19.1 INTRODUCTION

Much of the discussion in this book concerns a single user with a computer. However, in Chapter 14 we met computer-supported cooperative work (CSCW), which is about groups of users – how to design systems to support their work *as a group* and how to understand the effect of technology on their work patterns. Both HCI and CSCW draw on knowledge from a wide range of disciplines, but whereas the principal axis in HCI is psychology–computing, the equivalent axis in CSCW is sociology–computing. If we allow the *human* in HCI to be plural, we can regard CSCW to be within the general sphere of HCI, which is why it is present in this book and in conferences and journals on HCI. However, CSCW is now a field in itself, and would require a book of its own to do it justice. We looked at models of collaboration in Chapter 14; here we turn our attention to the technology that supports group working.

One major area within CSCW is the provision of computer systems to support group working. These products are often called *groupware*. As the reader may not have come across any but the most common of such systems, this chapter describes the range of such groupware systems and the associated architectural and implementation issues. In addition, we shall discuss frameworks that classify groupware systems and that describe how they fit into the wider area of cooperative working. Many of these applications are research systems; however, the number of commercial systems with a groupware component has risen significantly over recent years.

Any computer system, whether or not it is specifically groupware, will have an effect on the work-groups within an organization and upon the organization as a whole. It is not possible to cover the full sociological and anthropological background that is being brought to bear on these issues, but it is important that the reader is aware of these knock-on effects of even individual computer systems. In Chapter 14, we described some of the theory of human communication and the group and organizational factors that influence the design of groupware.

19.2 GROUPWARE SYSTEMS

In this chapter, Sections 19.3–19.5 describe a range of groupware systems, from email and video conferencing to shared editors and co-authoring systems. Section 19.6 uses several frameworks that help us to analyze groupware systems, and which will give some structure to the issues that arise during the previous three sections. Section 19.7 discusses some of the implementation problems facing a groupware developer.

Groupware can be classified in several ways. One of these is by *where* and *when* the participants are performing the cooperative work. This is summarized in a *time/space matrix*. Another classification is by the *function* of the system, for example meeting

	Same place	Different place
Same time	Face-to-face conversation	Telephone
Different time	Sticky note	Letter

Figure 19.1 Time/space matrix

support or group authoring. The subsections on specific areas of groupware follow this functional classification. However, there are at least 10 such categories here and the list could extend as fast as one could think up new application areas. To give a broader categorization for the sections, we follow a framework that classifies groupware by the aspect of cooperative work which it supports.

The rest of this section describes briefly the time/space matrix and cooperative work framework, both in terms of non-computerized 'real-world' examples. We will return to these and other frameworks in Section 19.6, once the reader is familiar with the range of groupware systems currently available. As with analysis techniques for single-user systems, these frameworks can help to structure the design of new systems, or to suggest possible application areas.

The time/space matrix is a very useful shorthand to refer to the particular circumstances a groupware system aims to address. Basically, we look at the participants and ask whether they are in the same place or not, and whether they are operating at the same time or not. Figure 19.1 shows how various non-computer communication technologies fit into the time/space matrix.

The axes are given different names by different authors. The space dimension is also called the geographical dimension and is divided into *co-located* (same place) and *remote* (different place). Many of the earliest groupware systems were aimed at overcoming the barriers of distance, for example email and video conferencing. More recently, systems have arisen which aim to augment face-to-face meetings and other co-located cooperation.

The time axis is often divided into *synchronous* and *asynchronous* systems; so we would refer to a telephone as a synchronous remote communication mechanism, whereas sticky notes are asynchronous co-located. These terms are used heavily in the CSCW literature; often the intended geographical location is obvious from context, or the application domain, and thus synchronous/asynchronous becomes the principal distinction. However, there are some problems with this simple

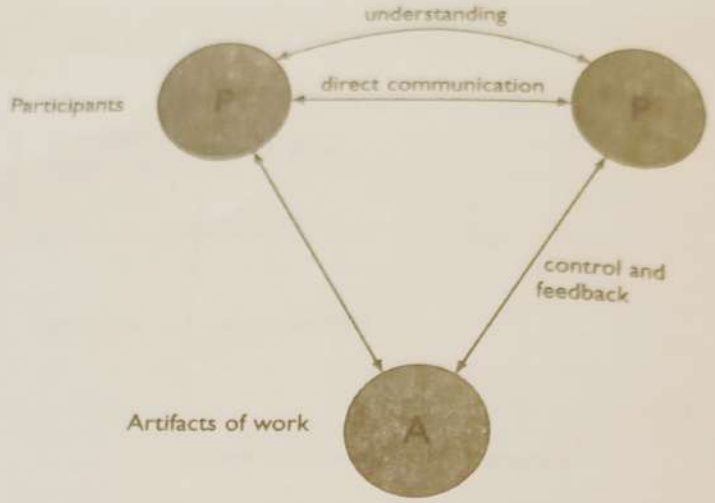


Figure 19.2 Cooperative work framework

distinction, which we will return to in Section 19.6.1. As we discuss each groupware system, it will be placed in its appropriate time/space category.

The framework used to organize Sections 19.3–19.5 is based on the entities involved in cooperative work, that is the participants themselves and the things upon which they work. Figure 19.2 summarizes these relationships. Implicit in the term 'cooperative work' is that there are two or more participants. These are denoted by the circles labeled 'P'. They are engaged in some common work, and to do so interact with various tools and products. Some of these are physically shared (for example, two builders holding the ends of a measuring tape), but all are shared in the sense that they contribute to the cooperative purpose. These tools and other objects are denoted by the circle labeled 'A' – the *artifacts of work*.

The participants communicate with one another as they work, denoted by the arrow between them. In real life this may be by speech (the builders with the tape), or letter (a lawyer and client); in fact, this *direct communication* may be in any of the categories of the time/space matrix. Part of the purpose of communication is to establish a common understanding of the task the participants are engaged in. This understanding may be implicit in the conversation, or may be made explicit in diagrams or text.

For some jobs, such as research and aspects of management, the development of understanding and ideas constitutes the primary task. Where this is not the case, the participants will interact with the tools and work objects to perform their job. This is shown by the arrows between the participants and the artifacts of work. This arrow represents a two-way flow of information: of *control* from the participants to the artifacts, and *feedback* from the artifacts to the participants. In real-world tasks, these two hardly seem distinct: as you wield a hammer, you feel the weight of it. However, as should be evident from the previous discussion of single-user interfaces, this will not necessarily be the case for computer systems.

We will classify groupware systems by the function in this framework which they primarily support:

computer-mediated communication supporting the direct communication between participants;

meeting and decision support systems capturing common understanding;

shared applications and artifacts supporting the participants' interaction with shared work objects – the artifacts of work.

Of course, many systems may support more than one of these functions, and indeed this can be seen as a sign of good groupware. Furthermore, there are some further relationships between these functions. Both these aspects will be developed in Section 19.6.3.

19.3 COMPUTER-MEDIATED COMMUNICATION

Implicit in the terms groupware and CSCW is that we have two or more participants and that they are communicating with one another. We begin by looking at systems which support this direct communication. This is called *computer-mediated communication* (*CMC*), and is an important part of CSCW. However, good communication is not sufficient – the participants must be able to cooperate about their work. Improving communications may help this, but not necessarily.

Under the heading of computer-mediated communication we will look at email and bulletin boards, well-established, largely text-based means of *asynchronous remote* communication, structured message systems, developed from simple email, instant messaging and SMS, and various video-based systems, which support *synchronous remote* communication.

19.3.1 Email and bulletin boards

Most readers of this book will have used some form of email system. Many will also have used some form of bulletin board or electronic conferencing system such as Usenet News or Yahoo! Groups. Although among the simplest groupware systems, they are most certainly the most popular and successful.

Consider the stages during the sending of a simple email message:

1. **Preparation** You type a message at your computer, possibly adding a subject header.
2. **Dispatch** You then instruct the email program to send it to the recipient.
3. **Delivery** At some time later, anything from a few seconds (for LAN-based email) to hours (for some international email via slow gateways) it will arrive at the recipient's computer.

4. *Notification* If the recipient is using the computer a message may be displayed, saying that mail has arrived, or the terminal may beep.
5. *Receipt* The recipient reads the message using an email program, possibly different from that of the sender.

These stages differ slightly from tool to tool, but are essentially similar. For example, preparation may take place within the mail tool itself, or the tool may import an externally prepared document. Although email systems are so well known and simple, they have been the springboard for several more advanced groupware applications.

In theory, from the user's perspective we need not worry about the exact delivery mechanism – it should be hidden – just as we do not worry about the details of telephone exchanges. However, with email these mechanisms become only too apparent, most obviously in the varying delays we may experience and in the reliability of different channels. These factors are not always predictable as they may depend on faults and loadings of computers used to relay the message. Furthermore, error messages and even the forms of addressing relate directly to the path from computer to computer which the message takes.

In the simple email example, there was just one recipient. However, most email systems also allow a set of recipients to be named, all of whom receive the message (see Figure 19.3). Like letters, these recipients may be divided into the direct recipients (often denoted by a `To:` field) and those who receive copies (`Cc:`). These two types of recipient are treated no *differently* by the computer systems – the distinction serves a social purpose for the participants. This is a frequent observation about any groupware system – the system should support *people* in their cooperation.

Often users will be able to set up *distribution lists*, named groups of other users to whom mail is often sent. There is only a small difference between the use of such lists and fully fledged bulletin boards or electronic conferencing systems. In these systems you address your message (or contribution) to a particular conference or newsgroup, and then anyone who has subscribed to that newsgroup receives the message.

It is to be expected that the differences between email and electronic conferencing, especially within the interface, will blur over time. However, there are important differences for the participants. First, they vary in terms of who controls the

```
To: janet, abowd
From: alan
Subject: HCI book
Cc: R.Beale@cs.brum.ac.uk

How are your chapters getting on?
Could one of you meet me over lunch?
I'm having trouble using the minipage
environment doing illustrations of email messages.
```

Figure 19.3 Simple email message

distribution list. Some email distribution lists are private to the sender, that is the *sender* can vary the list at will, and knows exactly who is on it. Alternatively, a distribution list may be shared, in which case it is often added to and updated by the *system administrator*. Finally, in a bulletin board or news systems, it is the *recipient* who decides which newsgroups to subscribe to. For example, Usenet News sends *all* news items to *all* connected computers. Users may well read contributions posted before they subscribed, and certainly the sender will have no idea as to who will read the contribution. We can see this progression in two ways. One is control which moves from the sender, to the administrator and then to the recipient. The second is the nature of the recipient, from specific individuals in email, through organizational or social groups, to topic areas. Again these different emphases have important social implications – communication is about people, not systems.

Although email between sites takes from several minutes to hours to arrive, LAN-based email within a single site will often arrive within seconds. It is thus possible to have email 'conversations'. Typically email interfaces are not designed to deal with this form of exchange, but it is relatively easy to have a sequence of exchanges, say every minute or so. Many computer systems also allow a form of *synchronous* text-based communication, whereby two (or more) participants can instantly see each other's contributions. Examples of this are the UNIX 'talk' program or VAX 'phone'. In these, the participants' screens are divided into two and while your typing goes to the bottom half, your colleague's goes into the top. The typing is usually echoed character by character, so you can see your colleague's half-typed phrases, deletions, etc. A variant of this concept, used as part of several groupware systems, is to have a single transcript. The participants have a separate composition area and when they hit the return key (or click on a special button), their contribution is added to the end of the transcript. Notice the difference in *granularity*: the 'talk' program works with a grain size of characters, whereas the transcript acts on contributions (often a single line, or at most a short paragraph). The granularity of standard email and bulletin boards tends to be greater again, as people may send quite large messages (although some interfaces discourage this).

The ether never sleeps

One advantage of asynchronous messages is that they do not interrupt your work and allow you to pursue tasks in your own way, giving you greater autonomy and personal control.

Because of the ubiquity of internet mail and personal access via modems, it is increasingly the case that people (especially in the US) assume that you read email every day, either at the office or at home – including weekends and when you are on holiday! You may arrive back after a weekend or week away at a conference and find messages which expect you to have done something yesterday . . . autonomy and personal control?

19.3.2 Structured message systems

A common problem with email and electronic conferencing systems is *overload* for the recipient. As distribution lists become longer, the number of email messages received begins to explode. This is obvious: if each message you send goes to, on average, 10 people, then everyone will receive, on average, 10 messages for each one sent. The problem is similar to that caused for paper mail by photocopiers and mail merge programs. If we consider that newsgroups may have hundreds of subscribers, the problem becomes extreme. Happily, most newsgroups have only a few active contributors and many passive readers, but still the piles of unread electronic mail grow. Various forms of *structured message system* have been developed to help deal with this overload, perhaps the most well known being the *Information Lens* [225]. This adopts some form of *filtering* in order to sort items into different categories, either by importance or by subject matter. As well as a text message, normal email has several named fields: `To`, `From`, `Subject`. Structured message systems have far more, domain-specific, fields. The sender of the message chooses the appropriate message type, say a notification of a seminar. The system then presents a *template* which includes blank fields pertinent to the message type, for example `Time`, `Place`, `Speaker` and `Title`. Figure 19.4 shows a typical structured message similar to those in Lens.

