17

Distributed and Collaborative Interaction

There are many cases where the user and the interactive application are on different machines. This requires that the user interface be distributed from where the application is running to where the user is working. This situation occurs frequently when server computing is involved. The days of operators wearing lab coats and occupying the same room with specially air-conditioned, noisy servers are gone. There is no need for humans to work in the same space with large computing. However, to achieve this, the user interface must be distributed from the server to the user's computer. The need for distributing the user interface also arises when providing support to users spread around a company or around the world.

Another reason to distribute the interaction is to support collaboration. People rarely work alone. They are almost always part of a team engaged in a larger enterprise. The ability to share work among team members and to work together is important. In many situations, coworkers are distributed geographically. As computing work has been distributed around the world the need for groups to collaborate has increased. Research by Kraut et. al.[1] has shown that a shared view of the work being done is more effective than collaboration solely over the phone or face-to-face videoconferencing. Such a shared view requires the distribution of the user interface among collaborators.

This chapter will first discuss the key aspects of collaborative software. This will provide a set of criteria for evaluating various strategies for distributing interaction. This will be followed by a discussion of the various points in the model-view-controller architecture where distribution can be implemented.

Collaboration: A brief introduction

Collaborative user interfaces is a very large field with numerous issues including the sociology of team work. In this chapter we will only cover the basic distributed interface techniques that are the foundation of most collaborative

systems. For additional reading the proceedings of the ACM Computer Supportive Cooperative Work conference (CSCW) is an excellent place to start. Before launching into the technology it is useful to discuss five major issues in collaborative interfaces. These are: types of collaboration, identity, access, awareness and device consistency.

Types of collaboration

Much collaborative work is either asynchronous, where various team members work at their own time with no tight coordination with others, and synchronous, where multiple team members are engaged at the same time on the same work product.

A good example of asynchronous collaboration is email. One can send a message, document, spreadsheet or engineering drawing to a coworker for comments and then move on to other work. The coworker deals with the message when it is appropriate to them. Asynchronous collaboration has huge advantages in that resources are not tied down while others are thinking, planning or obtaining other information and the overhead of synchronizing schedules among multiple people is eliminated.

A good example of synchronous collaboration is a phone call. All of the parties (if it is a conference call) are all simultaneously committed and involved. This can provide for a very rapid exchange of information and facilitates reaching a common understanding. Synchronous collaboration in meetings is notorious for wasting the time of some participants time while others focus on topics that are not universally useful.

Synchronous collaboration at the user interface is characterized by What You See Is What I See (WYSIWIS). The goal is for all participants to see and have access to the same interactive work product as they discuss, modify and approve whatever work product is under consideration.

Collaboration can also be characterized by the forms of communication. As with operating systems and many other concurrent systems there are message-based collaboration models and shared data models. As with other systems these two forms intermingle at various levels of implementation. Email and instant messaging are examples of message-based collaboration with email being asynchronous and instant messaging being mostly synchronous. There are systems like TeamRooms[2] that provide a common place for team information to be stored and various team members work on the information either synchronously or asynchronously as the work requires.

Identity

A very important part of collaboration is the identity of collaborators. This is the heart of the email spam problem where nobody knows for sure who is actually sending a message. An email address provides some measure of confidence about who will receive a message but almost no confidence in who sent it. In more synchronous collaboration instant messaging systems or collaborative internet games provide buddy lists and other mechanisms for locating potential collaborators. Identity is a huge problem for collaborative activity, but its solution lies outside of the user interface architecture and will not be addressed here.

Access

Once you know who the potential collaborators are, there is the question of how to grant access to the shared work product or user interface. Some people deal with the email spam problem by restricting the set of people who can send messages to them. You may also want to share your network configuration screen with computer support staff, but not the contents of the document that you are editing.

Providing access is a user interface design problem. Most systems provide access control as a hidden, special configuration option. The average user has only limited awareness of what has been shared and must look several layers deep to determine who can access the shared information. Older computing systems assumed that all users resided on the same machine and that all users were programmers. Such systems provided access to the individual user, a single group to which the user belongs or to everyone in the world. This means that to share with someone outside of your group you must share with everyone. This does not in any way reflect the practices of real collaborations.

