Design Issues for an Adaptive Mobile Group Editor

by

Tara J. Whalen

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 1997

©Tara J. Whalen 1997

Abstract

Software that supports collaboration (groupware) is rapidly gaining popularity, and many people wish to work together while using mobile computers. Mobile collaboration is made difficult by the characteristics of a wireless channel. The quality of a wireless connection can vary widely and is often of poor quality, characterized by low bandwidth and frequent disconnections.

Experiments with Calliope, a multi-user shared editor, indicate a number of application-level changes that allow more effective use of a wireless channel. Chief among these changes is the addition of adaptability, which allows the editor both to take full advantage of favorable network conditions and to handle difficult conditions as they occur. These experiments also indicate which network conditions are most relevant to groupware, information that will be used in a wireless network monitoring system that supports several types of adaptive applications.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The development of software that supports collaboration (groupware) has allowed people to work as a group on such projects as the writing of a document, even if they are separated in space or time. The specialized interface and communication tools provided by this software are especially useful in supporting a shared workspace for groups that have no way to meet in the same physical space. Thus, collaboration can now occur in situations where it would have been impossible but for this computer support. This opens up opportunities for such areas as business communication and academic collaboration.

With the recent rise in wireless and mobile computing, it is conceivable that some collaborators will be communicating over a wireless link. Portable computers and wireless networks have allowed people to have more "location independence" in accessing computing resources. Thus, it would be helpful to extend this independence to groups that cannot use a wired network. This situation can arise if a group wishes to hold an ad-hoc meeting whose expediency prevents waiting for a network to be installed (e.g. a meeting in an airport), or to collaborate in locations where a wired network cannot be

physically installed (e.g. on a beach).

Although this idea is promising, many collaborative applications are designed to use the rich capabilities of a wired network. Collaborative work often requires low response times and group interface features to support distributed interaction, but these requirements place heavy demands on a wireless link. A wireless connection is often poor, with low bandwidth and frequent disconnections. If network traffic becomes too heavy, users can be subjected to extensive delays or connection loss. Thus, there is a trade-off between extensive features and short response times. To reduce the delay, the software must be able to scale back its network resource consumption, a fact which leads to a less powerful, but fully operable, version of the groupware application.

Another factor to be considered is that the quality of a mobile connection can vary greatly. A mobile user can plug into a wired network, or may move into an area with a better wireless channel. In these cases, the user would likely prefer to use the full features of a groupware application, rather than continuing to use the reduced model necessary for a poorer channel. It would be unnecessarily restrictive to limit the capabilities of groupware so that a user cannot take advantage of greater network resources when they are available. Thus, collaborative applications should be *adaptive*, modifying their behaviour as the quality of the communication environment changes.

This research is concerned with two different problems. First, it is intended to enable mobile collaboration, specifically, collaboration on written documents through the use of a group editor. A multi-user editor was studied in order to determine what changes would allow it to work more effectively in a poor wireless environment. These changes primarily involved the group interface, which could be restricted as necessary to allow

for better operation when the channel quality was very poor. Also studied was the possibility of adding an adaptive component, so that the interface could return to full capacity when conditions permitted. Preliminary test results and extrapolations suggest that these changes would allow large reductions in network traffic without restricting the interface unnecessarily. These changes would allow the editor to be used as effectively as possible under a variety of circumstances, with minimized delays and disruptions. Many of the general design principles developed in this test case can be applied to other types of groupware that users wish to use in a wireless environment.

The second part of the research involved an application program interface (API) developed by the Mobile Computing Project. The API supports adaptability by notifying applications (such as a group editor) of relevant changes in network conditions. The API had an initial set of network parameters that were selected for monitoring, but this set was chosen without any experiments on actual applications. Test studies of several types of applications were needed to refine the list of parameters, by revealing which parameters were irrelevant and which new ones needed to be added. This test study of a multi-user editor provided information as to which parameters were useful for mobile editing, and many of these parameters are useful for other types of groupware as well. Along with tests of other applications [15] conducted within the Mobile Project, this thesis research suggested a new set of refined parameters which can be used by many types of adaptive applications, without wasting resources by monitoring irrelevant variables.

The thesis is organized as follows. Chapter 2 outlines basic issues and concepts of groupware and wireless communications, and the effects of a wireless channel on computer-supported collaboration. The need for adaptive wireless applications, includ-

ing collaborative applications, will also be discussed here, along with related work. Chapter 3 describes, in detail, the multi-user editor application, Calliope, used as the basis for this research. It presents the architecture and features of the original editor, and the toolkit used to develop and support it. Chapter 4 discusses potential modifications that could be made to a multi-user editor to improve its performance over a wireless channel. This chapter also presents details on specific changes made to Calliope, along with a description of the API that supports its adaptability. Chapter 5 presents evaluations of the application changes, discussing both communication improvements and suggestions for changes to the API. Finally, Chapter 6 presents conclusions and suggests some areas for future work.

Chapter 2

Background

This research brings together two areas of computer science: computer-supported collaborative work and wireless communications. It is necessary to understand the basic ideas of both areas in order to understand how collaboration might work over a wireless channel. Therefore, what follows is a brief overview of the two fields, plus a specific description of the major obstacles to wireless collaboration. Following this is a description of one solution for coping with the unpredictability of a wireless channel: adaptive applications. This discussion is followed by a description of how a group editor was chosen as the particular application type. After this background material is outlined, an overview of related work is presented.

2.1 Groupware

The development of software to support collaboration has allowed workgroups separated by space and/or time to coordinate their activities toward a common goal. The types of

groupware are as varied as the tasks that users wish to perform. There are simple packages for writing, drawing, and brainstorming, as well as complex multimedia systems for videoconferencing and distance education (see [1] for a survey).

Two central areas of groupware are distributed systems and human-computer interfaces: distributed systems are used to support communication, while specialized interfaces are necessary to coordinate the actions of people who are often communicating only through their computers.

When designing an application, developers of groupware need to consider several factors about the network over which their software will run: responsiveness, robustness, and concurrency control [2]. Responsiveness requires that response and notification times be short in order to facilitate coordination. Response time is the time taken for a user's changes to appear on their own interface, while notification time is the time taken for these changes to propagate to other users' interfaces [4]. Robustness is needed in order to ensure that a distributed collaborative session has minimal disruptions from communication failures; the system must recover gracefully from such problems as temporary disconnections. Finally, the network must support concurrency control, which resolves conflicts between different users' operations on shared data. One example of conflict could occur in a multi-user editor: one person might try to correct the spelling of a word that another user is trying to erase. Such actions must be carefully monitored and resolved to ensure that both users' documents remain consistent.

Given these basic network factors, the designer must select an appropriate architecture for a particular application. There are three basic ways of organizing shared information over a network [17]: via centralized, replicated, and hybrid architectures. A

centralized system consists of one server to which all users connect; it stores all information and executes all users' commands on the shared workspace. Replicated systems are composed of several copies of an application process, usually one for each user. Processes communicate with one another to perform actions and share data, rather than going through a central server. Hybrid systems are any combination of central and replicated architectures.

There has been much debate over whether a centralized model or a replicated model is more useful for groupware [9]. The centralized model is very simple to synchronize, which facilitates concurrency control and data sharing. However, communication with a central server can be slow, as all actions have to be processed at a single site. This communications bottleneck decreases responsiveness, which degrades coordination. Furthermore, a centralized architecture may not be robust, as a loss of the server will prevent all users from participating in networked collaboration. The replicated architecture, on the other hand, has better response time because many actions (such as updating) can be done in parallel. Several copies of the groupware ensures that a single software failure will not affect all other session participants. The price of this speed and reliability is that replicated processes are difficult to synchronize, requiring complicated software to handle concurrency control and data sharing. Developers of hybrid models still must deal with these issues, but can select the most appropriate model for various groupware components.

Besides network concerns, there are also several important user interface issues to be considered when developing groupware. Because collaborators may not be able to see or hear one another, the shared computer-based workspace must provide special features to

coordinate users' actions. Many group interface features are designed to support group awareness, a representation of what other collaborators are doing that helps the session run smoothly. Group awareness features include gesturing tools (used to indicate regions of interest on the screen) and specialized representations of the shared workspace (such as a miniaturized view of a large document). The interface often includes tools for concurrency control, providing access to such methods as serialized turn-taking or exclusive locking of data to prevent conflicts.

Groupware interfaces can be very simple or very complex, depending on the application, and different packages require varying amounts of network support. A text-based brainstorming tool, for example, consumes less bandwidth than a graphical, collaborative CAD package. While full-featured interfaces are extremely useful in coordinating collaboration, they may not be properly supported in all network environments, or at all times (for example, if a network problem develops during a session).

