a remote object or vice versa. It might be necessary to introduce two decorators (one for each direction) or to choose which direction takes precedence in terms of whether it is more important that the user understand how long it takes for the objects' updates to reach them or the other way around. For example, users who are passively accessing remote video and audio will be interested in the delay in the direction from the source object to their local machine.

Third-party delay decorator. Returning to Figure 1, it may sometimes be important for the user to be able to reason about the communication delay between two objects that are both remote. For example, in our network scenario, perhaps the user B is watching two remote users (or objects) A and C. Understanding the interactions between A and C may be helped by knowledge that is a roundtrip communication delay of 3000 ms between them.

Third-party delay decorators provide an inter-subjective view of the delay between two objects that are both remote from the local observer. Given that there are now three locations involved, there are several possibilities as to what might be revealed. A user might be made aware of how long it takes one object's messages to reach the other or conversely, might see the round trip time between them. Alternatively, the user might be given a sense of how long it could be before *they* will see the effects on one object's communication with another, which needs to take account of the delays between the two objects and also to the observer. These kinds of decorations could take the forms described above, but with the addition of a clear visual connection between the remote objects to show which delay values are being represented. Figure 2 shows a design in which an animated pulse travels back and forth along a connecting line between the two objects.

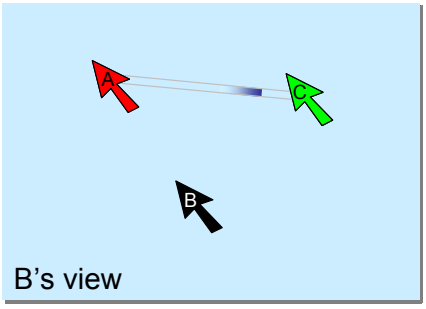

Figure 2. Third-party delay decorator to show the user B the delay between the remote objects A and C.

Past and future state continuum decorators

Our second family of decorators shows the state of artefacts as they were in the past or might be in the future so as to help a user predict how to interact with them.

Past state decorators. These show how an object appeared in the past, especially its past positions. Possible designs include telepointer trails for shared 2D workspaces (see Figure 3) or the animated shadow avatars described in [8], that follow their users around a virtual world and replay their actions from the recent past.

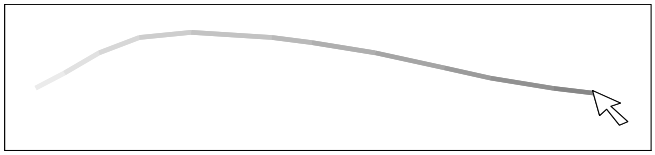

Figure 3. A telepointer trail as a past state decorator

Although past state decorators suit moving objects, they can also be used for other purposes. For example, they might help observers understand the timing of speech – the delay between when an utterance is made and when it is heard – by showing a visual trail of objects to indicate its progress between a speaker and a listener.

Future state decorators. These show the predicted future states of objects, derived from the histories of their past states, knowledge of constraints on their movement (e.g., do they have a maximum speed) and information about current delays. The intention behind future state decorators is to help users predict the potential course of events and so plan their activities in advance, for example moving to a point that correctly anticipates the arrival of an object. These decorators are at best reasonable guesses as to the likely state of an object, and so might try to show their equivocal nature through representations that suggest uncertainty, such as translucent, wire-frame or outline representations of a future position or trail. For example, a user's telepointer might be surrounded by a highlighted region that suggests where it is likely to be now (an example that we implement and test in our second experiment below).

It is also possible to combine past and future state decorators. For example, Figure 4 sketches a design for a decorated telepointer that shows past (solid trail), present

(solid telepointer) and possible future (dashed trail and faded telepointer) states.