On shared servers, access was managed on the file level and access privileges were visible in the file system view. Most personal computers began with the assumption that only one user was on the computer and therefore access rights were not important. Most such systems do no show access rights in a consistent and understandable way in spite of the fact that the majority of PCs are connected to the internet and are involved in world-wide collaborations.

It is also not obvious to the average user what the implications of sharing might be. On some systems one can share a file with another, but if the folder containing the file is not also shared, access is not truly granted. This may make sense to the implementers of the operating system but it makes no sense to the

average user. A frequent response is to open all folders from the desired file up to the root of the file system so that access is assured. This usually leads to exposure of many more items than the user intended. The implications of such a move are not visible to the user and therefore the cause of frequent errors. Most operating systems including all of the major ones in 2005 have serious usability problems with their control of shared access.

Within a single application, access manifests itself in the issue of *floor control*[3]. When more than one user is accessing an interface it becomes confusing if all can make changes at the same time. Many multi-user interfaces impose floor control mechanisms whereby only one user can make changes at a time. In a physical meeting floor control is frequently handled by a moderator that allocates speaking time to each participant or by whoever is at the whiteboard holding the marker. In distributed user interfaces such social mechanisms are less apparent and sometimes floor control features must be explicitly provided.

A final access problem is the input and output of the interface itself. In a distributed application both types of information must flow over the network and are thus exposed to a variety of snooping techniques. In particular the presentation may contain private information and the input stream may contain passwords being typed. Mechanisms such as the Secure Socket Layer (SSL) are important in protecting this data as part of a distributed user interface. One of the major problems with X-Windows is its unsecured protocol.

Awareness

In collaborative work awareness of who else is working on the same problem is very important. In many cases it is useful to know what changes they have made. The versioning systems mentioned in chapter 16 usually retain information about who made the changes. What is also needed is the ability to see in the user interface the places where changes have been made. Figure 17.1 shows change tracking from Microsoft Word. Figure 17.2 shows similar change tracking/awareness in Excel. Similar features are found in most source code control systems.

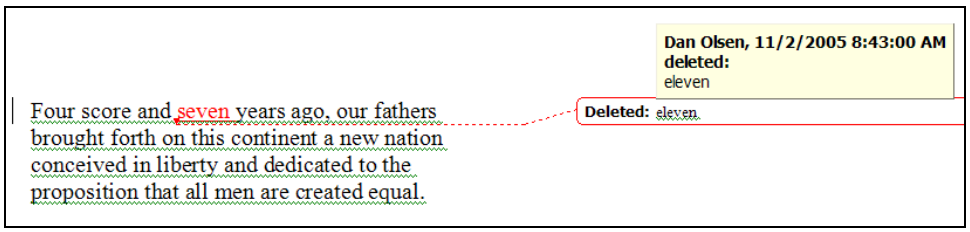

Figure 17.1 – Change tracking in Word

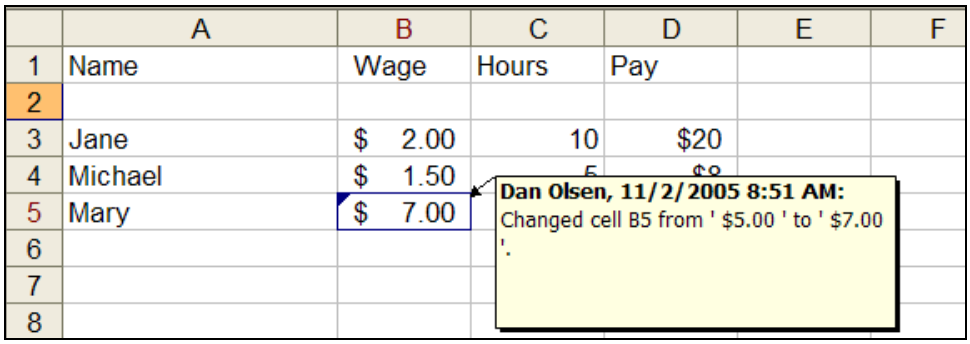

Figure 17.2 – Change awareness in Excel

Awareness is particularly important in the user interface because it is additional information that must be handled in the model or in the change history. As can be seen from figures 17.1 and 17.2 the nature of change awareness must adapt to the presentation of the particular application.