These groupware issues demonstrate that the underlying architecture and group interface are vital components of collaborative software. However, these features make several demands upon the network over which they run. Without proper network support, a distributed collaboration session can become disjointed and frustrating for the users. Often, designers assume a certain level of functionality (for example, an ATM network for videoconferencing). When such functionality is available, then the session should run smoothly. However, it may not always be available, or may not be available to all members of a group that wishes to collaborate.

In cases where a lesser form of network support is available, a groupware application might still be usable, but perhaps not as responsive or robust. Because network resources

are often limited, if a groupware application makes too many demands on the network, there will not be sufficient resources to run a coordinated session. This suggests that groupware should be able to use network resources efficiently or sparingly when necessary, to allow for a more limited form of collaboration to occur under more restrictive circumstances.

2.2 Wireless Computing

One possible network environment in which collaborators might wish to work is the wireless environment. Mobile computers, such as laptops and personal digital assistants, can communicate over a wireless channel. This allows a person to travel while maintaining connections with other machines, or to run networked applications in locations that do not have any wired network support (e.g. outdoors). Wireless networks can also be used to connect "immobile" machines that are plugged in to a power supply but do not have access to a wired network. This situation might occur in a factory, for example, where network cable cannot be laid between two offices that are separated by extensive machinery.

Figure 2.1 shows a generic wireless architecture. The coverage area could vary widely: it could be a small room, or a large city. The area contains several antennas which represent transceivers, known as *base stations*. The base stations are (usually) connected to a wired network, and communicate with mobile computers over a wireless link. (They can also communicate with wired machines, such as workstations, connected to the wired portion of the network.) Wireless communication can be carried over various wavelengths: wide-area networks generally use radio frequencies, while indoor

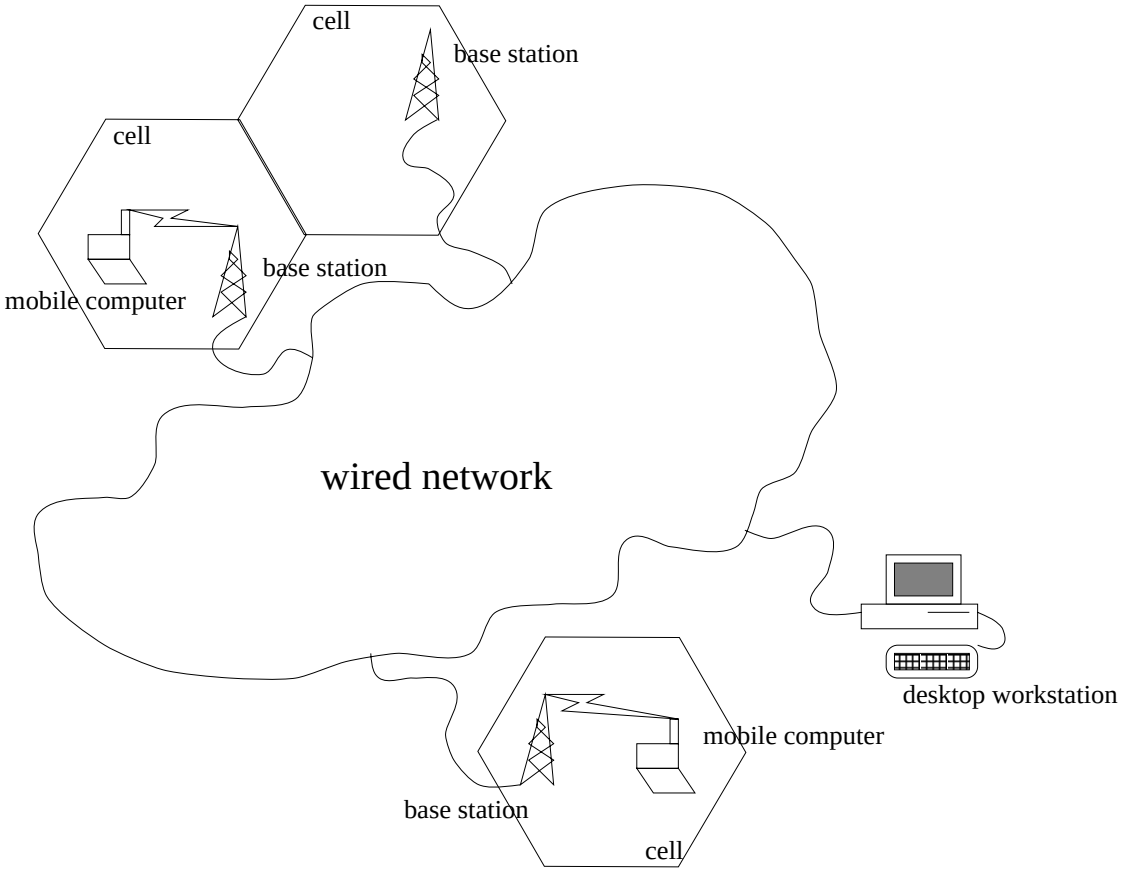

Figure 2.1: A typical wireless scenario.

LANs may use radio or infra-red frequencies.

The area over which a base station can transmit and receive signals is known as a *cell*. A mobile user may move out of the coverage area of one base station and into another cell. This situation, when a mobile unit moves to a new base station, is known as a *hand-off*. In this way, a mobile computer can maintain a connection despite a change in location.

The convenience of wireless computing comes at a price: there are a number of restrictions that limit powerful mobile computing. Mobile constraints can be broken into two classes: constraints on the portable computer, and constraints on the wireless network. Portable computers have more modest resources than stationary machines. For example, they have less disk space, a smaller screen, and a finite energy source [6].

This research deals primarily with network constraints, problems that will persist even as portable computer hardware improves. As mentioned above, a wireless network uses radio waves or infrared light to carry information. This signal can easily be interfered with: blocked by buildings, corrupted by noise, or attenuated by distance. This leads to high error rates and frequent, sudden disconnections. Disconnections may cause a networked application to behave unpredictably, possibly causing it to stop running suddenly.

Furthermore, wireless networks have considerably less bandwidth than wired networks. For example, CDPD, a wide-area cellular data network, has a nominal rate of 19.2 kbps [11]. This is substantially slower than conventional wired LANs, which can transfer data at rates between 10 and 100 Mbps.

Hand-offs can also cause communication problems. When a user changes base stations, information must be transferred to the new cell (for example, authentication information). Communication resources, such as an unused frequency, must also be obtained for the new user, which can be difficult if a cell contains many users. Thus, a hand-off can result in delays, short disconnections, and high error rates.

Another feature of mobile connectivity is that it is highly variable. Bandwidth and reliability may change dramatically: a user could leave a building with a high-speed

wireless LAN and move into an area covered by a low-speed cellular link.

Furthermore, a mobile user might decide to plug into a wired network for a short time, which might increase bandwidth by two or three orders of magnitude. It would be beneficial for an application to be able to take advantage of fast connections when they became available, while still maintaining its ability to use a low-bandwidth connection.

2.3 Wireless Collaboration

Having outlined the basic issues of groupware and wireless communications, the problems of wireless collaboration can be examined. Any application, including groupware, that runs over a wireless channel will frequently be subjected to delays and disconnections. These problems may increase or decrease, depending on the wireless channel quality, which may vary over time and the location of the user.

There are a number of problems specific to groupware that occur in a wireless environment. As described above, long response times induced by network delays will lead to a disjointed collaboration session. Session coordination can be further disrupted if participants are disconnected, particularly if the software does not alert other collaborators to the fact that one or more session members are temporarily unable to participate. Disconnections can also lead to lost work: an application may crash and destroy unsaved work, leading to lost time and increased user frustration. A peripheral consideration is that all networked applications must share a limited channel, and if a groupware application consumes a lot of resources, other software (such as email or World Wide Web browsers) may experience longer delays. This may be unacceptable for users who need data from more than one program at a time; in these cases, applications need to share

resources more equitably.

The solutions to these problems boil down to two elements: reduce delays and handle disconnections gracefully. Approaches to these solutions can be made at several places in the protocol stack. Delays could be reduced through improvements at the physical layer: for example, data transfer rates would increase with improved wireless transmission hardware. Another reduction in delays could occur at the transport layer: for example, changes made to TCP to make it more efficient over a wireless channel. This research looks at the problem from the application layer: what changes can be made to a program to allow it to make the most efficient use possible of a variable and often poor wireless channel?

From this perspective, a number of basic application design principles can be suggested. Because the application cannot change such factors as bandwidth and disconnection frequency, it must cope with these problems as best it can. Thus, the only way to reduce delays at this level is to reduce network traffic. Whenever there is more data to be sent than the channel can handle, there will be delays. Of course, the application must generate some traffic, or else it is of little use. To make the most effective use of the channel, it is best to send the most useful data, and cut down on redundant, irrelevant, or unimportant data. This ensures that the most important data gets through with minimal delay.