The named fields make the message more like a typical database record than a normal email message. Thus the recipient can filter incoming mail using database-like queries. This can be used during normal reading – 'show me all messages `From` "abowd" or with `Status` "urgent"'. Alternatively, users may set up filtering agents to act on their behalf. Such an agent is a sort of electronic secretary; it is programmed with rules based on the field contents and can perform actions such as moving the message into a specific mailbox, deleting the message or informing the user.

The problem with such systems is that they put a great burden on the sender to fill in the fields accurately, but it is the recipient who benefits. This problem of

```
Type:    Seminar announcement
To:      all
From:    Alan Dix
Subject: departmental seminar

Time:    2:15 Wednesday
Place:   D014
Speaker: W.T. Pooh
Title:   The Honey Pot
Text:    Recent research on socially constructed
         meaning has focussed on the image of the
         Honey Pot and its dialectic interpretation
         within an encultured hermeneutic.
         This talk ...
```

Figure 19.4 Structured message

disproportionate benefit recurs throughout CSCW. In order to make the job of the sender as easy as possible, message types will often be created with easy defaults for fields, and perhaps menus of alternatives. Also, in order to make finding appropriate message templates easier, they may be arranged in a type hierarchy.

Until recently these structured message systems were only found in academic and research centers, but simpler forms are now available in commercial PC-based email products.

More complex structured message systems are based on models of conversation. So, for example, if I am sent a message of type 'request for information', I am obliged to return either a message of type 'informative reply' or of type 'don't know'. There is a variety of such systems differing in the models used and the rigidity with which they are applied. In the most rigid, you are only allowed one of the pre-programmed replies, whereas more flexible systems merely suggest possibilities. We discussed one such system, *Coordinator*, in Chapter 14 (Section 14.3.6), in the context of models of conversation.

There is, in fact, considerable debate between those who feel that messaging systems should impose conversational structures and constraints, and those who believe they should supply systems within which the participants can develop their own structures. It is argued that such user-defined structures are more likely to meet users' needs, and the systems will be more flexible to accommodate changes in group working. The message systems based upon conversational models are the most constrained. Those offering structured messages, as we have described above, are toward the middle of the spectrum. The original email and bulletin board systems were at the other, unstructured, extreme, but lack sufficient features for the users to define their own structure. More modern systems from the 'user-structured' arena are developed from the bulletin board concept, but are more like a shared hypertext. The structure allowed by links and cross-references allows users to have subconferences and digressions, to annotate each other's messages and to post follow-on messages. An example of such a system is the *Amsterdam Conversation Environment* [113].

This conflict between *global structuring* by the designers and *local structuring* by the participants in their own situation arises within many areas of CSCW and has similarities to issues of user control in HCI in general (see Chapter 7).

19.3.3 txt is gr8

While groupware developers produce more and more complex structured message systems, in the wider world people have voted with their fingers and adopted unstructured informal text messaging systems: instant messaging while online and SMS or paging through mobile phones.

In November 1996 a small Israeli start-up company, Mirabilis, launched a program, ICQ, that allowed registered users to send short text messages to one another. By 2003 this had grown to over 100 million users. Other large internet companies soon noticed this rapidly growing phenomenon, including AOL and Yahoo! who launched their own, incompatible, messaging systems based on their existing user groups.

Many readers will have accounts with one or more of the instant messaging (IM) clients. Compared with email, messaging is more lightweight and instant. Users can send a message, usually a single line of text, to any other registered user of the system. If the other user is currently logged into the system they receive an instant notification of the message and can start a chat-style session. If the other user is not logged in, the central system remembers the messages and when the user next logs on they receive any messages stored for them. So IM systems are unusual in that they allow a fluid movement between asynchronous and synchronous use.

Whereas email is more like an exchange of letters, IM is more like conversation with short exchanges, often not even complete sentences.

Hi, u there
yeh, had a good night last night?
uhu
want to meet later

Because of the conversational style there is extensive use of *emoticons* or smilies (see Section 14.4.1). Whereas in traditional email these were simply a text convention such as ':-)', messaging systems (and now many email systems too) substitute small icons (like the face above) and allow easy insertions of faces using drop-down lists.

SMS stands for Short Message Systems, but few users know this or care – they simply send an 'SMS' or 'text' a friend. The SMS phenomenon is a salutary lesson to any of us who believe we can design the future. It was originally an 'extra' service, making use of a technical fix that allowed mobile phone companies to send messages to mobile phones to inform of voicemail or update internal settings. However, in many countries SMS has overtaken voice calls in its use of airtime.

Like IM, SMS messages are short plain text messages – limited to 160 characters in the original system. However, the potential recipient is anyone with a mobile phone. SMS was able to break the critical mass problem (see Chapter 13) because even when only a few phones were able to send SMS messages, many existing phones were able to receive them. The real surprise of SMS was not just that it grew, but that the group who adopted it as their own were not business people wanting the latest stock update, or sales representatives sending an order, but teenagers. From being an add-on service, text messaging became *the* reason people bought mobile phones and pricing plans are often focussed on the number of free text messages per month.

Perhaps the defining feature of SMS text messages is the fact that they are entered via a telephone keypad. To type 'hello' you typically need to type '4433555*pause*555666' using multiple number keys to produce the letters. SMS users soon 'lrnt 2 txt usng shrt wds', both to reduce the effort of typing so many letters and also to allow longer messages within the same 160 character limit. Belatedly, technology caught up and text entry was made easier using the T9 algorithm (T9 is a trademark of Tegic Communications), which matches single number strokes against a dictionary: so '43556' becomes 'hello', as other matches to the digits such as 'gfjkn' are not proper words. Paradoxically, many experienced txt-ers turn this off as it makes the messages longer (and so fits less into 160 characters) and also looks less cool!

Another paradox of SMS is that it is an area where the United States lags behind due to the fragmented nature of its mobile telecoms industry. Dedicated devices such as pagers with small keyboards are used, but not with the same penetration that SMS has achieved elsewhere.

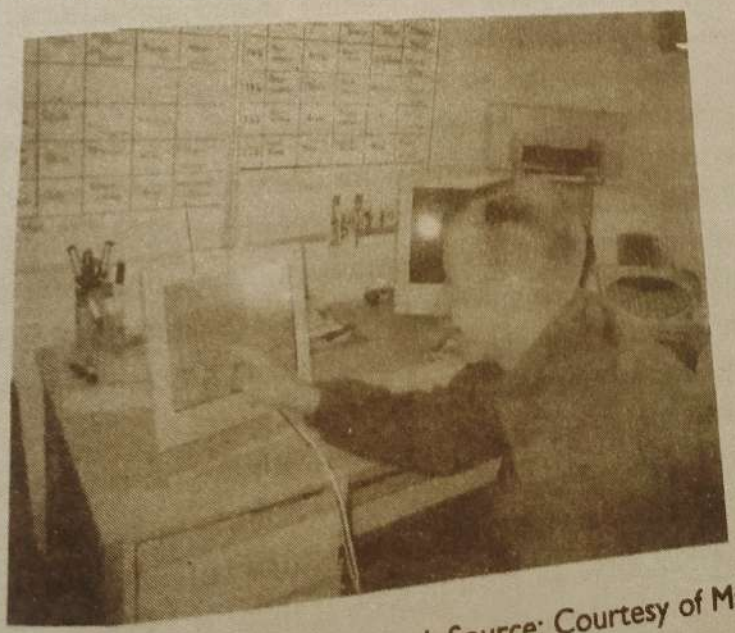
Both IM and SMS have gradually become more 'functional', adding features such as file sharing with IM and links between IM, email and SMS services. Also there are IM clients for PDAs allowing users to keep messaging even when away from their computers. As countries adopt higher bandwidth mobile phone networks, the potential for new services has increased. Japan has led the world here with several (incompatible) services. This was partly a technical issue and partly demand driven due to long commute times and high fixed-line telephone costs. Some of the services are likely to be closer to traditional broadcast media, but messaging is becoming multi-media with voice and image messages based on the MMS (Multimedia Messaging Service) standard. Many phones now include a digital camera so that it is possible to take a photograph whilst at a party and instantly send it to all your friends.

DESIGN FOCUS

SMS in action

SMS is not just for teenagers to hear the latest gossip or arrange to meet at the local night club. In the north of England the Croftlands Charitable Trust is using an SMS-based system called SPAM to coordinate the activities of its staff in a hostel and associated semi-independent living accommodation for former psychiatric patients [70]. Staff and patients can send SMS messages to a central number which are then displayed on touchscreens in the staff offices. As staff spend a considerable amount of time off site this is used extensively, for example when meetings over-run or people encounter traffic delays. Also, they can leave messages without tying up the phone during busy times of the day.

See the book website for a full case study: /e3/casestudy/spam/



messages received. Source: Courtesy of Mark Rouncefield

In contrast with email, IM grew as an informal communication medium. There has been some concern in the business community that it would add to internet surfing and private email as a drain on employees' time. However, several studies seem to indicate that the opposite is true and that IM is being used to facilitate group working and business contacts. SMS, too, is being used for 'serious' uses, from paying for goods at vending machines to coordinating staff activity in a residential facility for former psychiatric patients.

19.3.4 Video conferences and communication

The idea of *video phones* has been around for a long time, from Flash Gordon's days onwards, and early video phones have been available for at least 20 years. Until recently, pervasive person-to-person video has been impossible without special and very expensive cabling. However, the introduction of *ISDN* changes this dramatically. ISDN (integrated services digital network) is a form of moderately high-bandwidth (64 kbaud) digital telephone connection. It is available in major cities across the world and enables, amongst other things, LAN-type connections between computers and real-time video connections. The need for this becomes apparent if you calculate the bandwidth required for a typical video image. If we consider an 8 bit 600×400 pixel image being sent at 25 frames a second, this amounts to 24 million bits per second. This is not high video quality and yet requires a vast bandwidth far in excess of a normal telephone line. It is also far greater than the capacity of ISDN lines. However, recent (and expensive) *video compression* techniques can reduce the required capacity to a level within the range of ISDN. Pervasive video communication is clearly a technology coming of age.

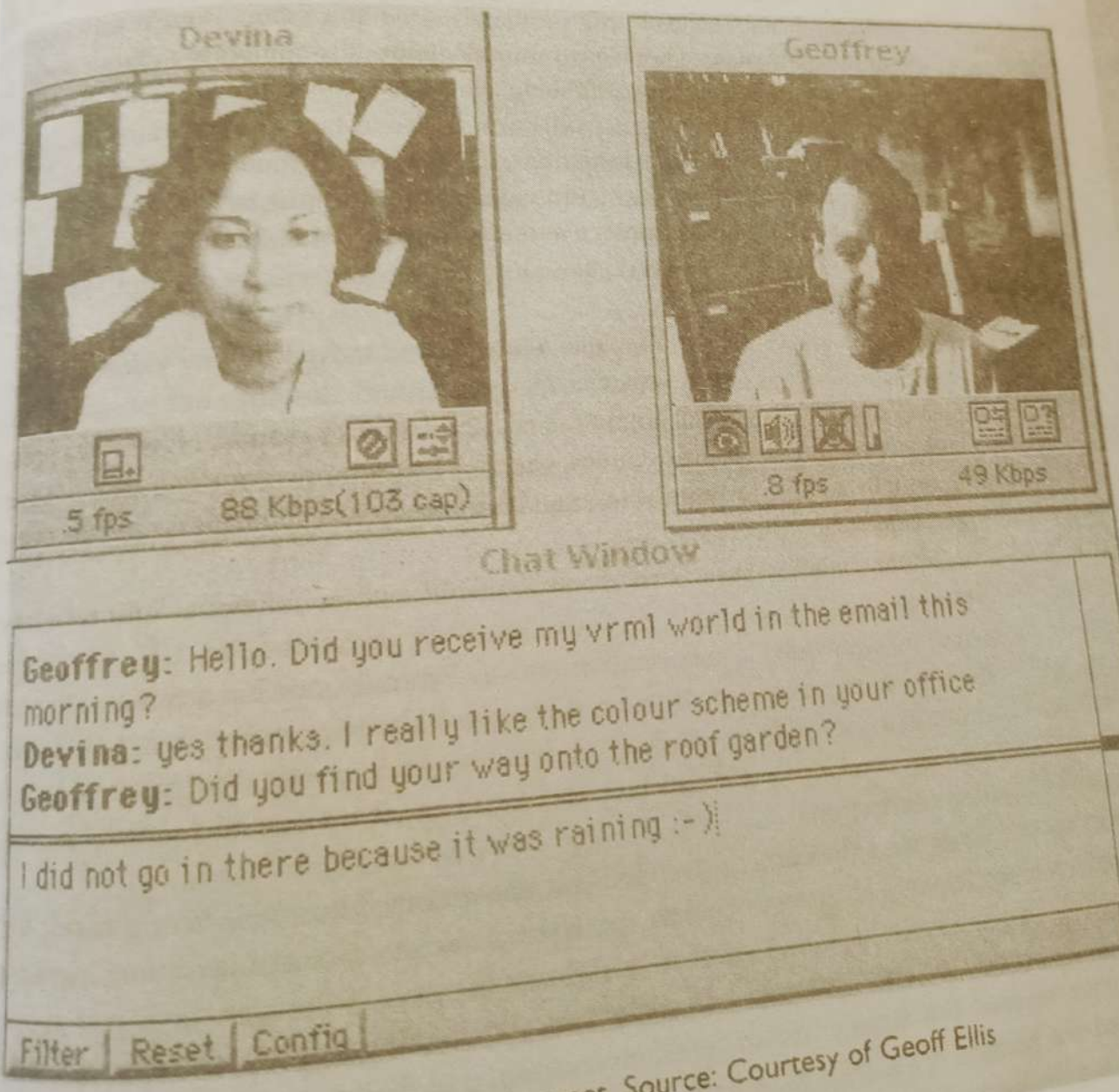
As these hardware developments are quite recent, most experimental and commercial video systems use existing technology and *lots* of wire. There are three broad uses of video: *video conferences*, pervasive video for enhancing social communication and video integrated with another shared application. These are all *synchronous remote* facilities.