Coupled with awareness is the concept of change approval. In many collaborative applications the changes are not finalized until others have considered them. A corporate software build process and corresponding source code control processes have this property. Document management systems where the end document is a collaborative effort will also have this property. In preparing legislation there are many changes to a law which have no effect until the final vote.

Device consistency

A last issue with distributing interfaces among multiple participants is the consistency of the interactive devices. If two users are interacting with the same application and there are differences between those computers it becomes a problem for the application to consistently deal with both. Suppose the two computers had slightly different sets of installed fonts. This would make the

widths of words slightly different and would make translating mouse clicks into text selection inconsistent. If one computer has a much larger screen than another, compatible window sizes may be impossible leading to complications in sharing interfaces. There are also problems of installation. One computer may have OpenGL installed for 3D graphics and the other may not. The incompatibility will impact the ability to share interaction.

A more extreme yet very interesting case is when multiple users are on very different devices. Suppose that one user is working on a graphics workstation and the other on a PDA with a correspondingly small screen. The scenario gets more interesting if one user has a screen, but only voice input or perhaps no screen at all as in a cell phone. These differences make identical interactive styles impossible, yet the desire to collaborate remains. The creation of good interactive architectures that can accomplish this level of sharing is still an open problem.

MVC and distribution of the user interface

In the Model-View-Controller architecture there are several places where interface distribution can occur. Each has advantages and disadvantages and systems have been built for all of them. Figure 17.3 shows five different distribution schemes and their place in the MVC architecture. We will discuss each of these in turn.

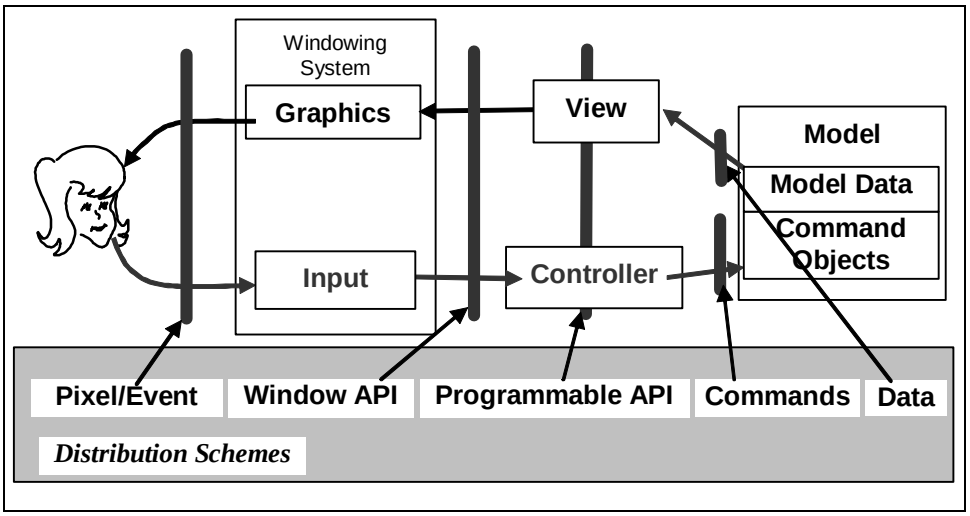

Figure 17.3 – Distribution of the user interface

In all of these schemes there is a *server* application and one or more interactive *clients*. For simple distribution of the interface for remote access there is only one client. For collaborative interactions there may be many clients.

Pixel/Event distribution

The simplest mechanism is the pixel/event distribution scheme. This scheme works at the screen level after the application has done its work. In this approach there is an interactive application that is operating normally. Special server code is enabled that monitors the pixels on the screen. Whenever changes occur on the screen the changed pixel regions are captured and sent over the network to each client. At each client the pixels are painted on the screen. Thus each client user sees the application exactly the way the user at the server sees it. A common implementation of this is Virtual Network Computing (VNC) [4]. VNC clients and servers generally use the Remote Frame Buffer (RFB) protocol. For a client there are only two commands: 1) paint this image at location (x,y) or 2) copy pixels from some rectangle to some other location. The first command is sufficient to perform all client display operations. The copy command provides more network efficiency when objects are dragging or windows are scrolling. Rather than package and send all those pixels, we reuse ones that were previously sent.