Unfortunately, disconnections cannot be reduced by changes in application behaviour. Instead, the application should handle disconnections as gracefully as possible. How this principle is applied depends very much on the type of program in question. For groupware, it is a matter of ensuring that work is not lost, and that the session moves as

smoothly as possible despite the periodic disconnection of collaborators.

Of these two solutions, the reduction of delays is far more complicated. If an application is to reduce its generation of network traffic, then it usually needs to perform fewer tasks. This leads to an application with fewer functions, but one whose remaining functions are performed efficiently. This leads to the question, "How can an application strike a balance between function and delay?"

As described above, a wireless channel is often of poor quality, leading to lengthy delays for networked applications. Under such conditions, an application may not be able to use all of its features, and would have to be run in a "restricted" mode in order to function with minimal delay. However, the features to be disabled might well be valuable ones; although the designer considered them to be of lower priority than those retained in the set of core functions, they nevertheless perform some task that a user may desire.

Thus, the restricted mode is only useful when network delays would make the application useless or overly frustrating to use. This is often the case in wireless networks, but this condition can change. Because of the variability of the channel, it cannot be assumed that the restricted mode is always appropriate: at times, it could be unnecessarily limited.

In order to make best use of the channel, then, an application should be able to adapt to changes in channel quality. This requires that it have a source of information about the network, which need not be gathered by the application itself. Also required is a set of network variables and values that the program can use to determine when to switch modes of operation, and the appropriate code to provide the different levels of feature support. For example, a mail application might only import the headers of mail messages (if the messages were stored on a remote machine) if the amount of remaining bandwidth

became very low. This would allow the user to select the most important messages for downloading, which would minimize use of the limited channel. When bandwidth once again rose to sufficient levels, the entire contents of the mail "inbox" could be downloaded at once.

The network monitoring system used in this research is described in detail in Chapter 4, along with a description of how adaptability was added to one particular groupware application.

2.4 Selecting an Application

As this research is not concerned with recreating any of the basic components of groupware (such as concurrency control or a group interface), it was decided that it would be best to modify an existing application. This would ensure that the software had been created by those with knowledge of groupware issues, and would cut down on the amount of coding needed to get an adaptive application written.

Because groupware is a growing and multi-faceted area, there were a number of different types of collaborative applications that could have been chosen for a test study. However, some types of applications could not be used due to their network demands. Any application dependent on real-time multimedia (such as videoconferencing software) makes heavy demands upon network resources, and research into supporting multimedia over wireless is an area beyond the scope of this thesis. The application source code had to be available, because of modifications needed to make it adaptive. Because the results from this test study are to be generalized to groupware beyond this one application, the software chosen should be a common type of groupware, rather than a niche

application.

These factors lead to the selection of multi-user editors as the best type of groupware to study. There have been a number of editors developed, which provided several sources of existing code and documentation. Group editors do not require multimedia support, and they have many of the features and requirements of other types of groupware. Because group editors can be used with varying numbers of users and different amounts of text, they provide a wide range of test cases for use in generalizing results.

2.5 Related Work

There has been a great deal of research conducted in the area of collaborative writing and editing. This thesis is not concerned with the specific issues of group editors so much as how they may be used over wireless networks. Thus, this research does not further such areas as group interfaces and menu design. However, in researching alternatives to current group editor designs and implementations, I required an overview of several types of systems, in order to understand basic issues. The work presented in [4] and the system descriptions in [1] supplied enough information to extract the most important issues surrounding collaborative editing, such as concurrency control and shared interface features.

Some recent work in mobile computing has examined the necessity for adaptive applications. Carnegie Mellon University's Odyssey project aims to provide Unix with extensions to support adaptive mobile computing [22]. The Odyssey project is designed to allow applications to negotiate for resources, such as bandwidth or cache space. If an application requires more resources than are available, then it scales back its features

to make best use of available resources. It can also register a window of interest with Odyssey, so that the application can be informed if the level of a resource improves or degrades beyond these limits. The application can then renegotiate to obtain additional or fewer resources. This approach is similar to our Mobile Project's API, although our project is not concerned with the fair distribution of resources among users or processes. The Odyssey project does not give any guidelines for application adaptation, so this work on adaptive groupware would be of use to those people who wish to use Odyssey but are unsure how to scale back a group editor when they are granted few resources.

An investigation into a specific adaptive application, a web browser, was conducted by Terri Watson at the University of Washington [26]. Watson's W* web browser is designed to adapt to variations in network connectivity, and provides the user with information on the interface about potentially slow hyperlinks. This project demonstrates some aspect of network monitoring, although the adaptive nature of the application is restricted to supporting disconnected operation and data migration. There are no changes to the interface to make better use of limited resources: the browser attempts to operate in "low-resource" mode all the time. The idea of informing the user about latency (and hence, potentially frustrating slow operations) is a good one, and should be made available in mobile groupware interfaces. However, W* might benefit from a more adaptive interface, and some of the changes proposed in this thesis might be applied or modified for use on a browser.

Further research into a specific adaptive application is being conducted at The University of Tromsø, where investigators are studying a diary application [5]. This application shows daily appointments, some of which may have attachments (e.g. notes for a meet-

ing). The application user applies for a level of quality of service, and if this service is granted, the user will be informed should the level of service no longer be available. The user, or user application, can then take steps to work within the resource constraints. For example, in the case of the diary application, the user is informed that any large attachments stored remotely cannot be loaded onto the mobile within a reasonable time. The user is also informed of disconnections, so that they are aware that they cannot access or update any diary information stored remotely. This project is very similar to the Mobile Project's work, although Tromsø have included resource management and distribution functions within their quality of service framework. However, as with Watson's browser, the changes to the interface are minimal. Thus, there are many issues discussed in this thesis that could be used to make the diary application interface less resource-intensive, which might allow it to work better within quality of service limits.

A small amount of work has been done in the area of collaborative software in a mobile environment. Koch's IRIS group editor [13, 14] model proposes an interface that considers some of the needs of mobile users. Users have the ability to request or send updates on demand, which gives them the ability to wait until connection quality improves before placing update-traffic demands on the channel. However, there is no integration of this editor with a network monitoring system, which means that the user does not know what the connection quality is at a given time. This is because Koch's research focuses mainly on an adaptive interface for different user requirements, rather than network restrictions. Furthermore, the issues surrounding the group interface, such as concurrency control, are not dealt with specifically. Thus, there are few recommendations for ways of restricting the interface to make better use of a limited channel. However, the idea of

controlling updates is a good one, provided it does not make the group editing session incoherent. If Koch's model were integrated with my suggestions for specific interface changes, then a very adaptive, robust mobile editor model would result.

The most comprehensive research on adaptive, mobile groupware has been conducted at Lancaster University [3]. Lancaster have developed a prototype to allow field engineers to use a geographical information system to view such material as network diagrams. There is support for shared views of data, diagram annotation, and for audio communication. The application sits above a modified ANSAware software suite which has been extended to provide a requested level of quality of service for such variables as jitter, throughput, and latency. If the quality of service degrades, then clients are informed of this change through a callback.

This research is similar to that of the Mobile Project, and Lancaster's discussions support our view that groupware must be adaptive in order to work effectively over wireless channels. The architecture developed at Lancaster could be used with several types of adaptive applications. Their research, however, deals with an application that is quite different from a group editor, which means that they do not make any suggestions for modifications that would enable an editor to run better in a restricted environment. Thus, my research could be used to extend their work by providing a set of recommendations for another type of groupware.

Overall, the results presented in this research could be used in two different areas. First, the thesis suggests changes that could be made to other group editors (and, to some extent, other groupware applications) to improve their behaviour over wireless networks. Second, the application description could be used to create more software for use with

architectures that support adaptive applications, thus extending their usability.

Once the background research into design issues had been completed, an application had to be selected for use in a case study. The details of the selected application, Calliope, are presented in the next chapter.

Chapter 3

Description of Non-Adaptive Group Editor

Once the basic design issues had been analyzed, it was necessary to study them on a real multi-user editor. After a number of group editors were tested, the best candidate found was Calliope, a prototype synchronous collaborative writing tool developed by Alex Mitchell at the University of Toronto [18]. After conducting a field test of a group editor used by grade six students, Mitchell learned what features were needed to create a writing tool that supports collaboration. The results of this study are realized in Calliope.

There were a number of reasons to select Calliope as the editor for this research. First, the source code was available, and Mitchell graciously granted permission to modify and expand the code. Second, the application had to be robust in order to run in our heterogeneous environment, particularly given that it was going to be modified. Calliope was developed after a field test, which suggested that it was stable enough to be used consistently. Third, the application had to be fairly representative of other editors, to

allow the test results to be generalizable. The field study demonstrated that Calliope had enough features to be used by people as a "real-life" editor. Fourth, the editor had to be easy to modify. Calliope was built using Groupkit (see below), a toolkit which facilitates groupware development and makes it easier to maintain and modify groupware applications. Calliope and Groupkit are both written in Tcl, a simple scripting language that is often used for rapid development [20]. These basic features and implementation details made Calliope a good choice for experimentation. Details about both Calliope and Groupkit are provided below.