Video conferences sit rather oddly in CSCW in that they typically do not use computers at all! However, computers and telecommunications are becoming so interlinked that it is widely considered an appropriate area of CSCW. Video conference facilities are readily available commercially using dedicated telecommunications lines and satellites for transcontinental conferences. The quality of a video conference compared with a face-to-face meeting is appalling, but, when faced with the costs of, say, flying executives across an ocean or even across the American continent, the inconveniences are often accepted.

One set of problems is connected with the small field of view of a television camera, and the size and quality of the resulting images. Even with a one-to-one video conversation, we need to decide whether to take a simple head and shoulders shot, the whole torso, or head to foot. If there is a group at either end, even just two or three people, the problems magnify enormously. If you view all three people at once, then the image of the speaker may becom

CuSeeMe

Special-purpose video conferencing is still relatively expensive, but low-fidelity desktop video conferencing is now within the reach of many users of desktop computers. Digital video cameras are now inexpensive and easily obtainable. They often come with pre-packaged video phone or video conferencing software. However, the system which has really popularized video conferencing is a web-based tool. CuSeeMe works over the internet allowing participants across the world owning only a basic digital video camera to see and talk to one another. The software is usually public domain (although there are commercial versions) and the services allowing connection are often free. The limited bandwidth available over long-distance internet links means that video quality and frame rates are low and periodic image break-up may occur. In fact, it is sound break-up which is more problematic. After all, we can talk to one another quite easily without seeing one another, but find it very difficult over a noisy phone line. Often participants may see one another's video image, but actually discuss using a synchronous text-based 'talk' program.



CuSeeMe – video conferencing on the internet. Source: Courtesy of Geoff Ellis

gestures. These gestures are one of the big advantages of video conferences over the much cheaper telephone conference. However, you need a skilled camera technician to follow the speaker, zooming in and out as necessary. Furthermore, zooming in to the speaker runs the risk of losing the sense of presence. The participants at the far end do not know whether the speaker's colleagues are nodding in agreement or falling asleep!

Video conferences support specific planned meetings. However, one of the losses of working in a different site from a colleague is the chance meetings whilst walking down a corridor or drinking tea. Several experimental systems aim to counter this, giving a sense of *social presence* at a distance. One solution is the *video window* or *video wall*, a very large television screen set into the wall of common rooms at different sites [134]. The idea is that as people wander about the common room at one site they can see and talk to people at the other site – the video wall is almost like a window or doorway between the sites.

The problems of camera positioning and focussing are, if anything, worse for the video wall than for the simple video conference. At least in a video conference the participants stay relatively still, probably seated at a table, whereas in a common room the participants are likely to wander about. It is quite easy to move out of the range of the camera, whilst still being able to see your colleague. That is, there is a lack of *reciprocity* compared with normal face-to-face conversation. In addition, positioning the camera is a nightmare. To give the 'window' illusion, the camera must be positioned very close to the video wall and must be focussed to get a full-depth shot of the conversants. Even then, because of the camera's restricted field of view, you must not stand too close or your colleague will get a 6 foot (1.8 m) high view of your navel!

A stereo audio channel for the video wall, and indeed for any video connection, can help the participants orient to the speaker and also filter out unwanted noises (see Chapter 1 for a discussion of selective attention to sound). However, a problem with communicating using loudspeakers in a common room is a lack of privacy. Normally, people just move closer and lower their voice, but this is not effective with the video wall.

Similar facilities have been made available within individual offices at several research sites. For example, at Xerox EuroParc in Cambridge, UK, every office is wired up with video cameras and monitors. Typically, one has a camera strapped to the top of one's monitor, or on an angle-poise-like arm, and an image on a separate monitor, or even in a window on the computer screen. Participants may have two-way or multi-way video conversations, or set up virtual rooms – a constant video connection.

Even with the camera strapped just above a monitor, it is very difficult for the participants to get *eye contact*. In normal face-to-face conversation, participants periodically look one another straight in the eye. If the camera is just above the monitor, your partner will always see you looking slightly downwards, and vice versa. For strictly one-to-one conversation, a technique called the *video tunnel* can be used (Figure 19.5). A half-silvered mirror is used so that the camera can view the user as if it were in the middle of the screen. The feeling of engagement between the participants is reported to be far greater than

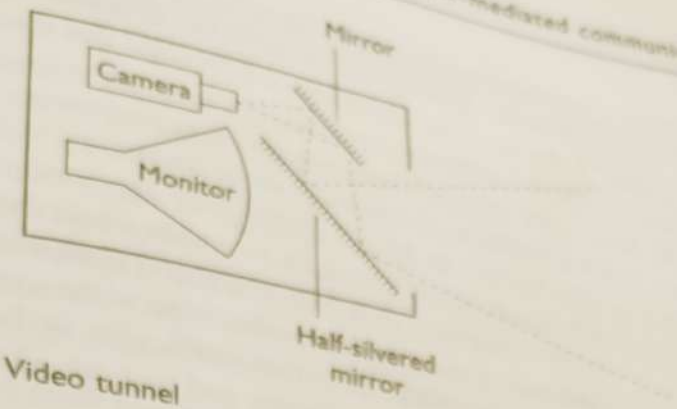


Figure 19.5 Video tunnel

[326]. Unfortunately, the sheer bulk of the video tunnel means that it is not really practical for large-scale use. However, frequent users of less optimal camera/monitor configurations eventually get used to the rather odd visual angles involved.

19.3.5 Virtual collaborative environments

Virtual reality techniques are now being used to allow participants to meet within a virtual world. Each participant views a virtual world using desktop or immersive VR. Within the virtual worlds the other participants appear, often rendered as a simple cube-based figure perhaps with a fixed or even video image texture mapped onto it. The representation of a participant in *virtual space* is called an *embodiment*.

As participants move around in virtual space then their embodiments move correspondingly. Thus one can approach another within the virtual environment, initiate discussion and focus on common objects within the virtual environment. These systems attempt to mimic as much as possible of the real world in order to allow the participants to use their existing real-world social skills within the virtual world. For example, when other people speak, the volume may be adjusted so that those a long way away or behind you (in virtual space) are quieter than those close and in front. This is clearly very important in a heavily populated virtual environment.

Given the virtual environment is within a computer, it makes sense to allow participants to bring other computer-based artifacts into the virtual environment. However, many of these are not themselves 3D objects, but simple text or diagrams. It is possible to map these onto flat virtual surfaces within the 3D virtual world (or even have virtual computer screens!). Text is especially difficult to read when rendered in perspective and so some environments take advantage of the fact that this is a *virtual* world to present such surfaces face on to all participants. But now we have a world the appearance of which is participant dependent.

Imagine you and a colleague are facing each other and looking at the same text object. It will be rather like having a piece of paper held up between you with the same text printed on both sides. Your colleague points at the left-hand side of the text and refers to it. What should you see – a virtual finger pointing at the wrong side of the text, a disembodied hand tear off from your colleague's arm and point at the

correct place on your side of the paper? This is similar to the problems of shared focus which we shall discuss in the next section, but perhaps worse in this context as the users are lulled into a false sense that the world they are dealing with is truly like the normal real world.

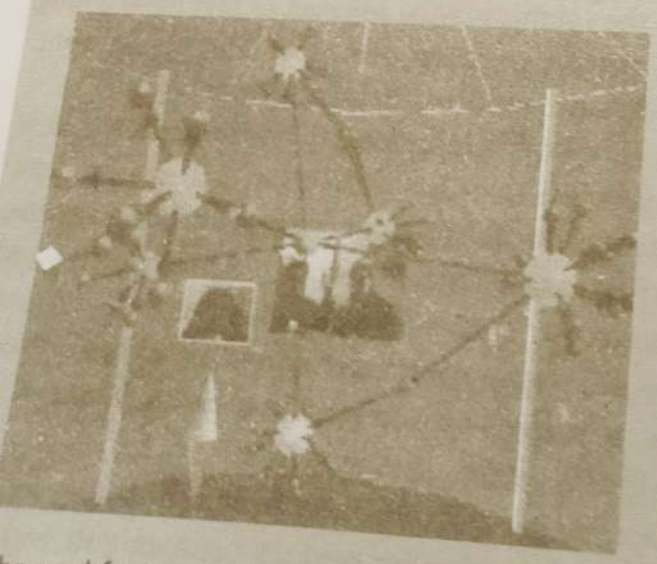
Although collaborative virtual environments are still mainly research systems, it is possible to sample some of the features by visiting 'virtual places' on the web [84], also known as *MUDs* (*Multi-User Domains*). When you visit the page, you are allowed to specify an icon or image which becomes your embodiment or *avatar* within the pages. Your own avatar and those of other visitors appear within the page. Although these are 2D worlds, many of the same possibilities arise. You can bump into people within the virtual world, who, as they are visiting the same place as you, may share common interests and you can also usually initiate text-based or even audio conversations.

Real and virtual worlds meet

The *video wall* allows people from remote locations to meet. In a sense it extends the normal physical space of the participants as they can see the remote room, but the image rendered is a real physical space, albeit on a video screen.

Virtual collaborative environments also allow remote participants to meet one another, this time within a purely virtual space.

The *Nottingham Internet Foyer* merges these two worlds. In a real physical foyer at Nottingham University there is a large video screen. However, what you see when you look at it is not a room in a remote building (as with a video wall), but a fixed view of a virtual space. The virtual space that is seen is itself a collaborative virtual environment, a sort of virtual foyer, which participants can enter over the internet using either desktop or immersive VR systems. The embodiments of the electronic participants are visible to those at the real foyer. In addition, the video image of the real foyer is also projected onto a surface in the virtual foyer. Thus the physical foyer and the virtual foyer are linked by a virtual video wall!



he real foyer and the electronic foyer. Source: Courtesy of Steve Benford, The Mixed Reality boratory, The University of Nottingham

19.4 MEETING AND DECISION SUPPORT SYSTEMS

In any conversation, the participants must establish a common understanding about the task they are to perform, and generate ideas. In some areas this is a secondary activity: you are discussing the job you are doing, the ideas support that job. However, there is a class of activities where the job is itself generating ideas and understanding. This is typically the case in a research environment, in design tasks, in management meetings and brainstorming sessions.

We will discuss three types of system where the generation and recording of ideas and decisions is the primary focus.

Argumentation tools which record the arguments used to arrive at a decision and support principally *asynchronous co-located* design teams.

Meeting rooms which support face-to-face groups (*synchronous co-located*) in brainstorming and management meetings.

Shared drawing surfaces which can be used for *synchronous remote* design meetings.

19.4.1 Argumentation tools

In Chapter 6, we discussed *design rationale*, methods of recording decisions and the arguments that lead to those decisions. We saw that design rationale could fulfill two roles: recording design decisions for future references and clarifying the available options during design. Both of these have a potential CSCW impact. One reason to record your design decisions is so that a future designer can understand why you made the decisions you did and hence be in a better position to modify or learn from your design. That is, you are communicating with someone possibly years in advance. However, the one-way nature of the communication makes it information passing rather than collaboration.