When input events occur at the server application they are handled as before. In this distribution approach the server application is unaware that there are interactive clients. When input event occur at the client, those events are packaged up and sent to the server over the network (generally encrypted). At the server the events are unpackaged and the server pushes those events to the windowing system as if they had been entered by the user.

Changes to the screen

A server's display system must capture the pixels from the display and forward changes to those pixels on to the clients. There are three main issues: 1) getting the pixels, 2) determining what has changed and 3) shipping changes to clients at a rate compatible with network speeds.

Getting the pixels is easy on most systems. Fundamentally a display card consists of pixel memory representing the displayed image. This memory can be read to obtain the currently displayed pixels. Most windowing systems provide such a method for screen capture so that server implementations need not know about the details of the display card.

Computing differences is a slightly harder problem. If the server saves a copy of the screen image last sent to the clients, a comparison can be made between the new pixels and the saved image. This comparison produces a set of changed rectangles that can be sent via RFB. This comparison, however, can have a significant cost. If a 1024x1024 window is being shared at 5 frames per second, then 5 megapixels per second must be compared. This can pose a significant load on some computers. This computational load is exacerbated if we try to use copy commands rather than just RFB image painting. To perform image copy we must take a second step of comparing all changed rectangles against the saved image to determine if any of the pixels can be reused in a move command.

The costs of computing differences can be sharply reduced if the windowing system will provide additional assistance. Remember that the interactive loop creates `damage()` calls for every part of the screen that the application wants updated. If the windowing system provides the necessary hooks to be notified of the damaged regions then there is no need to calculate difference regions by comparison. The damaged regions are the only ones that contain changes.

Network speed is a serious issue when distributing the user interface. Simply sampling the screen based on a timer will not work because it may flood a very slow network channel trying to present many intermediate steps that are not necessary. Many systems wait for a client request before computing the changes. If a client receives a change and then sends a request for the next change, this round trip effectively meters the network speed. Changes are accumulated and sent only as fast as the network will allow.

Client input events

In a simple distributed application with only one client and no actual user at the server the input event model is trivial. However, if there are multiple users then there is a problem. Suppose that user A pressed her mouse button to begin dragging while user B happens to jiggle his mouse. At the application the event stream would contain a mouse-down from A followed by move-move events from B but would not know that they are from separate users. The resulting application actions would be rather chaotic. This interleaving of events from multiple users is almost impossible to interpret correctly.

For pixel-based distribution the system generally enforces strict floor control. Each client has a special “floor request” button. When a floor request is received by the server, it waits for a good break point (generally when no mouse buttons are pressed) takes control from whoever has it, thus blocking that client from

sending input events and grants the floor to whoever requested it, thus enabling those events. With this mechanism the underlying application sees only one coherent stream of events.

Awareness

Awareness of what other users are doing can be a problem in these systems. It is tedious for a user to try to watch the entire screen to see where other users are making changes. In multi-user interaction there are frequently times when no inputs are being generated but the information on the screen is being discussed. To support discussion some systems provide each user with a telepointer[5]. A telepointer is generally a small icon or colored arrow associated with each user. Each user always has control of their telepointer so that they can point to information on the screen. The telepointer can also follow the user's input so that changes are readily visible to other users.

Advantages

The primary advantage of remote frame buffer collaboration is that it works with any application and without the developers of the application knowing that it exists. The clients are very simple to build and require little consistency among participating computers. Implementation of both client and server is also quite straightforward. As network speeds increase the universal nature of this approach shows great promise.

Disadvantages

The disadvantages revolve around the computing overhead required at the server and the network issues. The computing overhead on the server has been discussed previously. It should be noted that Moore's Law predicts continued increase in the speed of processors but will not increase the number of changing pixels that a user can want. This means that the computing resource issues will probably be solved by Moore's Law.