3.1 Calliope

Calliope shares many features with other group editors. Users may be geographically separated while writing simultaneously, and they will have a shared view of the text. Beyond these basic attributes, Calliope has six major features that facilitate collaboration: access control, public annotations, private text, external annotations, unstructured text space, and awareness tools.

3.1.1 Access Control

When writers are working on a shared document, they will at times run into conflict over who is able to modify a given region of text [4]. To resolve conflicts, some means of access control is needed. In Calliope, access control is maintained using multi-level locking. A user can request a lock on a linear, contiguous area of text. If no other person has locked this area, a lock is granted to the user. When a region is locked, no other user

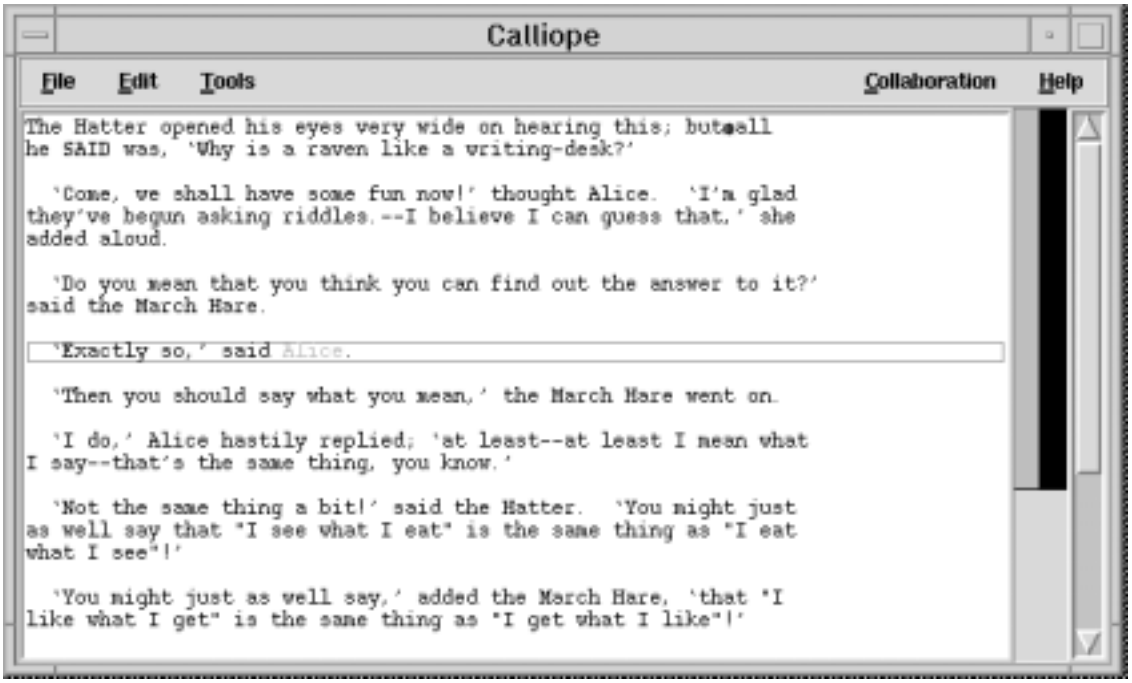

Figure 3.1: An editing session using Calliope. A lock is being used on line 11, and this same line displays the colored text feature. The telepointer can be seen on line one, and the group scrollbar is visible on the far right.

may modify the text within it: it becomes “read-only” to all others. The locked region is displayed with an outline around it, and although others may not change the text, changes are presented to all other participants as they occur.

Locks can be requested for a word, line, paragraph, or text selection, which allows more than one user to lock part of the text (although each user is limited to one lock at a time). Figure 3.1 shows an editing session where one user has locked a line in order to have exclusive access.

Once the lock is released, the text can once again be edited by other group members.

In this way, Calliope ensures that users can control a region of text for long enough to make changes without being interrupted by a collaborator.

3.1.2 Public Annotations

When writers are creating and editing, they require the ability to add comments about the document being worked on. If the writers were in the same room, such comments could be spoken. However, with geographically separated users, this mode of instant feedback cannot be done vocally: instead, Calliope provides a public annotation scheme. Using public annotations, a user can leave a message for other writers or suggest changes or additions. Figure 3.2 shows the annotation mechanism: a small window is presented to the user for entering their text.

Once a user has begun to enter the message, other users see an icon on their screens. If this icon is clicked on, the annotation window opens, and others can follow the annotation-writing as it progresses, a keystroke at a time. When the message is complete, the writer of the annotation can close the window, which leaves the annotation icon on every users' screen, which they can open and read at any time during the session.

3.1.3 Private Text

Calliope users are able to leave public comments, but they also require private spaces for creating and modifying their writing before their work is presented publicly to the group. To support this requirement, Calliope provides private text windows; see Figure 3.3 for a picture of this feature. A user selects an insertion point or section of text; if text is selected, it is copied to the private window. When the user has finished their private

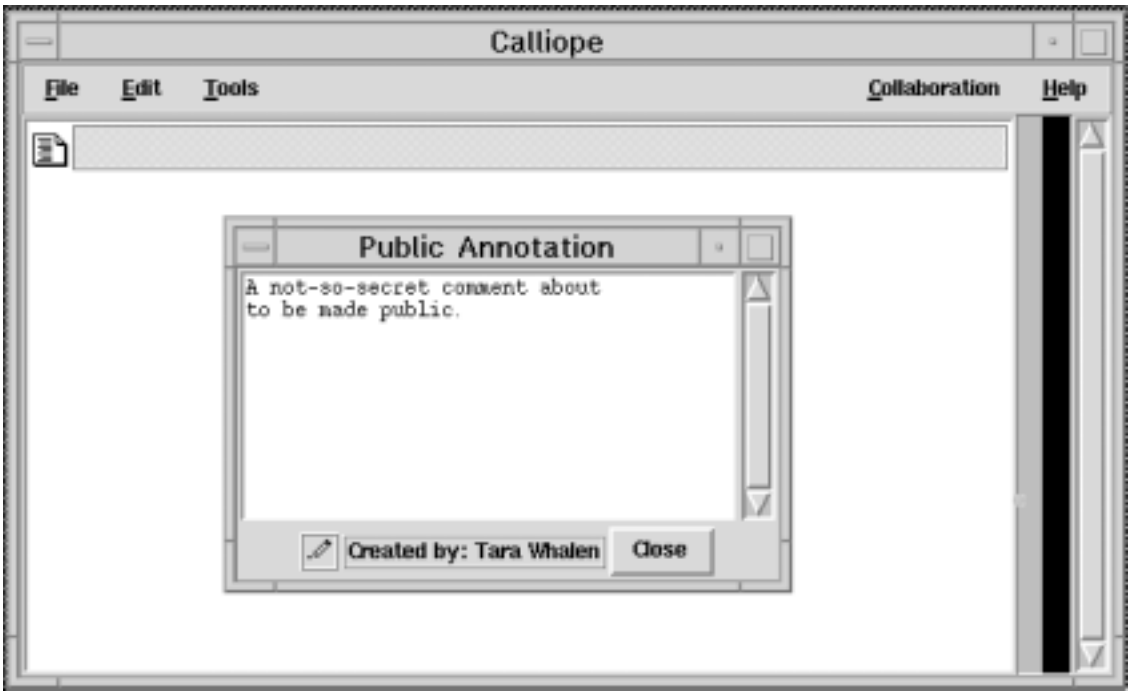

Figure 3.2: The public annotation feature.