More important from a CSCW viewpoint is when the design argument is used to communicate decisions between a group of designers. Here the communication is two way – the designers may both add to the design argument and look at each other's contributions. Argumentation support tools often have a hypertext-like structure, and may easily be used to support design teams as well as individuals. At the simplest level, the designers can use the tool one at a time, rather like writing a co-authored document by taking turns to use the word processor. As the detailed design of, say, a piece of software may take many months it is unlikely that two people will want to use the tool at exactly the same moment, and if they do . . . well it is a chance for an impromptu design meeting!

Sophisticated tools also have facilities to allow several designers to use the system simultaneously. To allow this, the tool must have mechanisms to stop the different designers' work interfering with that of another. This problem is called *concurrency control*. Happily, this is particularly easy with hypertext – so long as people work on different nodes, there is no conflict. To ensure this, a node must be *locked*; that is, when one participant starts to edit the node, no other participant is allowed to edit

or update the same node. Given that there are typically hundreds of nodes in a design rationale, with only a short text description in each, the chances of two participants wanting to work on the same node are small. However, if they do, the system will refuse permission to one or the other. In addition to these locking mechanisms, the systems have *notification mechanisms* to let the participants know which nodes have been edited. For example, changed nodes may be highlighted until the participant has examined them or marked them as read. This highlighting must, of course, be on a per user basis – if Jane edits a node and Sam marks it as read, then when Mary comes to the system she should still see the node highlighted.

A good example of argumentation tools is the family of tools developed to support the argumentation model called *issue-based information system* (*IBIS*), the best known being gIBIS [76] (which we met in Chapter 6). This system has node types including 'issues', 'positions' and 'arguments', and these are linked together by relationships such as 'argument *supports* position'.

Notice that argumentation tools may allow a range of interaction styles from asynchronous, when the designers use it one at a time, to fully synchronous, when several use it at once. Although there is no reason why the systems should not be used for distant collaboration they are typically used by groups within the same office, and in the case of one-at-a-time use, on one machine. They are thus largely, but not solely, *asynchronous co-located* groupware systems.

19.4.2 Meeting rooms

The advantages of email, bulletin boards and video conferences are obvious – if you are a long way apart, you cannot have a face-to-face meeting. Similarly, it is obvious why one should want to record decisions during a long-lived design process. The need for *meeting rooms* is less obvious. These are specially constructed rooms, with extensive computer equipment designed to support face-to-face meetings. Given face-to-face meetings work reasonably well to start with, such rooms must be very well designed if the equipment is to enhance rather than disrupt the meeting.

Early examples of such rooms, catering for groups of between four and 30 participants, included Xerox PARC's CoLab [329], Project Nick [32] and Capture Lab [226]. The general layout consists of a large screen, regarded as an electronic whiteboard, at one end of the room, with chairs and tables arranged so that all the participants can see the screen. This leads to a U- or C-shaped arrangement around the screen, and in the biggest room even several tiers of seating. In addition, all the participants have their own terminals, which may be recessed into the tabletop to reduce their visual effect. Figure 19.6 shows a computer-supported meeting using NLS, a very early groupware system [125]. Obviously, changes in display technology have made the displays less obtrusive, but the basic principle remains.

Such systems will support various forms of working, including private use of the terminals and subgroup working on a teleconferencing or email basis. However, the characteristic mode of operation is where all the participants' screens and the central screen show the same image. This is termed *WYSIWIS* ('*what you see is what I see*').

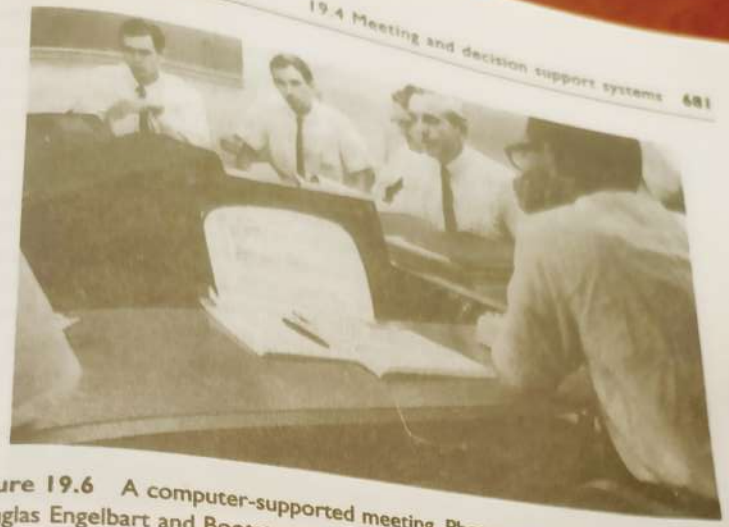


Figure 19.6 A computer-supported meeting. Photograph courtesy of Douglas Engelbart and Bootstrap Institute

The screen then takes the form of an electronic whiteboard, similar to a simple graphics drawing package, on which the participants can all write. One advantage of such an arrangement over a normal whiteboard is the ease with which data can be moved to and from the participants' normal computer files. As more work is prepared online this will increase in importance. Also, the electronic whiteboard has some advantages over the real thing. Whereas on a real whiteboard you can only write and rub out, on the electronic version you can move items around just as you would with a drawing tool, and, of course, you can get a printout of the results rather than just hoping the cleaners do not wipe it all off.

There are potential problems if several participants decide to write at the same time, so different systems adopt different *floor control policies* to determine which participant can write at any moment.

The simplest policy to implement is to use *locking*, similar to that described previously. When a participant, say Jane, wants to write to the screen she presses a key, or clicks on an on-screen button to request the floor. If no one else has the floor, she may go ahead and type on the screen, or if it supports graphics draw a diagram. When she has finished, she relinquishes the floor, using some other key or mouse selection. However, if some other participant, say Sam, already has the floor when Jane requests it, she must wait until Sam relinquishes the floor. There will be some sort of status indicator to say who has the floor at any moment, so Jane can ask Sam to relinquish, just as you might ask for the pen to write on a whiteboard.

In practice, such meetings tend to be punctuated with far too many requests such as 'Sam have you finished yet?', as often a participant will forget to relinquish the floor after writing. Also, it can be a pain to request the floor explicitly from the system – with a whiteboard you simply pick up your pen and write. Various more

lenient locking mechanisms are used to reduce these problems. A lock may be *implicitly* requested when you begin to type or draw. If no one else has the floor, then the floor is implicitly granted. If someone else has the floor, then your writing will be blocked. There is also implicit relinquishing: the lock may be automatically released after several seconds of inactivity. This may, of course, happen during a pause, rather than when the floor holder has finished, but the ease with which the floor can be regained does not make it a nuisance.

A conceptually simpler idea is to let everyone write to the screen at the same time – just as people may use several pens to write on a whiteboard at the same time. This sounds like a recipe for chaos, with people forever writing on top of one another. However, with reasonably small groups there is little problem. Try going to the whiteboard as someone is writing on it, and writing on top of what they write. It is possible, but it does not make you many friends. Similarly in the electronic version, as soon as two people begin to write on the same part of the screen they say 'oops!' and start to write elsewhere.

The various forms of locking constitute a software protocol for floor control. The way that people negotiate for screen space in the free-for-all situation is a *social protocol*. The reason that this is possible is because the participants are in the same room and are able to talk to one another.

If there is only one floor holder, then the screens can all show the floor holder's cursor. However, as soon as several participants are active at once, it is less clear what to do. One option is to display all the users' cursors. These may be accompanied by the user's name so that you can tell who is entering what. With large numbers of participants this can become distracting, and it is costly in terms of network traffic. The alternative is to show none of the cursors. If the participants are talking as they write, it is usually obvious who is writing what. In addition, during brainstorming phases, anonymity is an advantage – people are more likely to put up an 'off-the-wall' suggestion, thus stimulating more ideas and discussion.

If you are using a real whiteboard, you may go up to a diagram on the board and say 'I think *that* should go *there*'. As you say the words '*that*' and '*there*', you point at the relevant parts of the diagram. This is called *deictic reference* or simply *deixis*. If the participants' cursors are invisible to one another, then this form of pointing is impossible. Indeed, in such a meeting, even where the cursors are visible, the participants may momentarily forget and point at their own screen. Obviously, the participant can get up and point at the shared screen as you would at a whiteboard, but that is rather intrusive and precludes writing at the same time. To allow this deictic reference a *group pointer* may be supplied. This is an icon visible on all the screens, perhaps in the form of a pointing finger. Any participant can pick it up with his own mouse and use it to gesture on the screen. The control of the group pointer poses similar problems to those of floor control. However, the most lenient locking policies work well since the use of the pointer is usually synchronized with speaking, and thus it is easy to avoid and resolve conflicts.

The design and building of meeting rooms is both expensive and time consuming, but less sophisticated facilities are more widely available. The simplest is the hardcopy whiteboard, which h

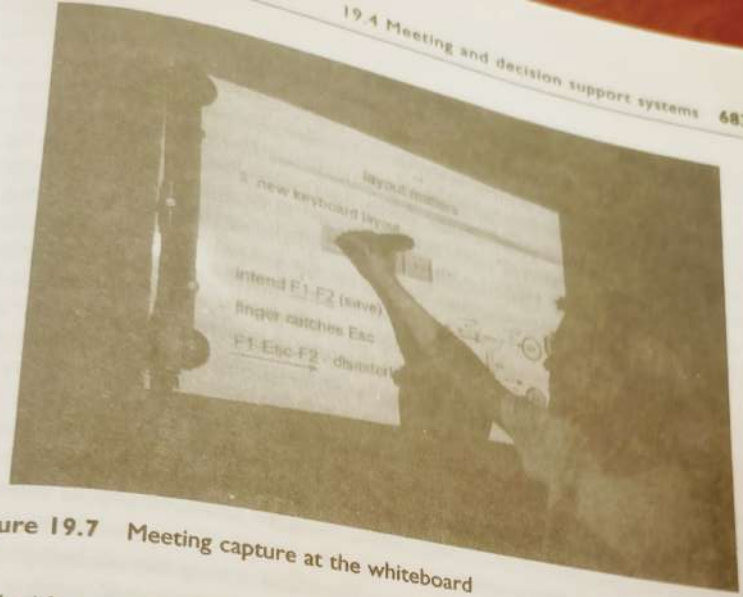


Figure 19.7 Meeting capture at the whiteboard

(and with greater resolution). If these were more closely coupled to a computer system, they would have even more scope. LCD projectors or transparent LCD screens for putting on top of an overhead projector can give any computer a vast screen image. In the simplest case this can just be manipulated by one person, but if several computers are networked together and use commercial shared screen software (see below in Section 19.5.1), one can obtain a similar effect to that of the more expensive conferencing facilities (Figure 19.7). However, experience of the various meeting room projects has shown that the social dynamics are very fragile and the difference between a successful meeting environment and a complete disaster is narrow.

These lower cost and more readily available technologies have made it possible to *capture* the results of ordinary meetings (see also Chapter 20). A number of devices are available commercially to turn an ordinary whiteboard into an electronic surface. Ordinary pens are placed in some sort of holder so that fixed detectors around the board can track the location of the pen on the whiteboard. This can then be used to record the strokes produced on the whiteboard and so keep an electronic copy. If instead an LCD projector is used to project a computer screen onto the whiteboard, then inkless pens can be used to act as a mouse pointer and so allow interaction with applications. Both of these can be combined with audio recording to allow rich capture of meetings for future review, aiding minute making and so on.

19.4.3 Shared work surfaces

The idea of a shared screen forming an electronic whiteboard is not confined to face-to-face meetings. One can easily imagine using the same software which runs in

a meeting room, working between several sites. That is, we can take the synchronous co-located meeting room software and use it for synchronous remote meetings. As before, each participant's screen shows the same image, and the participants can write on the screen with the same sort of floor control policies as discussed earlier. There are additional problems. First, the participants will also require at least an audio link to one another and quite likely video as well. Remember that the social protocols used during lenient floor control, not to mention the discussion one has during a meeting, are difficult or impossible without additional channels of communication. As well as the person-to-person communications, the computer networks may have trouble handling the information. If there are delays between one person writing something on the board and another seeing it, the second participant may write to the same location. A situation which is easily avoided in the co-located meeting could become a major problem when remote. Many researchers in the area blithely assume that such problems will be solved by cheaper high-bandwidth telecommunications, such as ISDN.

In order to make the whiteboard effect more realistic, several systems are arranged so that participants write by hand directly onto large screens. The writing is either filmed by camera (using complex arrangements of mirrors), or captured digitally using a sensitive screen. The image of one participant's writing is then displayed on the others' screens. The effect is very like all being able to write at once on the same screen, except that the other participants' writing will be slightly less distinct than your own because of the resolution of the TV image. You may also experience problems of *parallax* as your own writing is on the outer surface of the screen and the projected image on the inside.