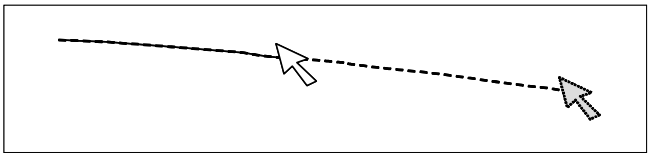

Figure 4. A decorated telepointer shows both past and possible future states

Having introduced two initial families of decorators in some detail, we now turn to the question of whether these techniques actually assist people in carrying out collaborative tasks in delayed conditions.

EMPIRICAL INVESTIGATIONS OF REVEALING DELAY

We carried out two experiments to determine whether decorators help people adapt to delay. The first experiment studied a magnitude-of-delay decorator in situations where jitter is a problem, and the second experiment looked at a future state decorator in a latency scenario.

Study one: a jitter magnitude-of-delay decorator

In a stream of messages such as those seen for a moving telepointer, jitter disrupts smooth motion. When the stream experiences jitter, the telepointer appears to freeze during periods when updates are delayed in transit, and then jumps forwards following updates that are less delayed. Previous work has shown that jitter causes problems when people rely on smooth motion to interpret and predict another person's movements [9]. In these situations, jitter makes it difficult for people to anticipate the pointer's motion. This is especially problematic when the viewer sees the telepointer stop as it is difficult to determine whether the stop is due to a jitter freeze, or whether the other person has really stopped moving.

We carried out an experiment to determine the effects of a jitter decorator on this prediction problem.

Method

Sixteen users who were regular users of basic networked applications were recruited from a local university. Half of the participants had experienced jitter from playing online games. The participants were asked to carry out a number of telepointer prediction tasks in a custom-built application (see Figure 5).

The study system presented a sequence of pre-recorded telepointer motions, and participants were asked to predict where the telepointer would stop. In different experimental conditions, different amounts of jitter were introduced into the playback. For each trial, the user's task was to predict which grid square the telepointer was moving to, and click their mouse cursor on that square. We asked people to try and aggressively minimize their prediction time while still trying to avoid errors.

Jitter was introduced into the telepointer motion by randomly pausing the playback at a particular telepointer message (at a frequency of 5%, and for a length of time determined by the experimental condition). When a freeze occurred, the cursor would halt until the jitter period passed, and then the playback – and hence the cursor – would jump forward to its correct location and continue moving (note that freeze times were not allowed to lengthen the overall time of the telepointer's motion).

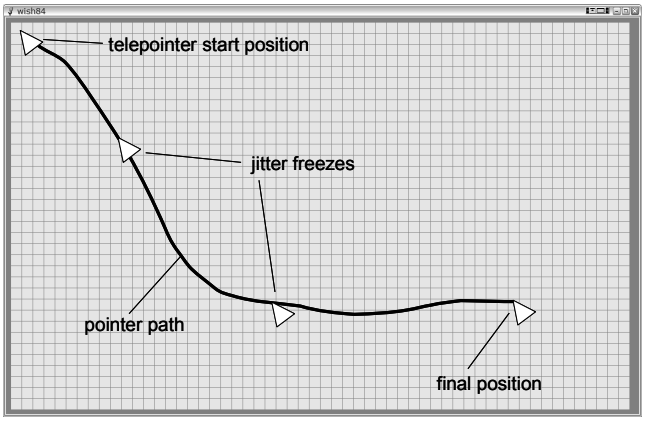

Figure 5. Jitter study system, illustrating telepointer motion in a jittery network condition

The study compared prediction performance with no decorator (the normal groupware situation) to a fading cursor (a jitter decorator from the magnitude of delay decorator family).

The fading cursor decorator

This decorator changes the colour of the telepointer based on the time since a position update has been received (see Figure 6). Assuming that the sender system is providing regular position updates, the receiver can easily calculate how 'stale' the current telepointer position is and colour it accordingly. This technique could reduce errors by differentiating between a telepointer that has really stopped (still white) and a freeze due to jitter (a rapid fade to black).