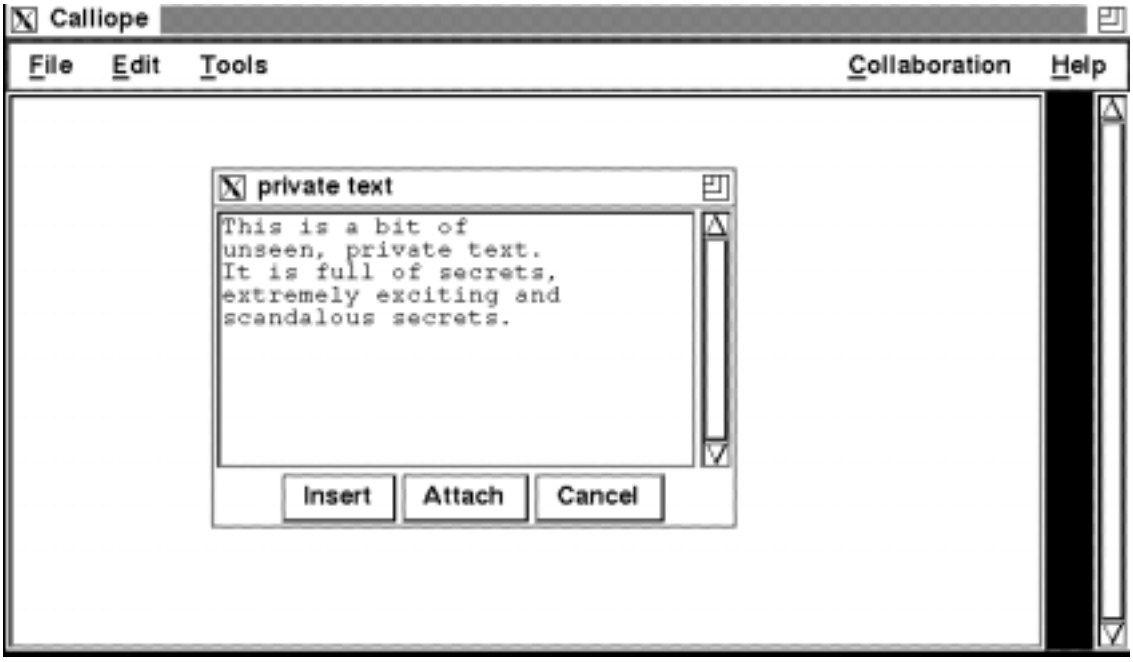

Figure 3.3: The private text feature.

work and wishes to add it to the shared space, they can insert their text straight into the document, or they can choose to add their text as a public annotation.

3.1.4 External Annotations

External annotations are related to the idea of hypertext: information related to and connected with the basic text. This naturally leads to the extension of linking a group editor to the World Wide Web. Calliope allows users to embed a Universal Resource Locator in the text; clicking on the external annotation label will spawn Netscape and bring the user to that URL.

3.1.5 Awareness Tools

While annotations allow users to make references outside the main text, the shared text space has to have some sense of cohesion. There may be several users working on different parts of the document, which can make collaboration more difficult. To support awareness of other users' actions, Calliope provides several mechanisms. Colour-coded text reveals which authors wrote which sections of a shared document, and information about when the text was modified is available to all writers. These tools give joint authors some notion of ownership over their text, and allow a small amount of document history to be maintained.

Two of Calliope's tools are provided by Groupkit (see below): the telepointer and the shared scrollbar. The telepointer is a small dot which shows the location of a user's mouse; this can be used to gesture or merely to reflect which section of text a user is currently focussed on. The shared scrollbar allows a person to "follow" another person's view: if a user selects this option, their window scrolls to match all scrolling movements made by the user whom they have chosen to follow, allowing both participants to have the same view of the workspace. The user can unlink these scrollbars at any time to regain control of their own separate view. Both of these features can be seen above in Figure 3.1.

Another awareness mechanism is a document overview. Like the telepointer, it reveals to the group what the full structure of the document is, and where people are in relation to one another. This overview is a "gestalt view," a miniature view of the whole document in one window with marks to show each user's selection point. The overview also uses a "fisheye lens" that magnifies one's own text and, in a slightly smaller font,

presents the text around other users' selection points. The rest of the text (the material that is not being worked on) is presented in an extremely small font in order to present the full view and highlight the sections being modified. All these features help a distributed writing session by giving a sense of the activities and focus of group members, a task made difficult when writers are not in the same room.

3.1.6 Unstructured Text Space

Many of the tools Calliope supports are designed for use on a central, structured text document, which is to be expected in an editor, but this structure limits the interaction between collaborators. Mitchell decided to implement an unstructured text space away from the main document, to encourage users to experiment while writing. Calliope provides a shared scratchpad for those users who require an unstructured text workspace. Users can paste text into the scratchpad, write comments, and draw on it like a whiteboard. The telepointer can also be used in this space, allowing users to gesture to text fragments. Entered text can be moved, deleted, marked up and edited, allowing more experimentation and free expression than can be supported in the main text window.

3.2 Calliope Architecture

To create Calliope, Mitchell used the Groupkit toolkit, a groupware development system that facilitates rapid prototyping of collaborative software. In order to describe the architecture of Calliope, we must describe the features of the Groupkit system that it is based on.

3.2.1 Groupkit Overview

Groupkit is a groupware toolkit developed at the University of Calgary [21]. It is written in Tcl/Tk, and is designed to run in a Unix environment networked using TCP/IP and running X Windows. Groupkit was developed in order to give developers some basic tools to facilitate the creation of groupware. The Groupkit developers state that "a developer using a well-designed toolkit should find it only slightly harder to program usable groupware systems when compared to the effort required to program an equivalent single-user system."

To support this goal, Groupkit includes a number of features that facilitate common requirements of groupware programming. These include process and communication management, two low-level tasks that can make programs unnecessarily complicated. When conferences are replicated rather than centralized (i.e. each user has a corresponding copy of the conference application), data must be shared and events from one user's process should be able to cause actions in other processes. Finally, user interfaces for groups have a different look from single-user interfaces, and the development of widgets for a group interface can be time-consuming. Groupkit fulfills these requirements by supplying four types of tools: a runtime infrastructure, groupware programming abstractions, groupware widgets, and session management.

3.2.2 Runtime Infrastructure

A central requirement of groupware is the use of multiple distributed processes. Groupkit provides a runtime infrastructure to manage these processes. This system handles such tasks as process creation and destruction, interconnection, socket handling, and multi-

casting. Furthermore, it provides the infrastructure for session management (see below). The infrastructure consists of a number of distributed processes which fall into one of three categories: conference application, session manager, or registrar. This infrastructure is presented in Figure 3.4 and its three components are described in detail below.

Conference Applications

Groupkit "conference applications" are simply groupware programs created by an application developer and invoked by the session manager. Examples of applications include group editors, shared whiteboards, and brainstorming tools. Under the Groupkit architecture, conference applications consist of a set of replicated processes (one copy of the program on each user's machine). The conference processes work together in what is termed a conference session: all the processes communicate with each other rather than going through a central server. This communication is supported by the use of the programming abstractions described below.

A user may be running several different conference sessions at once, all of which are managed by their session manager. The session manager can also provide applications with information on changes in conference state. A brainstorming application, for example, might wish to display the list of participants. The application requests this information from the session manager, which obtains it from the registrar and returns the desired list.