One system, VideoWhiteboard [340], arranges its lighting and cameras such that you can see not only the other participants' writing, but also shadowy images of their hands and bodies, getting gradually dimmer and more out of focus as they move away from the screen. For two participants, it is rather as if your colleague is writing on the other side of a smoked glass panel. When used by more than two participants, your ghostly colleagues can appear to occupy the same space and move through one another! This sounds rather disconcerting, but the users soon get used to the effect and are able to interpret one another's body language, even as a soft-focus shadow.

A third variation of the shared work surface is where the participants write on a sheet of paper on their desktop, which is then filmed from above. The images from each participant are then mixed and displayed on a screen in each participant's work area. By looking at the screen while they point and write, the participants can refer to one another's work. The advantage of such a system is that the participants' individual paperwork is easily integrated into their shared environment. The desktop images can also be mixed with a shared computer screen, so that paper and computer work can be mixed. One such system, the *TeamWorkStation* [188], has been used for the remote teaching of Japanese calligraphy. The student is able to paint letters on paper or on the computer screen and see these strokes overlaid with the teacher's strokes. In this system, the participants also have a face-to-face video link.

19.5 SHARED APPLICATIONS AND ARTIFACTS

The things that were being shared in the previous section were ephemera; they were there to support the meeting or design process, but were not the end purpose. In this section we will look at systems where the focus of sharing is the participants' work domain itself. These include the computers people are using, applications on those computers, and the documents they are working with. Some of these systems are similar in technology to the various shared work surfaces above, but the focus in this section is on *work*.

19.5.1 Shared PCs and shared window systems

Most of the groupware tools we have discussed require special *collaboration-aware* applications to be written. However, *shared PCs* and *shared window systems* allow ordinary applications to be the focus of cooperative work. Of course, you can co-operate simply by sitting together at the same computer, passing the keyboard and mouse between you and your colleague. The idea of a shared PC is that you have two (or more) computers which function as if they were one. What is typed on one appears on all the rest. This sounds at first just like a meeting room without the large shared screen. The difference is that the meeting rooms have special shared drawing tools, but the shared PC is just running your ordinary program. The sharing software monitors your keystrokes and mouse movements and sends them to all the other computers, so that their systems behave exactly like yours. Their keystrokes and movements are similarly relayed to you. As far as the application is concerned there is one keyboard and one mouse.

Imagine two users type at once. As the application does not know about the multiple users it will merely interleave the keystrokes, or should we say '*inkeytersltreaokevetshe*'? Interleaved mouse movements are, if anything, more meaningless. The sharing software therefore imposes some form of lenient locking. For the mouse, this will be an automatic lock while the mouse is being moved, with the lock being relinquished after a very short period of inactivity. The keyboard lock will have a longer period as natural gaps in typing are greater than gaps in mousing. Alternatively, the keyboard may have no lock, the users being left to sort out the control with their own *social protocol*.

A shared window system is similar except, rather than the whole screen, it is individual windows which are shared. While the user works with unshared windows, the system behaves as normal, but when the user selects a shared window the shared windowing system intervenes. As with the shared PC, all the user's keystrokes and mouse movements within the window are broadcast to the other computers sharing the window.

These facilities may be used within the same room, as originally suggested, in which case we have a *synchronous co-located* system. Alternatively, they may be used in conjunction with telephone or video connections at a distance, that is *synchronous*

remote. The extra audio or video channel is necessary when used remotely as the systems in themselves offer no direct communication. It is just possible to use such systems, without additional channels, by writing messages in the application's workspace (document, drawing surface, etc.). However, the social protocols needed to mediate the mouse and keyboard cannot be achieved by this channel.

Shared PCs and window systems have two main uses. One is where the focus is on the documents being processed, for example if the participants are using a spreadsheet together to solve a financial problem. The other is technical support: if you have a problem with an application, you can ring up your local (or even remote) technical guru, who will connect to your computer, examine where you are and offer advice. Compare this scenario with trying to explain over the phone why your column is not formatting as you want.

19.5.2 Shared editors

A *shared editor* is an editor (for text or graphics) which is *collaboration aware*, that is it knows that it is being shared. It can thus provide several insertion points or locking protocols more tuned to the editor's behavior. The software used in meeting rooms can be thought of as a form of shared editor and many of the issues are the same, but the purpose of a shared editor is to collaborate over normal documents. Just as with shared PCs and windows, the users are expected to have some additional means of communication, such as face-to-face (co-located), audio or video channels, or at the very least textual communications.

Shared editors may be text based or include graphics. For simplicity, we shall just consider text. Even so, there are a wide range of design options. Should you have a single insertion point with some form of *floor control* to avoid interleaving, or should you have one insertion point per participant? Assuming you have several insertion points, do you just see your own, or do you see your colleagues' insertion points as well, and if you can see them should they be identified by the user's name or be anonymous? In addition to the insertion point options, there is the issue about what you should see. Do all the participants see the same part of the screen, so if one participant scrolls, so do all the rest? Or do we allow different views on the document so that one participant can edit the beginning of the document while another edits the end?

There is not a right answer to these questions; different policies are useful for different purposes – close cooperation on a single sentence, or writing separate sections. Even within a single editing session the appropriate policy will vary. Unfortunately, in the past, these policy decisions were usually enshrined in the various shared editors, rather than being configured by the users. However, there is a growing recognition that more adaptable systems are needed to allow for the wide variation between groups, and within the same group over time. We will look at some of the options and how they affect the style of cooperation.

Thinking about the shared view versus different view options, it at first seems obvious that we should allow people to edit different parts of a document. This is

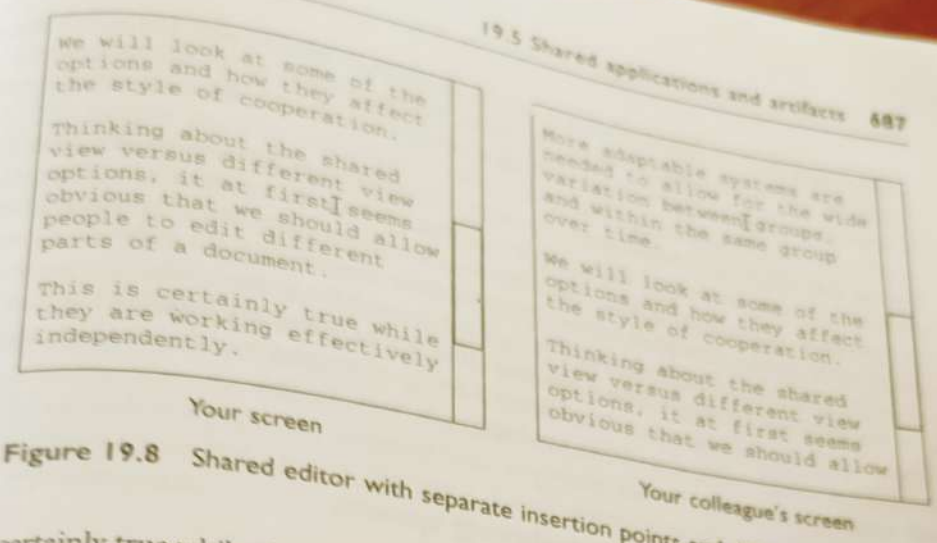


Figure 19.8 Shared editor with separate insertion points and different views

certainly true while they are working effectively independently. However, as soon as they begin to discuss the text together – that is, really *collaborate* synchronously – problems arise: 'I don't really like that line at the top' you say; 'I just wrote that, I think it's really good' your colleague replies. Possibly the end of your good working relationship, and, sadly, unnecessary. Your screens show different parts of the document and so the line at the top of your screen (which you disliked) is not the one your colleague has just written (Figure 19.8). Of course, the participants know they do not necessarily see the same screen, but you naturally use terms which relate to the context you can see, called *indexical expressions*. One reason given for the Charge of the Light Brigade is that the commander gave the order 'take the position on the hill'. Unfortunately the hill the commander could see was not the one in front of the Light Brigade. Paradoxically, the better the impression of a shared environment, the more likely it is that participants will accidentally use indexical expressions.

These problems are precisely why the principle of WYSIWIS – what you see is what I see – is used in meeting rooms. Even minor differences between displays, such as lags between the appearance of one participant's typing on the others' screens, can cause severe problems – no wonder different views cause trouble. Of course, WYSIWIS is not always appropriate, for example if we want to edit different parts of a document. Neither is it a solution to all problems. For example, if two people try to scroll the shared view at the same time, we have scroll wars. People find this conflict harder to resolve than typing clashes. This is probably because scrolling is a less direct and less predictable action anyway, and thus it is more difficult to diagnose what is going wrong. This suggests that better locking of scrollbars and visual clues are required. As we will discuss later (Section 19.7) graphics toolkits do not make such modifications easy.

Separate views, of course, demand separate insertion points. Even with shared views, it is not obvious why one should want a single insertion point with the attendant floor control problems. However, a shared cursor offers a point of focus for close cooperation, and should perhaps be an option. Of course, a shared view with

shared cursor is almost identical to a single-user editor in a shared window. Where there is no shared cursor, a *group pointer* can be used to focus discussion. Indeed, one could imagine using a group pointer with different views, but there are no extant systems which allow this.

As we have indicated, the differences between groups mean that some configuration by the group is essential. That is, shared editors require some form of *local structuring*. However, that does not mean that there will be no problems. Participants will point at the screen with their fingers when in different rooms, and use indexical expressions when they are in separate screen mode. The more their sense of *engagement*, that is the more they feel as if they are working together, the more likely they are to revert to natural forms of expression. We cannot prevent this entirely; as we saw, mistakes can happen in 'real life' too.

19.5.3 Co-authoring systems

Shared text editing is a short-term activity, occurring over a timescale of, at most, a few hours. Co-authoring is much longer term, taking weeks or months. Whereas shared editing is synchronous, co-authoring is largely *asynchronous*, with occasional periods of synchronous work. This may involve shared editing, but even if it does this is only one of the activities. Authors may work out some sort of plan together, apportion work between them, and then exchange drafts commenting on one another's work. In fact, this is only one scenario and if there is one consistent result from numerous studies of individual and collaborative writing, it is this: everyone and every group is different.

Fortunately, even though the details of writing differ, especially in the process, there are enough similarities to produce systems which support co-authoring. The majority of these are built around a hypertext model. The text itself is the basis and comments and discussion of these comments are linked into this basic structure. Although a general-purpose shared hypertext can be used, specific co-authoring systems are tuned to the writing task. The shape of the hypertext may be limited to a tree rather than the arbitrary graph of a hypertext. The document itself may be flat, a linear list of sections or may be grouped into a hierarchical section–subsection tree. Comment nodes are then attached to the document nodes, and other follow-on remarks attached to the comments. These are intended to emulate the scribblings on paper drafts during normal collaborative writing. Special facilities may be added to inform one writer of another's comments or additions.

As in previous systems, co-authoring systems must have some sort of *concurrency control* to cater for times when two participants attempt to edit the same text at the same time. This may take the form of locking, as described previously, or the system may allow the users to enter into periods of synchronous activity. That is, the node editor acts as a shared editor. However, this is not the dominant form of working and most often co-authors will have divided up the work amongst themselves, so that a section will have only one principal author, and thus *social protocols* ensure that writing clashes do not occur.

These social protocols may be supported or enforced by the co-authoring system. For example, in one system, *Quilt* [210], users are assigned *roles*, such as author, commentator, reader, with respect to each document node. An author is allowed to edit the text and add comments, a commentator is only allowed to add comments, and a reader cannot alter it at all. These roles resemble access rights in a normal filestore or database. However, both their naming and their particular semantics are aimed at supporting the types of role which occur in co-authoring situations. Of course, we are again into the territory of local versus global structuring. Who decides on the roles and the associated access rights, and can they be changed during use? These questions are not too much of a problem for Quilt, where the roles are on a per node basis, but some systems have roles which apply to the whole document. It is often the case that a person who gives extensive constructive comments is asked to co-author a paper, or someone who originally was to be an author cannot find sufficient time. Roles in real life have a degree of flexibility; those in groupware must try to keep suit.

19.5.4 Shared diaries

We want to find a time for a meeting to discuss the book we are writing: when should it be? Four diaries come out and we search for a mutually acceptable slot. We eventually find a free slot, but decide we had better double check with our desk diaries and the departmental seminar program. This sort of scenario is repeated time and again in offices across the world. The idea of a *shared diary* or *shared calendar* is simple. Each person uses a shared electronic diary, similar to that often found on PCs and pocket organizers. When you want to arrange a meeting, the system searches everyone's diaries for one or more free slots.

There are technical problems, such as what to do if no slots are free (often the norm). The system can return a set of slots with the least other arrangements, which can then form a basis for negotiation. Alternatively, the participants can mark their appointments with levels of importance. The system can then assign costs to breaking the appointments, and find slots with least cost. Mind you, someone may regard all their appointments as critical.