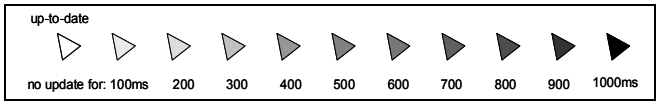

Figure 6. Fading cursor effects for different jitter delays

Study procedure and design

Participants carried out several practice trials both with and without jitter, and then completed 20 test trials with each jitter amount and each decorator. The study used an order-balanced 4x2 within-subjects factorial design. The factors included Decorator type (None or Fade) and Jitter amount (0ms, 800ms, 1100ms, and 1400ms). These amounts were chosen because previous studies showed that prediction errors are a significant problem in this range [9]; although these amounts are relatively high, they are within what we have observed with Internet-based groupware. The study

system collected two types of data: completion time (from the start of the telepointer's motion until the user's correct click), and errors (the number of incorrect clicks).

Results

There were negative effects on performance as jitter increased, regardless of decorator (see Figure 7). However, the fading cursor allowed people to predict telepointer movements significantly more quickly and significantly more accurately than with no decorator. ANOVA showed main effects of both completion time ($F_{1,15}$=6.93, p<0.005) and error rate ($F_{1,15}$=11.56, p<0.001). The actual completion time difference was very small (about 20ms per trial), but the improvement in error rate was substantial: the fading cursor reduced errors from more than one in three trials to about one in five (see Figure 7). We checked whether prior experience with delays led to any differences in these results, but no significant differences were found.

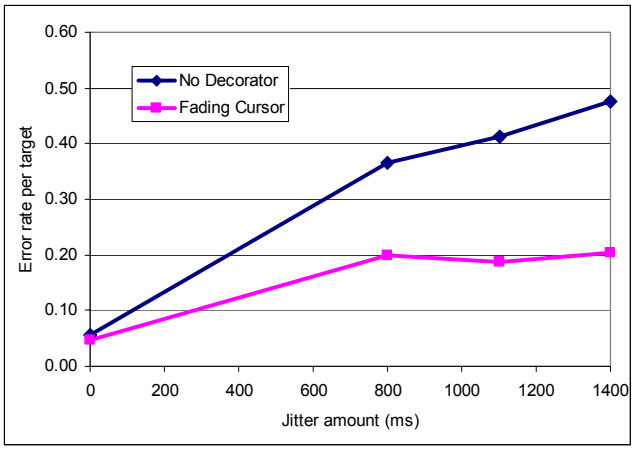

Figure 7. Mean error rates for all jitter amounts.

For prediction of telepointer movement in jittery conditions, the fading-cursor decorator allowed people to better understand the state of the distributed system and adjust their behaviour accordingly.

Study two: latency and future state decorators

Latency causes well-known coordination problems when two people attempt to obtain a single shared resource, whether a graphical object in a workspace or a free space in a telephone conversation. Both parties believe that they are 'next in line' for the resource, and both take the resource (or at least their local copy of it). After the delay period, however, it becomes clear that they have both taken the same resource, and a conflict occurs.

We carried out an experiment to determine whether a future state decorator (from the past and future state continuum family) could reduce this coordination problem in a shared visual workspace.

Method

Twelve pairs of students were recruited from a local university. Again, about half of the pairs had experienced latency through playing on-line games. These pairs were asked to carry out tasks that involved shared access to visual objects.

The study was carried out using a custom-built distributed groupware application that allowed different amounts of latency to be applied to the message streams between the two participants. In the system, participants were asked to drag blocks from a central stack and deposit them in a drop zone, one for each participant, moving around a set of fixed obstacles (see Figure 8). The blocks could be grabbed by either user, and the system recorded instances where one person grabbed a block that had already been taken by the other. The groups were told to minimise the number of these 'double grabs.'

Participants were represented on their partner's view of the workspace with a telepointer, which appeared either as a normal telepointer or as a pointer with a 'halo' decorator.