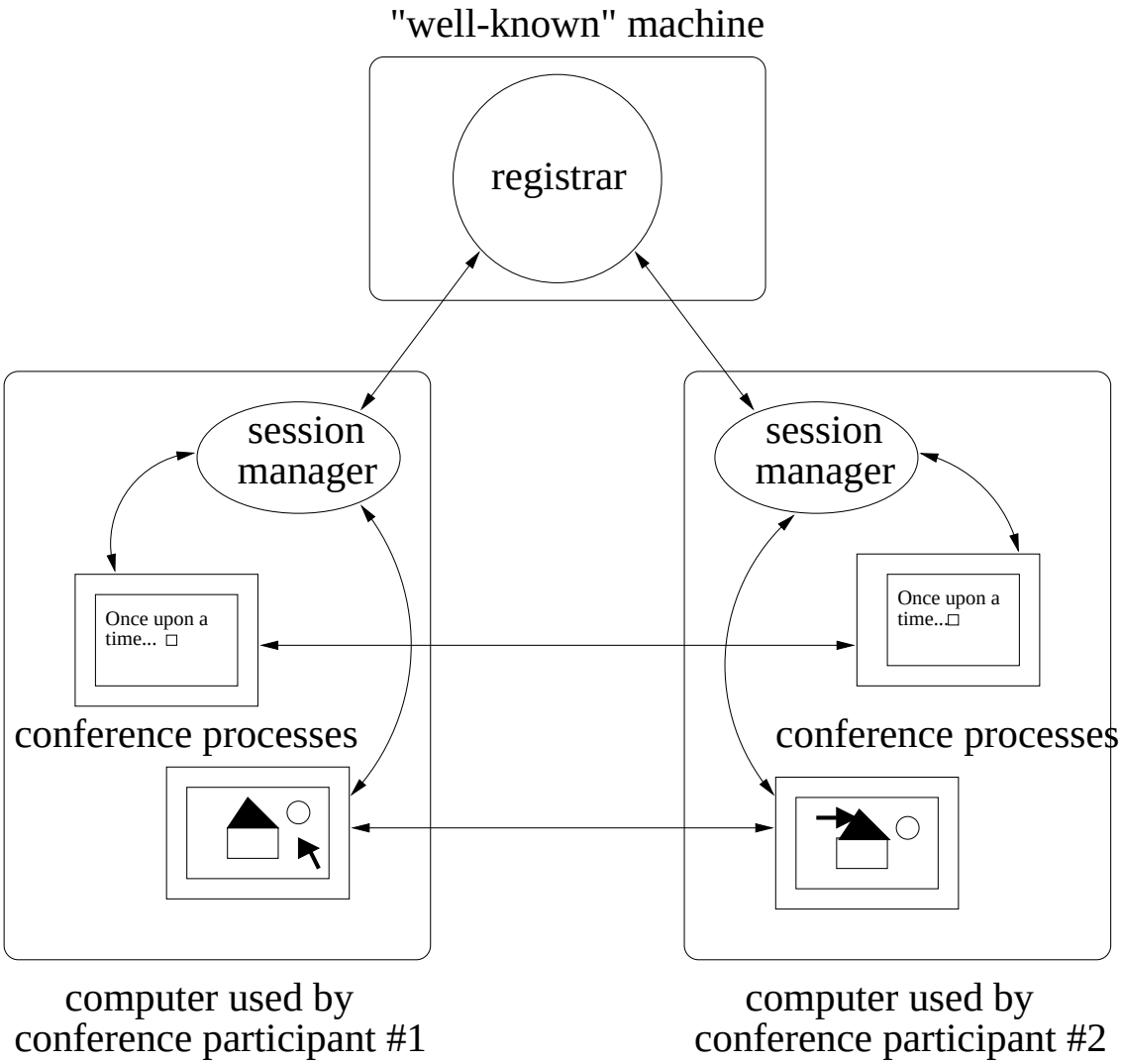

Figure 3.4: The Groupkit runtime infrastructure.

Session Manager

Every Groupkit user has a single session manager process that handles conferences. The session manager lets people create, monitor, join, leave or delete conferences. The interface displays a list of conferences and participants, which gives the user the information and means to join or initiate a conference session. Furthermore, the session manager has a policy that dictates how users can enter and leave conferences; this policy is reflected in the user interface. (For example, a conference that is open only to a small group need not appear in every user's list of current conferences.) Session managers do not communicate with one another; instead, they send updates to the central registrar, and check with the registrar for information on conference state.

Registrar

The registrar must begin running before a conference session can begin. It runs as a server process, and is the only centralized process required in the Groupkit infrastructure. The port number and machine name of the registrar are included in each user's configuration file, making it a "well-known" address for conference participants.

When a session manager process is spawned, it connects to the registrar. The registrar maintains a list of all the current conferences and the users participating in each conference. The registrar provides the session manager with the information necessary to locate (and connect to) a conference. Any significant changes to a conference process (conference destruction, change in participants, etc.) are relayed by the session managers to the registrar, who updates the conference information listing. If the registrar fails, conference participants can continue to collaborate. However, attempts to add or

delete conferences will fail, and when users leave, their name will not be removed from the participant lists maintained by the remaining conference participants.

3.2.3 Groupware Programming Abstractions

The management of distributed processes is a common need for all groupware, but there are a number of application-specific tasks that programmers should be able to handle effectively. Thus, Groupkit supports a number of programming abstractions that allow programmers to handle such tasks as interprocess communication, sharing of relevant data, and taking action on conference state changes. These tasks are facilitated by three abstractions: multicast remote procedure calls, events, and environments.

Multicast Remote Procedure Calls

As described above, Groupkit conference applications consist of replicated copies of groupware processes. These processes will need to share information and communicate with one another in order to maintain a session. These tasks are performed using a specialized form of remote procedure call (RPC). The Groupkit RPC mechanism uses Tcl sockets, a feature that was added to Tcl in version 7.5. Using Tcl sockets, a command can be sent from a local process to a remote one. This Tcl command is then executed on the remote machine as if it were called locally. Groupkit has abstracted RPCs to provide two major benefits: a programmer need not know the addresses of all conference processes, and the multicasting of commands to some or all conference processes is automated.

There are a number of RPC calls available to the programmer. Gk_toAll sends a command to all copies of the conference program, including the local copy. Similarly,

gk_toOthers sends a command to all processes except the local copy. A third useful call is gk_toUserNum, which sends a command to a particular user's process. This process is not identified using an address; instead the programmer retrieves a unique user number that is maintained by Groupkit.

Messages may arrive at different times at different conference sites, which can lead to problems for conference applications that need consistent ordering of messages. To solve this problem, Groupkit includes another type of RPC, gk_serialize, which serializes messages through one of the conference processes to ensure in-order delivery.

One final comment about Groupkit RPCs is that most are non-blocking. After an RPC call is made, program execution continues without waiting for a reply from the remote process. This decision was motivated by the fact that Groupkit programmers rarely require a return value from an RPC call. Because blocking RPCs can lead to delays (due to such factors as network latency, disconnection, etc.), this approach leads to faster response times. (Groupkit has, however, included a blocking variant of gk_toUserNum in its RPC list.)

Events

Groupkit provides another abstraction, the *event*, which allows conference applications to be notified when certain things occur. An event comprises an event type and a set of {*attribute, value*} pairs which describe the event (the particular attributes depend on the event type). Some events are generated automatically by the Groupkit infrastructure, such as a user joining or leaving a session. However, Groupkit can also handle custom events created by a programmer for application-specific needs. In order to trap an event,

programmers use an event handler, or *binding*. Bindings are specified using the command gk_bind [event-type][action]. The action consists of callback code which is to be executed when the event happens (e.g. "notify all participants that User X has left"). This mechanism makes it much easier to propagate information about conference state from one process to another.

Environments

Another mechanism for co-ordinating distributed conferences is Groupkit's *environments*. An environment is a hierarchically structured dictionary containing *keys* and associated *values*. Groupkit allows distributed environment copies to communicate, thus forming a data structure that is shared across machines. This ensures that changes can be propagated from the environment of one conference process to the others. Currently, the default concurrency control mechanism for environment updates is "no concurrency," but programmers have the option of serializing changes if desired.

To access and update the information in an environment, a programmer uses a hierarchical key. For example, one of the basic Groupkit environments is the users environment, which maintains several facts about the current participants. In order to retrieve information about the user's full name, the command "users get local.usernum" would return the unique number assigned as an identifier for the local user.

The environment has two features that make it valuable for groupware. First, as mentioned above, it is a shared data structure that transmits changes to all copies of a replicated conference. Second, a programmer can bind callbacks (similar to events) to environment changes. This allows for other actions besides updates to occur when a

change is made (e.g. saving a copy of the new data).

3.2.4 Groupware Widgets

Developers creating a user interface for collaborative software need to consider the special needs of a group. To make the developers' job easier, Groupkit includes a number of multi-user groupware widgets that can be added to an application. These widgets come in three classes of functionality: participant status, telepointers and location awareness.

Participant Status

Conference users often need to have an idea of the composition of their collaborative group. To fulfill this need, Groupkit has provided a "participants widget" that lists all participants in a session and is updated as the group members enter or leave. Participants may have provided information about themselves (address, picture, etc.) which can be retrieved by double-clicking on the button with their name on it.

Telepointers

Groupkit's designers wished to include *gesturing* tools into their set of groupware widgets. These tools allow people to gesture through the computer, for such tasks as drawing attention to items on the screen. The particular gesturing tool available in Groupkit is the *telepointer*, a small circle that follows the movement of a remote user's mouse pointer. Thus, a person can indicate a word by running their pointer under it, which will draw other participants' attention to that word.

Location Awareness

While collaborators can use gesturing tools and participant lists to help maintain a distributed session, they also need to know what other participants are doing in the workspace. Without *location awareness*, people have difficulty coordinating synchronous group work. Groupkit has provided two tools to solve this problem: multi-user scrollbars and a gestalt viewer. The multi-user scrollbar allows a person to "follow" the scrolling of another user so that two users can share the same view. The gestalt viewer provides a display of a document in miniature, with boxes to highlight the sections of text that users are currently viewing.

3.3 Calliope Architecture – Details

With the knowledge of how Groupkit is constructed, we can now move on to the details of Calliope. Calliope is written in Tcl/Tk, and its main text window is a Tk text widget. A text widget has a number of built-in facilities for handling many common editing tasks (text entry, deletion, insertion, etc), features on which Calliope relies heavily. Calliope's special features (annotation windows, the scratchpad, etc.) are implemented using Tk canvas widgets and small text widgets.

Calliope also uses many of Groupkit's features to handle process management, communication and the group interface. Because Calliope is set up as a Groupkit conference application, it can use the registrar and session manager to handle its replicated processes.

Every user has their own copy of Calliope, and consistent copies are maintained by sending updates to every users' copy when any change is made. This feature relies on

Groupkit's multicasting capabilities, as well as events and shared environments. Finally, the Groupkit groupware widgets are all included in Calliope. The participant status list, telepointers, multi-user scrollbar, and gestalt viewer have all been added to Calliope to give it a rich group interface.

The basic components of Calliope needed to be studied in order to construct an adaptive model that included a restricted interface for extremely poor network conditions. The basic changes that could be made to restrict a group editor interface, and how they were applied to Calliope, are presented in the next chapter.