This reminds us that whatever the technical sophistication, it is people who use these systems and people who must cooperate. There are varying reports of success and failure with shared diaries. Where they have failed it is invariably because they have ignored the social needs and behavior of their users.

One such area is privacy. Are people allowed to look at your diary to find free slots? If so, do they just see 'busy' or can they see exactly what you are doing? You would be extremely annoyed if someone looked in your personal paper diary; is the electronic one any different? There is a trade-off between privacy and cooperation. In an office situation, one has a succession of diaries and calendars for specific purposes: a private diary with personal information, a desk diary and possibly a parallel desk diary with your secretary, and various forms of wall calendar showing periods on vacation and similar major meetings. These vary in their visibility to other

members of the organization: your secretary can consult her copy of your diary but not your personal diary; anyone can look at the wall calendar. You choose your level of privacy by where you put information. Of course, the problems of keeping such diaries up to date with one another is one reason why electronic diaries are produced, but they must supply similar forms of privacy control or people will simply not use them. This is precisely what happens.

There are similar problems with the update of diaries. If someone wants to book a meeting with you, can they fill in a slot, or must they ask you? Again you may want to vary these rights according to who it is and the sort of appointment. Many systems allow other users to 'pencil in' appointments, but require them to be confirmed by the diary's owner.

Many people use private electronic diaries and time management software. It is reasonable to expect that groupware versions of these will become commonly available in the near future. The extent to which they work will be largely the extent to which the design takes note of personal and social factors.

19.5.5 Communication through the artifact

In each of the last four systems – shared PCs and windows, shared editors, co-authoring systems and shared diaries – the focus has been upon the artifacts on which the participants are working. They act *upon* the artifacts and communicate with one another *about* the artifacts. However, as well as observing their own actions on the artifacts, the participants are aware of one another's actions. This awareness of one another's actions is a form of communication *through* the artifact.

This can happen even where the shared artifact is not 'real' groupware. For instance, shared files and databases can be a locus for cooperation. Sales figures may be entered into the company database by a person in one department and then used as part of a query by another employee. At a loose level, the two are cooperating in jobs, but the database, and the information in it, may be their only means of communication. Such communication is one-way, and is thus a weak form of collaboration, but often important. For example, casework files are a central mechanism for communication and cooperation in many areas from taxation to social work. However, the facilities for cooperation in a typical shared file store or database are limited to locking, and even that may be rudimentary.

People may also explicitly pass documents, produced by single-user systems, between themselves. For example, the cooperative use of spreadsheets has been studied. Some of this involves close working between the participants, but some users simply pass the spreadsheet data between them with little, if any, comment. If you change a formula, then the intent is obvious – you thought it was wrong.

Of course, in most situations, direct communication is necessary as well, especially where there is some conflict (your colleague thinks the original formula was right). The lesson from these more extreme examples is that cooperation does not necessarily involve direct communication and, even where it does, the indirect channel through the artifact may be central to effective working.

One shared database which is explicitly designed to promote cooperation is *Liveware* [383]. This is a card-index-type database implemented over HyperCard, designed to be updated by users at many sites, and spreading the information by floppy disk as people meet during normal social contact. When different versions of the Liveware database 'meet', they choose the newest version of each record. Each record has a single owner, so problems of conflicting updates are avoided.

Lotus Notes

The most widely used groupware application, excepting email, is Lotus Notes. Notes has aspects of a structured message system, a bulletin board and a shared database. At its simplest a Notes database consists of a collection of items. Each item belongs to a particular type and depending on the type has different named fields – just like a structured message system. In addition, each item may have responses and responses to responses, very like a bulletin board. The distinguishing feature, however, is its ability to replicate and reconcile replicas. There may be several copies of a single Notes database lodged on different servers. Users may connect to different servers and add or edit items and responses. Later, when the servers connect via network or modem, they update each other taking the most up-to-date version of each item just like *Liveware*. This has made it particularly popular with users of portable computers working away from their central office.

Although Lotus Notes has been very successful, much of its use has been simply for email. However, it is also customizable and programmable so many bespoke applications have been produced using it as a base platform. In particular, it can be used to implement or support *workflow* systems.

Notes' key market is in corporate information and messaging systems. However, it is being challenged in this market by web-based *intranet* solutions, where the web's cross-platform and cross-vendor compatibility makes it an ideal candidate. Clearly aware of this threat, Lotus (now part of IBM) has added web publishing features to Notes.

19.6 FRAMEWORKS FOR GROUPWARE

In this section we will discuss several frameworks for understanding the role of groupware. One use for these is as a classification mechanism, which can help us discuss groupware issues. In addition, they both suggest new application areas and can help structure the design of new systems.

19.6.1 Time/space matrix and asynchronous working

First of all we will look again at the time/space matrix. At the beginning of Section 19.2, we placed familiar technologies in the matrix. Figure 19.9 shows the same matrix, but populated with the groupware systems we have discussed.

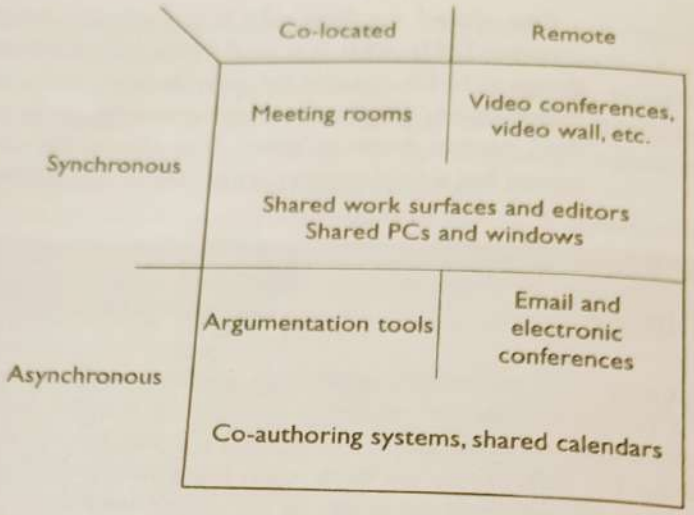


Figure 19.9 Groupware in the time/space matrix

This matrix has become a common language amongst the CSCW community. It can also be useful during design as one of the earliest decisions is what sort of interaction you are planning. The design space for *synchronous* interaction is entirely different from that for *asynchronous*.

However, the synchronous/asynchronous distinction is not as simple as it at first seems. In Section 19.2 we simply said 'whether [the participants] are operating at the same time or not'. However, for an email system, it makes no difference whether or not people are operating at the same time. Indeed, even in Figure 19.1, when we classified letter writing as different time/different place, a similar objection could be brought.

The difference between email systems and (most) co-authoring systems is that the latter have a single shared database. Thus when people work together, they know they are working together, and, depending on the locking regime used, can see each other's changes. An email system, on the other hand, may take some time to propagate changes. Perhaps, a better distinction is to look at the data store and classify systems as *synchronized* when there is a real-time computer connection, or *unsynchronized* when there is none.

For unsynchronized systems it makes little difference whether or not the participants are operating at the same time. Also location is not very significant. (A co-located unsynchronized system is possible; imagine two computers in the same room with no network, which are periodically brought up to date with one another by floppy disk transfer.)

If we consider synchronized systems, then the actual time of use becomes more important. If the participants are operating at the same time (*concurrent access*), we have real-time interaction as seen in meeting rooms (co-located) or video conferences (remote).

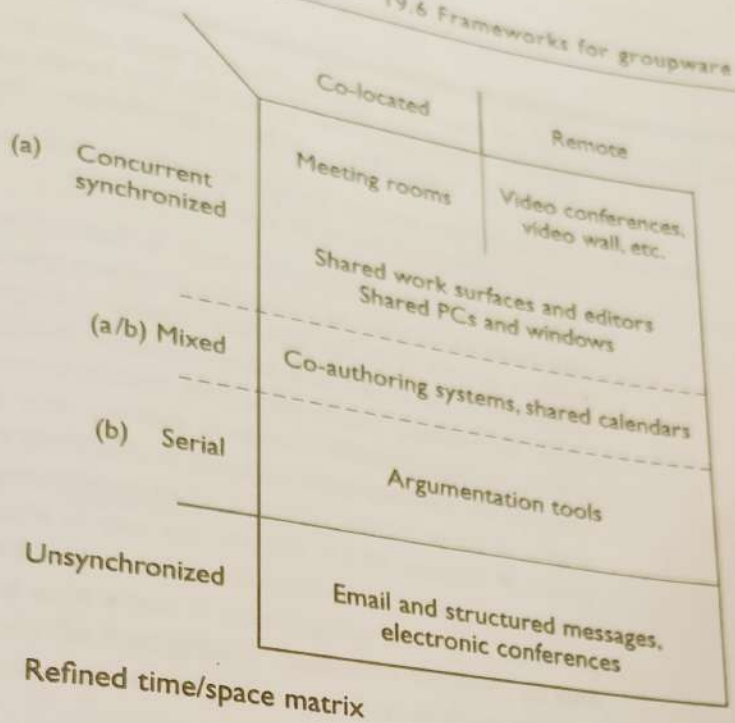


Figure 19.10 Refined time/space matrix

Alternatively, the system may prevent users working at the same time, by large-scale locks, leading to non-concurrent synchronized working. Earlier argumentation tools fall into this category. Because the participants are forced to use the system one after another, we can call this *serial access*.

Finally, co-authoring systems like Quilt [210] allow fine-grained locking so that participants can use the system at the same time or not. Therefore, they allow both serial and concurrent synchronized access.

Figure 19.10 places the groupware systems into this refined matrix. This matrix is not widely used, but is far more accurate for placing a prospective design. This is particularly obvious when one considers systems designed for users of mobile computers or home computers. These may be totally unconnected to central computers, except by occasional direct connection, floppy disk transfer or dial-up modem. The only existing groupware systems which could support such users are email and message systems, and *Liveware*. However, even email systems are rarely available except when connected via a modem. Liveware is the only groupware system *designed* for unsynchronized cooperation.

Systems, principally co-authoring tools, which describe themselves as operating in both synchronous and asynchronous modes are normally in class (a/b) in the diagram. That is, they have centralized data and allow users to operate in closely coupled modes like a group editor, or one at a time. No current system allows fluid access across the synchronized/unsynchronized divide. The authors of this book are operating from different sites and different continents and all often work at home. So we can vouch for the need for such systems.

Worked exercise *Find out how many different forms of direct computer-mediated communication are available on your system (start with email). Are they heavily used, and if so, where do they fit in the time/space matrix (Figure 19.9) and its refinement (Figure 19.10)?*

Answer Obviously the answer to this depends on your own particular facilities, but here are some suggestions based on the authors' sites.

The question says 'start with email'. In fact, on many systems several email interfaces are available. On one of the authors' systems there are two command line interfaces as well as various GUIs for different window managers (under X, on the Macintosh, etc.). Although email is just one communication mechanism, it is used in a wide variety of ways. Its 'normal' use is essentially asynchronous/remote, or in terms of the refined matrix, unsynchronized. However, delivery over a LAN is often virtually instantaneous and you can observe users engaging in effectively synchronous communication. The possibility of these modes of use is often determined by the style of the interface, as was shown in Section 18.2.4. Clearly, the way in which new mail is notified and the speed with which mail can be read both determine the pace at which the conversation can proceed. A further complication in some sites is that internal and external email have entirely different interfaces: sending mail off site may involve remotely logging into a gateway machine and then using some obscure mailer there, rather than the familiar workstation-based tools.

In addition to email, you may have one or more text-based synchronous communication tools. For example, many UNIX machines have two tools, 'write' and 'talk'. The command 'write Alison' followed by some text prints that text on Alison's screen. She can respond by doing a 'write' back to the sender. The 'talk' command establishes a more continuous conversation, as described in Section 19.3.1, where both participants' screens are split in two, each half displaying one of the participants' contributions. Although both can be seen as synchronous/remote, they operate at different levels of *sharing* (Section 19.6.2). Whereas 'write' only sends the text after the contribution is completed, 'talk' sends each user's contributions character by character as they are typed. That is, 'talk' sits further down toward the bottom left of the diagram in Figure 19.11. Note also that 'write' can be used in a semi-asynchronous mode. If the other user is logged in, but not at his terminal, the message waits there until his return.

Another asynchronous communication tool commonly available is bulletin boards, such as Usenet News. As discussed in Section 19.3.1, these have different properties from email, being rather less personal.

In addition to these standard tools, some of the authors' colleagues have permanent video connections between their offices. You may have similar facilities, or other computer-augmented communications, perhaps video in a window on your workstation, or possibly CUSeeMe. Don't forget apparently low-tech things such as computers which automatically dial telephone calls.