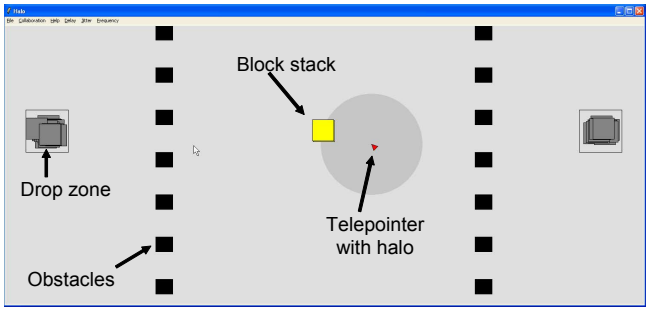

Figure 8. Latency study system (halo effect has been darkened for clearer printing).

The halo technique

The future state decorator used in the system is one of the simplest possible: it shows the potential future location of the telepointer, based on the current speed of the pointer and the current latency in the network. The 'halo' used to represent this area of potential movement is drawn on the screen as a light-grey circle. The size of the halo changes with the speed of the pointer; slow movement will result in a small halo, fast movement (or large latency) will result in a large halo. The intent of the technique is to indicate when a pointer could potentially be in range of the central block stack, and hopefully to help people reduce conflicts in grabbing blocks from the stack.

Study procedure and design

Participants carried out several practice trials with different amounts of latency, and then completed 50 test trials (i.e. 50 blocks dragged by the group to the drop zones) with each latency amount and each decorator. The study used a balanced 5x2 within-subjects factorial design. The factors were Decorator type (None or Halo) and Latency amount (0ms, 200ms, 400ms, 600ms, 800ms). Again, these values were chosen because a previous study had showed that latency was a problem for this range of delays [9]. The study system recorded completion time per condition, and the number of errors (where one person clicked on a block already held by another person).

Results

Again, increasing latency above 200ms led to increasing error rates in the system; however, as with the jitter study, performance with the halo decorator was significantly better than with no decorator. ANOVA was used to compare error rates per block of 50 trials. A significant main effect of Decorator type was found for error rate ($F_{1,11}=5.98$, $p<0.05$). Performance with the halo decorator was on average about 25% better than with no decorator. For example, at 400ms latency, error rates decreased from about one in 11 trials to about one in 15; at 800ms latency, they decreased from one in six to one in 10. No effects of decorator type on completion time were found ($F_{1,11}=0.91$, $p=0.36$). In addition, no significant effect of prior experience was found.

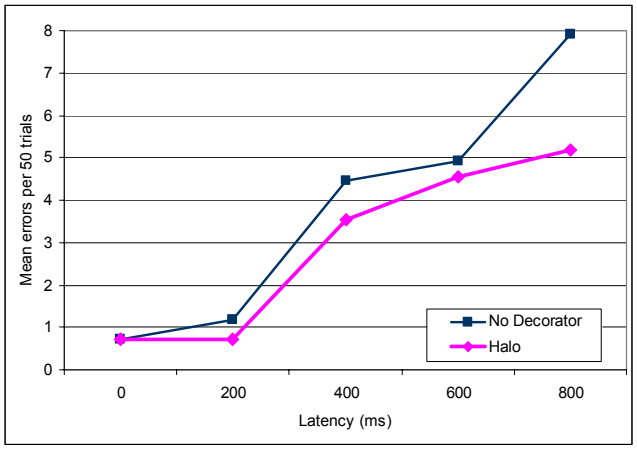

Figure 9. Mean error rates at all latency amounts.

The latency study shows that even with a simple (and imperfectly-estimated) future state decorator, coordination can be significantly improved.

DISCUSSION

Our two experiments provide specific examples of where the addition of delay decorators to an interface helps users cope with the effects of delay. More generally, we now have multiple pieces of evidence to suggest that the overall approach of revealing the presence of delay in the interface can be effective. In the next paragraphs, we step back and consider this evidence with respect to three issues: why the decorators worked, whether they will be successful in real-world tasks, and the potential problem of distraction.