Chapter 4

Designing an Adaptive Group Editor

The first step in making an adaptive editor is to equip it with a “low-resource” mode. (The “high-resource” mode is what most editors currently implement, so there is little need for change there.) Many changes can be made to ensure that an editor makes efficient use of scarce resources, with less bandwidth consumed and better operation under disconnection. A study of various editors, such as those outlined in [17], provided information on common requirements and features that are found in most group editors. This information suggests a number of changes that could be made to reduce delays and handle disconnections. This is by no means an exhaustive list, and some editors may have special features or architectures that demand additional or different changes. This is, however, a good initial list to consult when designing an editor for use over a wireless link.

4.1 Document Overview

Because a wireless channel has little available bandwidth, large data transfers take a long time to complete, which can increase user frustration and generate high costs. Furthermore, if one application consumes most of the channel with a large transfer, this may preclude other mobile applications from getting sufficient bandwidth. Thus, we wish to minimize the amount of data sent over a wireless link whenever possible.

One situation that generates a large traffic volume is a user downloading a large file for editing. Often, the user does not intend to work on the whole text: she may want to update Chapter Three only, and thus a lot of unnecessary data is being shipped over a limited channel.

What would be useful in this case is an overview of the document that allows the user to select only the relevant pieces of text for the editing session. In this way, the document is filtered to allow for more efficient use of the wireless channel.

The difficulty with this approach is that it is hard to find a way to structure text for an overview unless it is marked in some way. Markup languages such as SGML, HTML and LaTeX provide document structure within the text; such marks show how a document is partitioned, and thus are well suited to text filtering schemes. However, we cannot assume that all documents being edited are going to be written in a markup language. Thus, we need to find a way to deal with plain text so that a sensible overview can be created to filter irrelevant text.

A further extension to current markup systems would be to allow the user to embed tags in the document for use in future sessions. The filtering mechanism could search for these tags, and fetch only the text between these user-set limits.

This text-filtering approach may pose some difficulties for consistency if different users are allowed to download and edit separate portions of the same document during a shared session. For example, two users might be working on the same document, but have downloaded pages 9–12 and 11–15, respectively. Changes on pages 11 and 12 must be reflected on both users' copies. Thus, an editor that supports this sort of interaction must ensure that these portions are updated and merged properly.

4.2 Updates

Another source of heavy network traffic is frequent updating. Many editors support a "What you see is what I see" (WYSIWIS) interface, which requires continual updates in order to ensure a consistent shared view [4]. Thus, every time a person changes the text, this change is propagated, on a keystroke-by-keystroke basis. As each keystroke is encoded as a small amount of information, much of the traffic generated is packet overhead (several mostly-empty packets). This sort of overhead may be far too expensive in a wireless environment.

What is necessary for a wireless channel is to have more efficient updates: less traffic, and more control over timing. Updates can be sent more efficiently in blocks of characters, so it might be better to send periodic updates of a block of text.

It would also be useful for updates to be sent over the wireless channel when conditions are favorable (e.g., when the transfer will not interfere with a higher-priority networked application). Ideally, updating should work both ways: a user should be able to control the flow of information both to and from his mobile unit. However, if users are waiting on updates, or not sending updated information immediately, then a consistent

view cannot be maintained at all times. An absence of a consistent view will obviously degrade a WYSIWIS interface, which depends on shared views of data. This could interfere considerably with a shared editing session, and so it should be invoked only when absolutely necessary.

4.3 Concurrency Control

Limited bandwidth and sudden disconnections must be considered when designing methods of concurrency control in a mobile group editor. One very common method for concurrency control is the use of locks. A user can request one lock at a time for a given linear region of text, and if no other user has locked this region, then the lock will be granted. No other user can modify this text region until the lock is released.

If lock negotiation is to be handled in a mobile environment, then a centralized lock manager should not be used, as it is not fault-tolerant. Instead, a replicated model could be used. Each user maintains a list of the currently-held locks. Whenever a user requires a lock, he can check his own list to see if the region is not currently being held by any other user. If the region can be locked by this user, then he can send a serialized update to all other participants, informing them that this region is now under his control. Each user will then have a current list of all locks being held, a list which is updated each time a lock is released or granted. The serialization step ensures that when a region is unlocked, this update will be propagated before any new locks are granted. Furthermore, it ensures that all users have a consistent view of the current lock state, so no lock can be granted simultaneously to two different users.

This model may send little traffic (depending upon the specific implementation), as

it only needs to send the single new lock update to other users. However, there are ways that it can send less traffic. One simple way to cut down on traffic is to reduce the number of lock updates sent. This could possibly be done if the user was "greedy" in requesting locks; they could, for example, request a whole line if they thought they would need to change two words, which allows one lock request instead of two. An additional change that could reduce traffic is the creation of "modes of operation" that would restrict the number of users involved in lock negotiation. If a user does not want to amend the document, then she could be in "read-only" mode, which requires no locks. Thus, the lock request and subsequent release would not be sent to her machine.

Another issue that arises when studying concurrency control is disconnection. A mobile editor must ensure that a user cannot hold on to a lock after they are disconnected, as it is difficult to say when that user might be able to rejoin the session (and release the lock). Locks must be released smoothly upon disconnection to allow other users reasonable access to the text. It might be useful to include a timeout mechanism that releases the lock shortly after disconnection. This mechanism would be invoked in connected users' machines after a specified interval. The remaining session members would update their lock lists so that the lock held by the disconnected user was released, using a similar serialization system as described above. The difficult part of this implementation is to determine which process will initiate and serialize the lock updates. It may be necessary to use a centralized process (like Groupkit's registrar) to perform the necessary updating tasks. Once the disconnected party was able to rejoin the editing session, their list of locks would have to be updated to reflect the new list of locks created during their period of disconnection. Thus, if lock timeout were implemented, a person experiencing a short

disconnection when crossing a cell boundary would be able to maintain control over their region of locked text, but a longer disconnection would trigger a lock release to free up the region.

4.4 Document Backup

As discussed above, disconnections can be frequent and sudden in a wireless system. Such an unreliable network environment necessitates a document backup system for any mobile editor. Mobile users can be expected to disconnect, and their work could be lost if there is no stable storage with a recent copy of the document. Saving on demand is not sufficient in such an unstable system.

Because people will want to be able to work even when disconnected from the session, keeping local backups is a good idea. A default "time between backups" could be set, with this time changing as network conditions change. For example, a high error rate suggests that a disconnection may happen soon, so a backup should be made to avoid losing data. There might have to be support for later merging of different versions of a document, but this is a small price to pay for stable, accessible copies.

4.5 Restricting Features

One final issue that must be considered is wireless support for group interface features. Several editors provide group awareness tools to give individual users a better sense of the working group. While these tools are valuable, they may prove too expensive under poor network conditions. For example, a feature may generate frequent updates, such

as an overview that displays each user's position in a document. This feature could be turned off temporarily if there was insufficient bandwidth to support it.

Any editor that has a bandwidth-intensive feature might want to provide a less resource-consumptive version for use on mobile hosts. Possibly this could involve fewer updates or a less complete interface, depending on the feature. In either case, it would likely be better to allow limited use of a feature rather than having to disable it completely to allow the editor to work.

4.6 Applying Design Issues to Calliope

Using the list of suggested changes, many modifications can be made to Calliope to enhance its performance over wireless networks. The changes recommended below have not been implemented, except for the two new features (document overview and backup). Instead, they are presented as alternatives to the current implementation. These suggestions are evaluated in the next chapter, where they are compared with the existing features to show estimates of savings in bandwidth consumption.

No changes needed to be made to Groupkit to support adaptability in Calliope, except for those widgets which Calliope took directly from Groupkit (telepointer and group scrollbar).

4.6.1 Document Overview

Calliope did not have a mechanism that allowed a user to select a section of a larger document, so one had to be created. This was a relatively straightforward task, except

Here we have an ordinary text document. It is being used for purposes of illustration.
Note that there are a few paragraphs here, so we expect the first lines of these paragraphs to appear in the document overview.
To have a particular paragraph imported, select its first line, and mark this as the start. The line that is selected as the end will be the last paragraph imported.

Figure 4.1: The full text of the document.

for the problem of finding structure in a plain text document. That is, the user needs to be presented with such identifying markers as major headings and the opening lines of paragraphs, which will identify the section they will need to download.

To address this problem, I used txt2html, a text-to-HTML converter developed by Seth Golub [7]. Txt2html is a Perl script that scans a plain-text file and adds HTML tags based on spacing, capitalization, and ASCII marks (e.g. dashes for underlining). These tags add formatting information for such items as paragraph breaks, pre-formatted text, bulleted and enumerated lists, and section headers.

When a user opens a file, the file is first run through txt2html to create a marked-up document. This HTML document is then scanned for paragraph breaks; the first line of every paragraph is exported to a document overview widget (see Figures 4.1 and 4.2).