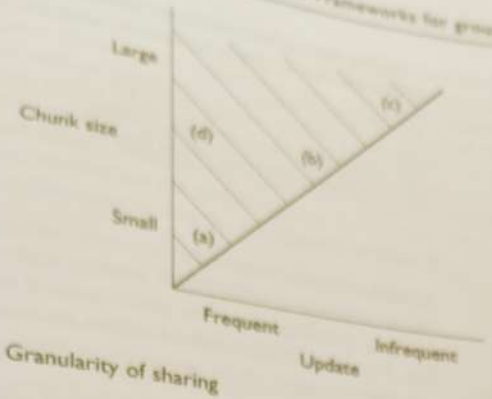


Figure 19.11 Granularity of sharing

19.6.2 Shared information

Electronic conferences and shared workspaces share information primarily for communication, whereas a document is shared for the purpose of working. Both raise similar issues concerning the degree of sharing required.

Granularity

The groupware systems we have seen differ as to the *granularity* of sharing they allow, in terms of both object chunk size and frequency of update.

Looking first at object chunk size, some systems operate at a very fine grain, allowing participants to edit the same sentence, or even the same word in a sentence. At the other extreme, shared file systems may often have locks so that only one user can edit a file at the same time. The granularity here is the document. The majority of groupware systems, in particular argumentation and co-authoring tools, operate somewhere between these extremes. They have some idea of a node or a section, which only one person can update at once, but which is significantly smaller than a whole document.

In the time dimension, systems may show participants' updates to one another immediately, within seconds, or when the user has finished editing the chunk. It is usually the case that a fine-grained chunk size requires fine-grained updates. For example, if you are allowed to edit the same word, it is not helpful if it takes a minute to see your colleague's typing! However, the converse is not necessarily true. Some systems operate locks on largish chunks but show other participants the updates immediately. Examples of this are some meeting rooms which have long-lived floor holders. This means that groupware systems all reside in the hatched region of Figure 19.11.

In the figure, four points are plotted representing different grain size choices, as typified by the examples we have discussed:

(a) shared editors
(b) co-authoring systems like Quilt
(c) network file systems with locking
(d) meeting system with floor holder.

Levels of sharing

As well as varying in terms of how much is shared, systems vary as to what is shared. At one extreme are WYSIWIS systems, such as shared window systems and many meeting rooms. In these, the participants all see exactly the same presentation of the data. However, in other shared editors, such as *Grove* [121, 122], the participants can edit different parts of the document at once. That is, they share the object, but not the presentation.

There is an interesting middle ground which is rare in explicit groupware, but common in the use of shared databases. That is, sharing a view, but not the presentation of that view. For example, two people may be viewing the same part of a database, but one person sees it presented as a graph, and the other in tabular form. We thus have three levels, as depicted in Figure 19.12.

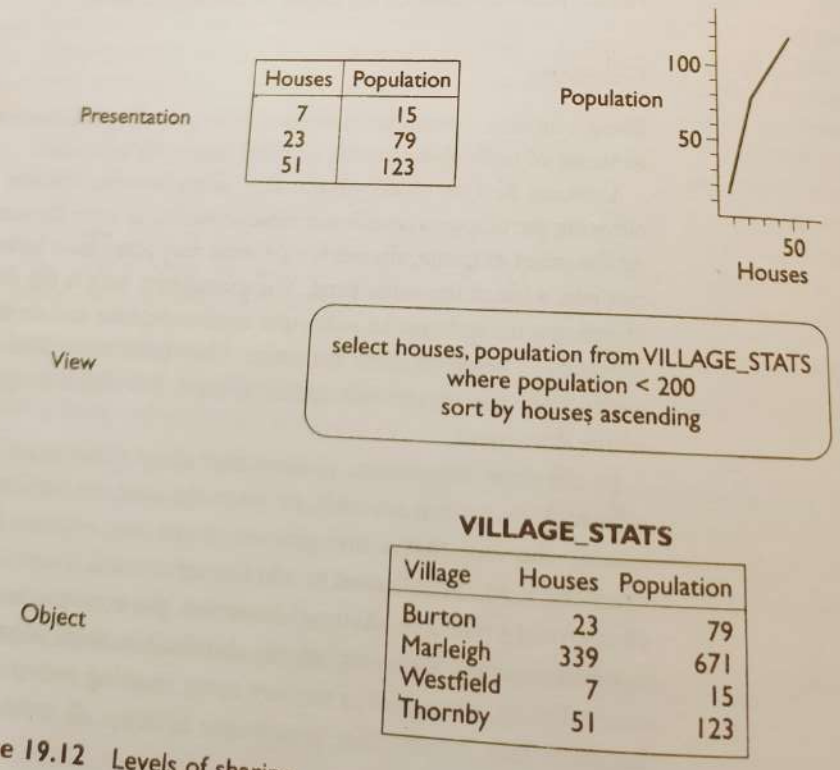


Figure 19.12 Levels of sharing

In addition to this output-oriented sharing, we can also look at input sharing. On one hand there are those systems which have a single shared virtual keyboard, for example the shared window systems. On the other, we have the majority where the participants can input at different places. This can be characterized as single versus multiple insertion points. There is no real middle ground here, but for those with separate insertion points we have the issue of visibility: whether or not the participants can see each other's insertion points or mouse pointers. Furthermore, if the other participants' cursors are not visible, we may have a *group pointer*, as discussed in Section 19.4.2. This gives us four levels of input sharing:

single insertion point
multiple insertion points

- shared virtual keyboard
- other participants visible
- group pointer
- no visibility

Again, there are loose connections between the two levels of sharing. For example, it makes little sense to have a single insertion point but different views. However, one document annotation system has separate insertion points but a shared view. Any user can choose to scroll the view of the document, but it then scrolls for all users. To make matters worse, the other users' insertion points stay at the same point on the *screen* as the document moves. So if they are typing when the screen is scrolled, their characters appear all over the document! It is a testimony to the power of social *protocols* that this system is not only used successfully, but also enjoyed.

Types of object

The kind of object or data we are cooperating over obviously affects the way we share them. This is particularly important in the *unsynchronized* case or where there is a danger of *race conditions*. That is, where two participants perform updates simultaneously and there is confusion as to which comes first.

Consider first the text of a shared editor. Participants can add, edit or delete text anywhere in the document. We thus have to worry about them interfering with one another, for example one participant deleting the text that another participant is in the middle of editing. Contrast this with a linear text transcript, as produced by some electronic conferencing systems. The transcript is *monotonic*, that is one can only add to it, never take away, and *appending* contributions are always added to the end. This makes the job of handling updates much easier. Every time a participant completes a contribution, it is simply added to the end.

However, the text transcript is inherently *sequenced*. This makes it best suited to *synchronized* groupware. If we imagine two distant sites, with no fast communications, it is difficult to keep the transcripts similar. If each site adds any new items to the end, the sites will show different transcripts. Imagine we have a user, Alison, at site A, and a user, Brian, at site B. The transcript has two contributions 'a1 b1'. Alison and Brian both make a contribution at the same time, say 'a2' and 'b2' respectively. So site A sees 'a2' first and site B sees 'b2' first. After a while the contributions are transmitted from site to site, leaving at site A the transcript 'a1 b1 a2 b2' and at

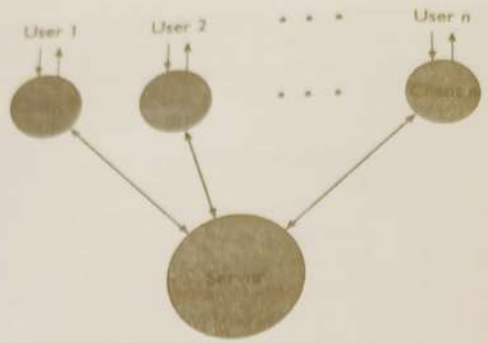


Figure 19.16 Client–server architecture

The second major architecture is *replicated*. Each user's workstation runs its own copy of the application. These copies communicate with one another and attempt to keep their data structures consistent with one another. Each replicate handles its own user's feedback, and must also update the screen in response to messages from other replicates. The intention is often to give the impression of a centralized application, but to obtain the performance advantages of distribution.

Compared with a client–server architecture, the replicated architecture is difficult to program. In the last paragraph we deliberately wrote '*attempt* to keep their data structures consistent' as this is a major problem. If two users, say Jane and Sam, hit a key almost simultaneously, then Jane's workstation will process Jane's keystroke first, and Sam's second, but Sam's workstation will process them the other way round, and yet they must give the same result – help!

This race condition is a common problem in distributed computing. A standard solution there is to *roll back* one or other replicate and re-execute the commands. However, if the results have already been displayed on the user's screen this is not acceptable – standard computing algorithms often fail for groupware. Happily, many of the *concurrency control* mechanisms, such as locking or floor holders, mean that such races do not occur, or at worst occur only when users obtain locks or other large-scale events. So, when rapid feedback is not required, standard mechanisms may be applied, but, for real synchronous update, special-purpose algorithms are required.

The main advantage of a replicated architecture over the client–server is in the local feedback. However, the clients are often not completely dumb and are able to handle a certain amount of feedback themselves. Indeed, the server often becomes merely a central repository for shared data with the clients having most of the application's functionality. On the other hand, a replicated architecture will rarely treat all the replicates identically. If a user tries to load or save a document, that action does not want to be replicated. Either one of the replicates is special, or there is a minimal server handling movement of data in and out of the system. So we see that there is a continuum between the client–server and the replicated architecture.

Figure 19.17 Single-user application

19.7.3 Shared window architectures

Shared window systems have some similarities with general groupware architectures, but also some special features. Recall that there is a single-user application which is being shared by several participants on different workstations. The single-user application normally interacts with the user via a window manager, say X (Figure 19.17). The shared window manager works by intercepting the calls between the application and X.

Where the application would normally send graphics calls to X, these are instead routed to a special application stub. This then passes the graphics calls from the X library (Xlib) on to a user stub on each participant's workstation. A copy of X is running on each workstation and the user stub passes the graphics calls to the local copy of X. Similarly the users' keystrokes and other actions cause X events which are passed to the user stub and thence through the application stub to the application (Figure 19.18). In fact, the nature of X's own client–server approach can make the user stub unnecessary, the application stub talking directly to an X 'server' on each workstation.

The input side has to include some form of floor control, especially for the mouse. This can be handled by the application stub which determines how the users' separate event streams are merged. For example, it can ignore any events other than those of the floor holder, or can simply allow users' keystrokes to intermingle. If key combinations are used to request and relinquish the floor, then the application stub can simply monitor the event streams for the appropriate sequences. Alternatively, the user stub may add its own elements to the interface: a floor request button and an indication of other participants' activities, including the current floor holder.

The problem with a client–server-based shared window system is that graphics calls may involve very large data structures and corresponding network delays. One can have replicated versions where a copy of the application sits on each workstation and stubs communicate between one another. But, because the application is not *collaboration aware*, problems such as race conditions and reading and saving files become virtually intractable. For this reason, most shared window systems take a master–slave approach where the application runs on the first user's workstation and

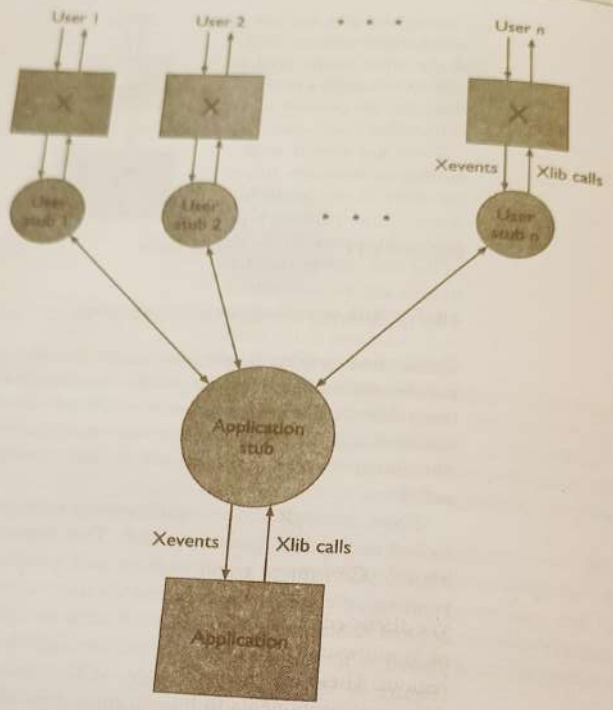


Figure 19.18 Shared window architecture

subsequent users get slave processes. The delays for other users are most noticeable when starting up the application.

19.7.4 Feedthrough and network traffic

We have discussed the necessity of rapid *feedback* to the user who performs an action, and we have seen how replication or partial replication can solve this. However, we must also worry about *feedthrough*, the reflection of one user's actions on the other users' screens. The requirements for feedthrough are not quite so stringent as for feedback, and this can be used to reduce the amount of network traffic. There is little difference in this respect between client–server and replicated architectures, so, for the sake of argument, we will assume a client–server architecture.