Why the decorators worked

Both of the decorators in our studies worked because they provided answers to questions that were important to the task. For the prediction study, the fading cursor answered the question "has the telepointer really stopped?" For the coordination study, the halo answered the question "is the other person taking a block?" Although these are very simple questions, they cannot be answered in a standard collaborative environment (unless the delay is negligible).

It is important to note, however, that the decorators are not over-fitted to these tasks – they simply indicate a delay value, which can be the answer to other questions as well, in other situations. For example, these decorators can also help people disambiguate deictic references in situations where the voice channel is less delayed: the fading cursor by indicating whether the object in front of the cursor is the real referent, and the halo decorator by constraining the number of objects that the speaker could currently be pointing at.

Generalising the results

There are two factors that govern the degree to which these results will generalize to real-world collaborative situations: the type of task being performed, and the temporal granularity of the interaction.

First, the tasks used in our studies were not realistic group activities. However, they were constructed from real task components: predicting another person's movement, and coordinating access to a shared object are elements of group work that appear in many different activities. They are evident in any situation where people undertake real-time collaboration, where the environment contains shared artifacts, and where people are represented by embodiments. Therefore these decorators will have an effect whenever the task component occurs. In cases where these task components are frequent, we expect the decorators to make a substantial difference to the overall activity.

The second factor is the granularity of interaction. It is clear that if the 'turns' that make up an interactive task happen on a time scale that is much larger than the amount of delay, then the delay is unlikely to cause a problem for that interaction. For example, instant messaging happens (with some exceptions) on a timescale of a few seconds for each turn – therefore, delays of significantly less than that amount will not typically cause a problem, and would not require delay decorators. This essentially states that delay-induced phenomena should be defined in terms of the user's perception of them: if no phenomena are apparent to the users, then there is no need to reveal the delays that are present. The typical delays present on the Internet, therefore, suggest that delay decorators are applicable primarily to closely-coupled real-time work, such as games, design sessions, code reviews, or discussing shared data.

The potential for distraction

There are of course some potential drawbacks to using decorators, including the additional computation and rendering that they require, the difficulty of obtaining information about delay from the system infrastructure, and the potential for distraction, which is the focus of this discussion. Decorators add additional visual information to the collaborative environment, and this may distract or annoy participants. While this was not the case in our experiments where the decorators were always useful to the

task, distraction could become more of a problem when users are engaged in real-world tasks or are interacting with many more objects. Ultimately, it is designers who will have to determine the appropriate trade-off between the potential value and distraction of decorators. However, we do offer a few suggestions as to how to approach this decision. First, it may be important to decorate only selected objects. One approach is to decorate those objects upon which the user is focused or with which they are likely to interact. This might exploit fish-eye techniques [24], proximity or other expressions of focus or interest [7], or might build on predictive-locking techniques where system locks are allocated on the basis of predicted user interactions [21]. A second approach might be to introduce decorators when there is a significant change in delay conditions. Previous experiments that did not involve decorators revealed that users could adapt to delay providing that they were aware of its presence and magnitude [5]. It may be that users who are familiar with the current level of delay won't require decorators, but that a change in conditions will require an explicit decoration until the user has adapted or conditions have returned to their usual state. Alternatively, decorator-like techniques may be used outside the primary interaction space to support a more peripheral or occasional awareness of general delay characteristics.

FUTURE WORK: OTHER FAMILIES OF DECORATORS

So far we have introduced and tested two families of delay decorators. We plan to continue studying these families, adding our existing decorators to realistic collaborative systems, and implementing new designs. However, there are other delay-related phenomena that might also usefully be exposed to users through further families of decorators. Two that we plan to investigate in the future are the double feedback and causality preservation families of decorators which are intended to help users cope with the effects of delay on underlying system consistency mechanisms.

Double feedback decorators

Many distributed systems adopt an approach to consistency where each user accesses a local replica copy of an object while the true state of the object is maintained by a remote master copy. Building on the principle of double level feedback [3], we introduce two decorators to help users understand the relationships between master and replica copies, depending on the underlying concurrency control mechanism in use.