Networking issues are another problem. First there is the bandwidth required to ship images over slow connections. With dialup internet connections this can be a serious problem. However, there is not near as much movement in interactive applications as in video so the bandwidth problems are not as high as the screen size would indicate. Solving the video over the internet problem will probably solve these distributed interaction problems.

A more serious problem is network latency. The larger the internet, the more routers, switches and firewalls a message must go through. Though lots of bandwidth may be available the time for a message to make the round trip to the other side of the country may be a second or two. For web pages this is not an issue. The problem is in remote interaction. If a remote user grabs the scroll bar and drags it, the mouse move message must be sent to the server; the application must process the mouse-move, redraw the scroll-bar and redraw anything else that moved as a result. The image changes must then be sent back to all clients. If the remote user is trying to drag something and the round-trip on each mouse-move is 1 second the quality of interaction will be very poor.

A final disadvantage is that all users must interact in lock step with each other. Users are not free to move to different parts of the workspace when the work might call for it. Users cannot use any of their customizations that adapt the application to their own work style. Distribution across different modes of interaction is clearly impossible.

Graphics package layer

A second strategy for distributing the user interface is to capture the presentation calls at the graphics layer. As we learned in chapter 2, drawing is done through a Graphics object. The application code does not know what that object does with the information passed to the Graphics object. We can distribute the interface by capturing those calls, packaging them up into network messages and send them to remote clients. The remote client receives the drawing commands and reproduces them on the client screen. The input events are handled in the same fashion as the pixel distribution approach.

This technique has two advantages. The first is that there is no capture and analysis of screen pixels to locate change regions. This computational load is eliminated. The second advantage is that the graphics calls are generally much more compact than the pixels that they change. This reduces network bandwidth requirements.

This approach was first pioneered by the X-Windows[6] system. This system was invented to allow graphical user interfaces for applications running on different compute servers. X-Windows did not use the object-oriented Graphics object model. Instead they provided a procedure library for drawing in windows. This idea was later extended by Hewlett Packard into the Shared-X[7] system. In Shared-X the stream of drawing commands can be sent to multiple clients allowing for multi-user interfaces. X-Windows still survives as the windowing

system of choice on Linux systems. This graphics package distribution approach is found in Microsoft's remote desktop facility. Because Microsoft controls the API to Windows it is relatively easy for them to distribute those calls.

Advantages

This strategy has all of the advantages of the pixel-based approach without the computational cost of image differencing and with lower bandwidth requirements. If one has control of the graphics API this technique is relatively easy to implement.

Disadvantages

With the exception of speed and bandwidth issues, the approach shares most of the disadvantages of the pixel approach. Interaction in the presence of network latency is still a problem and all users are locked into a strict WYSIWIS view.

Chapter 17 is broken into two parts at this point to work around a Microsoft Word infinite repagination bug.

---

[1] Kraut, R. E., Miller, M. D., and Siegel, J., "Collaboration in performance of physical tasks: effects on outcomes and communication." *1996 ACM Conference on Computer Supported Cooperative Work (CSCW '96),* ACM Press, New York, NY, (1996), pp57-66.

[2] Roseman, M. and Greenberg, S., "TeamRooms: network places for collaboration," *ACM Conference on Computer Supported Cooperative Work (CSCW '96),* ACM (1996) pp 325-333.

[3] Boyd, J., "Floor control policies in multi-user applications," *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems* ACM Press, (1993) pp 107-108.

[4] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, Andy Hopper, "Virtual Network Computing", *IEEE Internet Computing*, Vol. 2 , No. 1, 1998

[5] Gutwin, C., Dyck, J., and Burkitt, J., "Using cursor prediction to smooth telepointer jitter." *International ACM SIGGROUP Conference on Supporting Group Work (GROUP '03)*. ACM Press, (2003) pp 294-301.

[6] Scheifler, R. W. and Gettys, J. "The X Window System," *ACM Transactions on Graphics*, vol 5(2), (April 1986).

[7] Garfinkel, D., Gust, P., Lemon, M., and Lowder, S., "The SharedX multi-user interface user's guide, version 2.0," , HP Research report, no. STL-TM-89-07, Palo Alto, California, 1989.