Imagine a user has just typed a character. The character appears on the user's screen, either through local feedback or after an exchange with the server. However, all the other clients need to be informed also. That is, with n participants, each user action causes a minimum of $n-1$ network messages. If this is repeated for each character, the network will grind to a halt with the number of messages. Just think of the effect if we wanted to send out updates for each mouse movement!

There are several ways out of this, the common thread being to reduce the number of messages. In principle, one could send a single *broadcast message* to all the other clients, as the information is the same for all of them. Unfortunately, in practice, many network protocols only support point-to-point messages. A more successful approach is to increase the *granularity* of the messages. The rapid feedback to the user who initiated an action is necessary, but the feedthrough to other users may be able to cope with less frequent updates. The server can save up several characters' worth of updates and send a single message. We thus only need to send the $n-1$ messages to the clients on each chunk.

The choice of this chunk size can be crucial to the success of the system. If the participants can talk to each other (either co-located or with audio/video channels) they will refer to the contents of their screens. In such cases, lags of more than a few seconds can be disastrous. However, even if the computer is the only communications medium, the chunk size has an effect as the gradual appearance of text is an indication of other people's activity.

19.7.5 Graphical toolkits

We discussed in Chapter 8 some of the *widgets* one finds in a typical graphics toolkit or window manager, such as menus, buttons, dialog boxes and text and graphics regions. These are useful for creating single-user interfaces, and one would like to use the same components to build a groupware system. Unfortunately, the single-user assumptions built into such toolkits can make this very difficult.

Some widgets may take control away from the application. For instance, a pop-up menu may be invoked by a call such as

```
sel = do_pop_up("new","open","save","exit",0);
```

The call to `do_pop_up_menu` constructs the pop-up menu, waits for the user to enter a selection, and then returns a code indicating which choice the user made (1 for 'new', 2 for 'open', etc.). Of course, during this time the application cannot monitor the network. This can be got round, by careful programming, but is awkward.

More fundamentally, the functionality of toolkit widgets may be insufficient for groupware. This is particularly obvious for text areas. The toolkit often takes over a lot of the tedium of handling an editable text region: the user can type and delete, do cursor movement and even cut and paste, all without the application's intervention. However, the groupware developer may want to have multiple cursors, or to ensure that all the participants can see the same portion of a document. Unfortunately, even information such as what portion of the document is in view, or where a particular

logical character is displayed on the screen is difficult to come by, as is control over a scrollbar if this is to be shared. One is often forced to design the application round the limited capabilities of the text widget, or to use bitmap operations to paint the text oneself.

Despite the difficulties of toolkits, some have facilities which are a positive help. For example, the SunView toolkit and its X version, XView, are both *notification based* (see Chapter 8). As well as notifications of user events, they can be asked to tell you when a network channel is ready for reading or writing. Thus, the interface and the network can be programmed in a similar style.

19.7.6 Robustness and scaleability

If you are producing a shared application to test an idea, or for use in an experiment, then you can make a wide range of assumptions, for instance a fixed number of participants. Also, the occasional crash, although annoying, is not disastrous. However, if you expect a system to be used for protracted tests or for commercial production then the standards of engineering must be correspondingly higher. Four potential sources of problems are:

1. Failures in the network, workstations or operating systems.
2. Errors in programming the shared application.
3. Unforeseen sequences of events, such as race conditions.
4. The system does not scale as the number of users or rate of activity increases.

A full description of these problems and their solutions is, of course, beyond this book. For this, the reader should turn to texts on software engineering, real-time and distributed programming. Furthermore, the details will differ between networking software, operating systems and other software support. However, there are a few general remarks and specific issues which we will discuss.

To some extent, the above problems are common to single-user systems: hardware problems, bad programming and the like. Indeed, most commercial software the authors have used suffers the occasional (or frequent) crash. However, there are factors which make multi-user systems more fragile. The large number of different hardware and software components of a multi-user system means that a fault of type 1 is more likely to occur. Also, the complexity of the algorithms used in groupware makes a fault of type 2 more likely. In both cases there is a danger that the consequences of the failure will propagate throughout the system. If a single failure crashes the whole system, it probably will not be used twice! Interleaving of different users' actions and the unpredictable effects of network delays increases the chance of errors of type 3. Finally, system development and testing may involve only two or three people, and thus hidden assumptions about the number and activity of users may not become apparent until the system 'goes live'. So, faults are more likely and the effects are far worse – instead of losing one user's document or datafile (sad though that might be), a groupware crash can destroy the work of a whole team of people.

Server faults

The most obviously disastrous problem in a client–server-based system is a server crash, whether hardware or software. Fortunately, this is most amenable to standard solutions. Most large commercial databases have facilities (such as transaction logging) to recover all but the most recent changes. If the groupware system is not built using such a system then similar solutions can be applied; for example, you can periodically save the current state using two or three files in rotation. The last entry[2] in each file is the date of writing and the system uses the most recent file when it restarts, ignoring any partially written file. Remember, though, that this might mean your server going 'silent' for a few seconds each time it saves the file – can your clients handle this? In really critical situations one can have multiple servers and copies of the data, so that a backup server can take over after a crash of the primary server.

Workstation faults

More often, individual workstations, or the programs running on them, will crash. This is partly because there are more of them, and partly because their code is more complex. In particular, these programs are handling all the user's interactions and are built upon complex (and frequently flaky) graphical toolkits. Of course, one tries to program carefully and avoid these errors, but experience shows that they will continue to occur. The aim is to confine the fault to the particular user concerned and to recover from the fault as quickly as possible. When thinking about a client–server architecture, there are three 'R's for the server:

Robust A client failure should not destroy or 'hang' the server. In particular, *never* have the server wait for a response from the client – it may never come. The server should either be *event driven* or *poll* the clients using non-blocking network operations.

Reconfigure The server must detect that the client has failed and reconfigure the rest of the system accordingly. The client's failure can be detected by standard network failure codes, or by timing out the client if it is silent for too long. Reconfiguring will involve resetting internal data structures, and informing other participants that one of them is unavailable and why. Do not let them think their colleague is just being rude and not replying!

Resynchronize When the workstation/client recovers, the server must send sufficient information for it to catch up. A server may normally broadcast incremental information (new messages, etc.), so make sure that the server keeps track of all the information needed to send to the recovered client. This is very similar to the case of a new participant joining the groupware session.

2 A crash might happen in the middle of the save.

Replicated architectures have similar issues. Imagine we have three replicates A, B and C. Replicate A has crashed, so both B and C must detect this and correct their internal data structures accordingly. An additional problem is that one replicate, say B, may detect the failure before the other. So, C may send B a message which refers to A in some way – B must deal with this situation gracefully.

Algorithm faults

Some application failures do not crash the application and may therefore be more difficult to detect. For instance, data structures between replicates or between client and server may become inconsistent. Obviously this *should not* happen, if the algorithms are correct. Indeed, such critical algorithms should be given the closest scrutiny and perhaps proved correct using formal methods (see Chapter 17). However, one should be prepared for errors and, where possible, include sufficient redundancy and sanity checks in the code so that inconsistencies are at least detected and, ideally, corrected.

Such defensive programming mechanisms, against hardware and software errors, may be very expensive in programming effort and execution time. For experimental systems they may be excessive. A minimum requirement for all but toy systems is some form of reset, which forces the system to resynchronize all its data structures between clients and server or between replicates; hopefully, losing none but the most recent updates in the process.

Unforeseen sequences of events

Distributed programming has many problems of which possibly the most well known is *deadlock*. This is when two (or more) processes are each waiting for the other to do something. A common scenario is where process A is trying to send a message to B and B is trying to send one to A. Because A is busy trying to send, it does not want to receive a message from B, and vice versa. The possibility of deadlock can often go undetected during testing because of operating system and network buffers. A's message to B is stored in the operating system's buffer, so A can then read B's message. Unfortunately, as load increases, one day the buffer is full and deadlock can no longer be avoided – this may only happen *after* the system has been released.

The first rule to avoid deadlock is *never* to use blocking input or output. That is, always use network calls which time out, or return immediately if the operation cannot continue. Use of an event-driven programming style can also help, as does the use of constructs such as the UNIX 'select' system call, which monitors several communication channels simultaneously.

At a higher level, one should also avoid making assumptions about the ordering of incoming events. This is also important to avoid problems with race conditions. For example, if a client process has sent a message to a server requesting information, it should *not* assume that the next message from the server will answer the query.

A common assumption in groupware programs is that messages sent from one computer arrive in the same form at another. This depends very much on the particular protocols used. Consider UNIX stream-based sockets, commonly used in experimental groupware applications. The communication paradigm is not packet based, but of a character stream. Imagine a client sends two messages to the server: The first is 26 characters long, 'abcd . . . xyz', the second is 10 characters long, '0123456789'. When testing such a system it is very likely that the server will read two messages of 26 and 10 characters, exactly as sent, but this is not guaranteed. It is perfectly possible to get one message of 36 characters, 'abc . . . xyz01 . . . 89' or even one of 10 and one of 26, 'abcdefghij' and 'klm . . . xyz01 . . . 89'. The solution is easy: one always uses fixed-length messages, or alternatively you code the message length into the message header. The recipient can then reconstruct the messages from the byte stream. The difficulty is that the problem can go undetected for a long time and then cause a major disaster. Read your network manual very carefully and assume nothing.

Scaling up

In general, the most certain way to avoid algorithmic errors is to use simple algorithms: tables rather than complicated data structures, fixed-length fields for names and messages. These all go toward reducing the likelihood of some of the faults described earlier, and are the recommended techniques for prototyping an application. Unfortunately, these are precisely the programming techniques which frustrate the scaling up of the system to larger numbers of users, or greater loads.

As the system develops, the initial algorithms and data structures will need to evolve. This will be easier if the future scaling of the system is taken into account at the beginning. Good software engineering practice helps. For example, if the message passing is in a separate module then an initial design can use fixed-length and textually coded messages (for ease of debugging). But, when the throughput of the system increases, the message passing can be easily changed to variable length messages, perhaps buffering several messages together (to reduce network traffic) and binary coding of the messages (for efficiency).

Where fixed-size assumptions are made in early versions, these should be documented: 'the present design only caters for up to six users'. Even more important, the system ought to detect when these bounds are broken and behave sensibly. In programming terms, this tends to mean checking array bounds, rather than scribbling randomly over memory! Sometimes these bounds are known in advance, but always try to encapsulate these decisions so that if the bounds do need to change you know where in the code to find them.

A particular problem for the server is that the operating system may limit the number of open files/network connections at any time. For example, let us suppose the limit is 16 files and suppose the server is using one for the application datafile and one for logging. This means that at most 14 clients can be connected at once. Even worse, say the server periodically opens the datafile to save data, but closes it between

times. Unless the server itself keeps the number of clients below 14, it may be that 15 clients get connected, which, including the logging file, saturates the server's allowable files. Now, when the server comes to save the data, it cannot open the datafile. Obviously, the groupware writer must at least be aware of these limits in order to prevent such a disaster.

There are a few solutions to this problem. The operating system limit can be altered; for example, in MS-DOS this involves changing the 'FILES=' line in the file 'CONFIG.SYS'. However, this is more difficult under UNIX, and in general the limits and how to alter them are very system dependent. The server can avoid the use of permanent network connections and instead use *datagram protocols*, where the client and the server are only connected while a message is being exchanged. Finally, there may be more system-dependent 'tricks', for example forking extra servers under UNIX.

Testing for robustness

Often the functionality of an application is tested by having several windows on the same workstation, each acting as a different 'user'. Unfortunately, this is unlikely to catch the sort of problems discussed above as it is impossible to type simultaneously into two windows. A more violent approach is required; in fact, the general rule is . . . mistreat it.

Crashes and major faults can be simulated. Try rebooting a workstation or pulling out a network connector.[3] To be slightly more gentle, you could simply kill a client process and see the effect on the server (or on replicates).

Similarly, you can simulate race conditions and odd sequences by running the system between two workstations and then hitting keys on them simultaneously. A little bit of knowledge of the system will suggest the best combination.

Random input may crash your system. Push it hard at several levels. Have a group of colleagues on different workstations type and hit mouse buttons as fast as they can – but log the keystrokes as you may want to recreate the resulting situations for later debugging. Create a rogue client/replicate, which sends random, but correctly formed, messages to the server or other replicates. Alternatively, this can be arranged without network communications by building a test harness round a single process. A similar, possibly less fair, approach is to send random data down the network at a process.

Finally, the real acid test. Offer a group of computer science undergraduates a drink each if they can break the system – you will lose your money, of course.

Unfortunately, discovering you have a fault is only the first step – correcting it is more difficult still. Modular and defensive programming and logging of communications so that errors can be recreated are a good beginning, but experience and hard work are the final answer.