Last true state decorators are designed for systems with optimistic concurrency control. These systems provide immediate local feedback to the user based on the state of their local replica. However, the true state of the object represented by the master will lag behind this, and may even ultimately conflict with it (e.g., if another user is already updating the object). A last true state decorator enhances the existing local view of the replica with some further information about the last confirmed state of the master. This is comparable to the clone objects used with the CIAO system's optimistic concurrency control mechanism [26]. A last true state decorator might take the form of a ghostly representation of the master object that is connected to the local replica's representation by a rubber band. As the object is manipulated, the user sees a solid representation move immediately, followed by its ghostly shadow sometime later as the master is updated.

Instant feedback decorators are designed for systems with conservative consistency policies. These already show the last agreed representation according to the master. However, interaction with this may be subject to a visible lag which may potentially confuse the user (have they managed to grab the object or not?). This decorator enhances the existing representation with an additional portrayal of the immediate local action (e.g., as a ghostlike object that this time precedes the true representation and is connected to it).

Causality preservation decorators

Causality preservation decorators are concerned with the effects of delays on the apparent causality of actions. This is important when causally related events may be seen in the wrong order or with an increased separation in time.

Explicit event decorators emphasizes that a particular event has happened that may otherwise have been missed. Systems that employ optimistic concurrency control sometimes apply corrections to represented positions as conflicts become apparent and are resolved, with the result that users may never see key events such as collisions. An explicit event decorator generates a new object to indicate that a key event has happened in case it was missed.

Correction decorators take this idea a step further by explicitly showing corrections. For example, they might show the corrected path of an object alongside the original path so that the user can understand the correction that has been applied.

CONCLUSIONS

We have argued for an approach of revealing the characteristics of delay to users of collaborative applications in order to support their own natural coping strategies. This is achieved though the introduction of delay decorators – interface objects that visually show the presence, magnitude and even consequences of delay. We have described two experiments to assess the effectiveness of different kinds of delay decorator. In the first, a fading cursor trail (a jitter decorator) was used to convey the nature of jitter in a task in which users had to predict the stopping point of a second cursor. The presence of the decorator enabled them to make more accurate predictions. In the second, a halo around a cursor (a future state decorator) was used to help the user coordinate the grasping of objects with others. Again, the presence of the decorator led to a significant reduction in errors.

We conclude that revealing delays to users is one way in which groupware can benefit from accepting and working with the reality of distributed systems, rather than trying to maintain the illusion of co-present interaction. The key issue that must be addressed when applying this work to real collaborative environments is that the decorators be chosen with appropriate consideration of the task, the network conditions, and the potential for distraction.

REFERENCES

1. Bolot, J-C, Fosse-Parisis, S., Towsley, D. (1999) 'Adaptive FEC-Based error control for Internet Telephony', Proc. Infocom '99, New York, NY.
2. Campbell, A. and Coulson, G. (1997) 'QOS Adaptive Transports: Delivering Scalable Media to the Desk Top', IEEE Network.
3. Dix, A. (1994) 'Que Sera Sera – The Problem of the Future Perfect in Open and Cooperative Systems', Proc. HCI'94, 397-408, Cambridge University Press.
4. Dyck, J., and Gutwin, C. (2002) 'Improving Groupware Performance in Lossy Networks with Adaptive Forward Error Correction', Technical Report HCI-TR-2002-01
5. Fraser, M., Glover, T., Vaghi, I., Benford, S., Greenhalgh, C., Hindmarsh, J. and Heath, C. (2000) 'Revealing the Reality of Collaborative Virtual Reality', Proc. CVE 2000, 29-37, ACM.
6. Greenberg, S., and Marwood, D. (1994) 'Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface', Proc. CSCW'94, 207-217, ACM.
7. Greenhalgh, C, Benford, S. and Reynard, G (1999) 'A QoS Architecture for Collaborative Virtual Environments', Proc. Multimedia'99, 121-130, ACM.
8. Greenhalgh, C., Purbrick, J., Benford, S., Craven, M., Drozd, A. and Taylor, I. (2000). 'Temporal links: recording and replaying virtual environments', Proc. Multimedia 2000, 30-37, ACM.
9. Gutwin, C. (2001) 'Effects of Network Delay on Group Work in Shared Workspaces', Proc. ECSCW 2001
10. Gutwin, C., and Penner, R. (2002) 'Improving Interpretation of Remote Gestures with Telepointer Traces', Proc. CSCW 2002
11. Hindmarsh, J., Fraser, M., Heath, C., Benford, S. and Greenhalgh, C. (2000) 'Object-Focused Interaction in Collaborative Virtual Environments', in ACM ToCHI, 7(4), 477-509, ACM.
12. Hsu, C-Y., Ortega, A. and Khanshari, M. (1997) 'Rate control for robust video transmission over wireless channels', Visual Communications and Image Processing '97, 3024(120), SPIE.
13. Kameda, H., Li, J., Kim, C., Zhang, Y. (1997) Optimal Load Balancing in Distributed Computer Systems, Springer-Verlag London Ltd, 1997, Pages 212-223.
14. Karlsson, G. (1996) 'Asynchronous Transfer of Video', IEEE Communications, August 1996, pp.118-126.
15. Liang, Y., Steinbach, E., and Girod, B. (2001) 'Multi-stream Voice Transmission over the Internet Using Path Diversity', Proc. ACM Multimedia 2001.
16. Litiu, R., and Prakash, A. (2000) 'Developing Adaptive Groupware Applications Using a Mobile Component Framework', Proc. CSCW 2000.
17. Park, K. and Kenyon, R. (1999) 'Effects of Network Characteristics on Human Performance in the Collaborative Virtual Environment', Proc. of IEEE Virtual Reality '99, 104-111, IEEE.
18. Pendarakis, D., Shi, S., Verma, D., Waldvogel, M. (2001) 'ALMI: An Application Level Multicast Infrastructure', Proc. USITS '01.
19. Perkins, C., Hodson, O., and Hardman, V. (1998) 'A Survey of Packet Loss Recovery Techniques for Streaming Audio', IEEE Network, Sept/Oct 1998.
20. Phillips, W.G. (1999). 'Architectures for Synchronous Groupware', Technical Report 1999-425. Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada.
21. Roberts, D., Sharkey, P and Sandoz, P, A Real-time Predictive Architecture for Distributed Virtual Reality, Proc 1st ACM SIGGRAPH Workshop on Simulation & Interaction in Virtual Environments, Des Moines, Iowa, 179-288, July 1995, ACM Press.
22. Rodden, T and Blair, G. (1991) 'CSCW and Distributed Systems: The Problem of Control', Proc. ECSCW'91, 49-64, Kluwer.
23. Ruhleder, K. and Jordan, B. (2001) 'Co-Constructing Non-Mutual Realities: Delay-Generated Trouble in Distributed Interaction', Journal of CSCW, 10(1), 113-138, Kluwer.
24. Sarkar, M., and Brown, M. (1992) 'Graphical Fisheye Views of Graphs', Proc. CHI 1992, 83-91, ACM
25. Schulzrinne, A., Casner, S. (1993) 'RTP: A Transport Protocol for REAL-Time Applications', Internet Engineering Task Force, Internet Draft.
26. Un-Jxe Sung, Jae-Heon Yang & Kwang-Yun Wohn, 'Concurrency Control in CIAO', Proc 1999 IEEE Virtual Reality Conference, VR '99, 22-28, 13-17 March 1999, Houston, Texas, USA, IEEE.F
27. Vaghi, I., Greenhalgh, C., and Benford, S. (1999) 'Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments', Proc. VRST'99, 42-49, ACM.