The user selects the first and last paragraphs that they wish to work on, and imports only those selections into Calliope.

4.6.2 Backup System

A simple backup system has been added to Calliope to ensure that disconnections do not lead to a loss of work. The interval between backups is set to a default of every

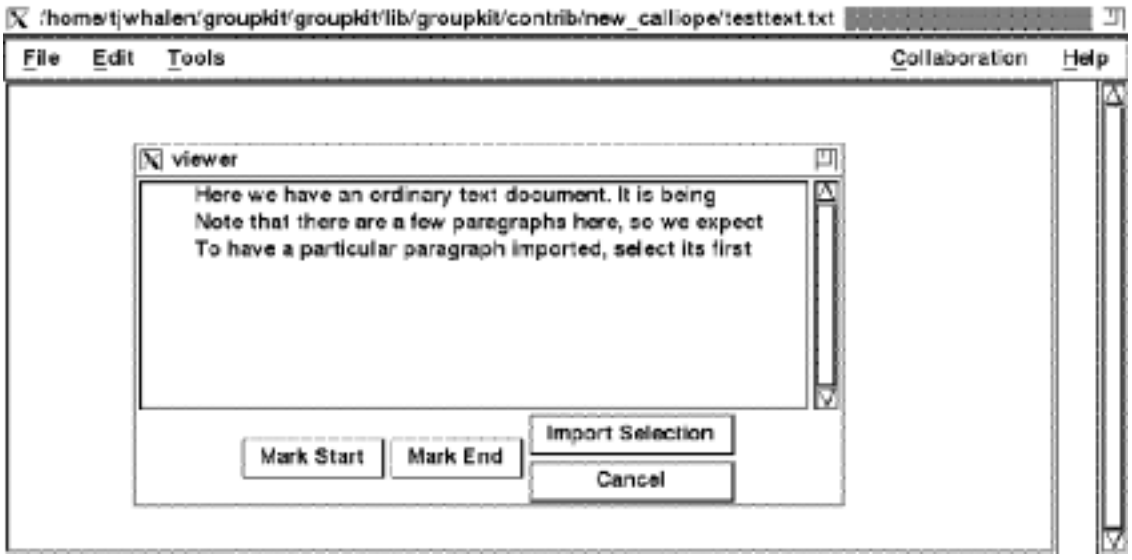

Figure 4.2: The document overview window.

five minutes. This interval can be changed as information is returned about network conditions (e.g. error rates). The file is saved to the local file system of the machine running the Calliope client; this machine could be the mobile host, or it might be a remote machine which has exported only its display to the mobile.

Maintaining consistent stored copies is a difficult problem, as the text versions across different users' machines are likely to differ to some degree. This backup system does not deal with the problem of automatically merging inconsistent copies, but the document backup could be used to make a manual update, if necessary. Mechanisms for managing the consistency of replicated documents exist in other group editors ([13], [12]; the models used in these editors appear to be applicable to Calliope.

4.6.3 Block Updates

Calliope uses keystroke-by-keystroke updating in order to maintain a WYSIWIS interface. Changes are first made locally, before being sent serially to other copies, to ensure that communication delays do not prevent a person from viewing their own changes quickly. This aspect can be preserved, but block updates should be implemented instead of character updates. These blocks can be of varying lengths, depending on network conditions or user preferences. Sending a whole word, sentence or an even larger block of text will cut down on the number of small packets being sent over the wireless channel. Another way to implement block updates would be to use a locked region as a block of text to be updated only when the lock is released. This would allow many text changes to be made, but only requires that one set of changes be sent when the user decides that the changes to the region are complete. This solution is limited in that it prevents users from seeing updates in progress, but it could be included in a set of block update mechanisms from which a user could select. Calliope already uses a limited form of block updating, as both the private text window and the external annotation features only insert text once (when the user has finished entering the text). A variation on this mechanism could be used to support a relaxed WYSIWIS interface, without sending frequent, small updates. Whenever bandwidth improves, keystroke updating could resume, if desired.

4.6.4 Concurrency Control

Calliope uses a replicated locking mechanism, which is useful for a wireless environment, but the lock negotiation procedure can be streamlined. Currently, each user maintains a “list” of all locks being held (these locks are stored as tags in the text widget).

This list is consulted when a lock is requested; if there is no conflict, the lock is granted. Whenever a lock is obtained or released, this new lock information is sent to every users' widget. Modes of operation may complicate the interface unnecessarily, as this algorithm was already fairly efficient. One change that could be made under poor conditions would be for the user to request larger locks, thus cutting down on the number of requests. In other words, the user could be generous in their estimates of how much text they intended to control, and thus request, for example, an entire paragraph rather than its first two sentences. If they later decided to work on the rest of that paragraph, they would not need to make another lock request.

4.6.5 Restricting Group Interface Features

Calliope has four central group interface features that could be disabled or used sparingly: the telepointer, group scrollbar, unstructured text space, and annotations. Each of these features must be considered separately in terms of how it may be used in a limited environment.

The original Calliope code already provides for the telepointer to be enabled and disabled by the user. The group scrollbar and annotations do not need to be disabled, but it could be suggested to the user that they not use these features when conditions are poor. For example, it might be suggested that a comment written in a public annotation not be sent until conditions improve (or after the user has waited a reasonable time), if the comment is not of vital importance or time-critical.

Telepointer

The original Calliope code already provides for the telepointer to be enabled and disabled by the user, but there should be some middle ground that allows this tool to be used under poor conditions. Limiting the number of telepointer updates that are sent would reduce network traffic, allowing the telepointer to be used when conditions are slightly less than ideal. There may be a lower sampling rate that allows reasonable position updates, such as sending an update for every third or fourth mouse motion event, which would allow the telepointer to be supported without overwhelming the wireless link. If the link became even worse, then the sampling rate could be decreased further, which could lead to some interface problems if the user did not receive enough updates to make for a smooth representation of the remote user's gesturing. Again, this is preferable to disabling the telepointer entirely, but should only be invoked if conditions required it. In the very worst circumstances, where the telepointer traffic interferes with more important traffic (such as receiving text), then it could be temporarily disabled, then enabled to a limited degree when conditions improved.

A more complicated solution involves replacing the telepointer with a different type of semantic cursor, one that is less bandwidth-expensive. The creators of Groupkit have proposed some alternatives to the telepointer, such as sending a "point request" by clicking a mouse button over selected text [8]. Similar alternatives, such as selecting a text region of interest, would allow users to maintain the ability to gesture even under adverse network conditions.

External and Public Annotations

The annotations are very simple features, and as such, are difficult to modify for wireless use. The external annotation can either be used or not used, depending on channel quality, but it cannot be reduced any further. The public annotation is similar, but it does have one source of heavy overhead: keystroke updating. Because other users may wish to read the contents of the annotation while it is being written, Calliope supports keystroke updating of the text typed into the public annotation window. (Other users will see an icon that they can use to view the annotation in progress.) The public annotation, like the main text window, could use block updating to cut down on the number of text updates sent over the wireless channel. This change will make the public annotation mechanism more efficient; the only way to reduce it further would be to not use it at all when conditions are particularly unfavorable.

Group Scrollbar

As with the annotations, the scrollbar is a simple feature that cannot be easily converted into a less resource-consumptive version. A reduction in the number of scrolling updates will reduce network traffic, but this can only be done in limited circumstances. For example, if a user wanted to page through a document until they found a certain passage, and felt that the intermediate scrolling did not need to be shared, then they could send an update only when they had arrived at the screen they wished to work on. When one person is scrolling through a long text file, it may be irrelevant for other parties to be following along. Once the right section has been found, the other participants can be sent the proper coordinates so that they can share the same view. This would cut down

on updates that serve little purpose. If it became necessary, this feature could remain unused under adverse network conditions.

Unstructured Text Space

The unstructured text space, or scratchpad, has a number of features within it that make for a complicated group interface feature. It allows for cutting and pasting text from the main document into the scratchpad, for text insertion, editing, and moving, and for freehand drawing with the mouse. Each of these features can be reduced in some way, and their similarities with other features described above suggest that some of the same solutions proposed can be used here. The scratchpad, like the main text window, uses keystroke updates, which may be unnecessarily expensive. A block updating option could be used to cut down on bandwidth consumption. The drawing and dragging features use mouse motion events to generate updates, and thus suffer from the same problems as the telepointer. The same solutions proposed for the telepointer could apply here: fewer updates, as much as can be allowed without degrading the interface too severely. The freehand drawing tool could also be supplemented with a small set of predefined shapes that are used frequently in an editing session (such as circles, question marks, etc.). This would allow a single-shape update to be sent, which should be less expensive than generating the same shape with mouse location updates.

These changes may allow the scratchpad to be used when bandwidth drops, although for very poor network conditions, it is likely that the feature could not be used at all.

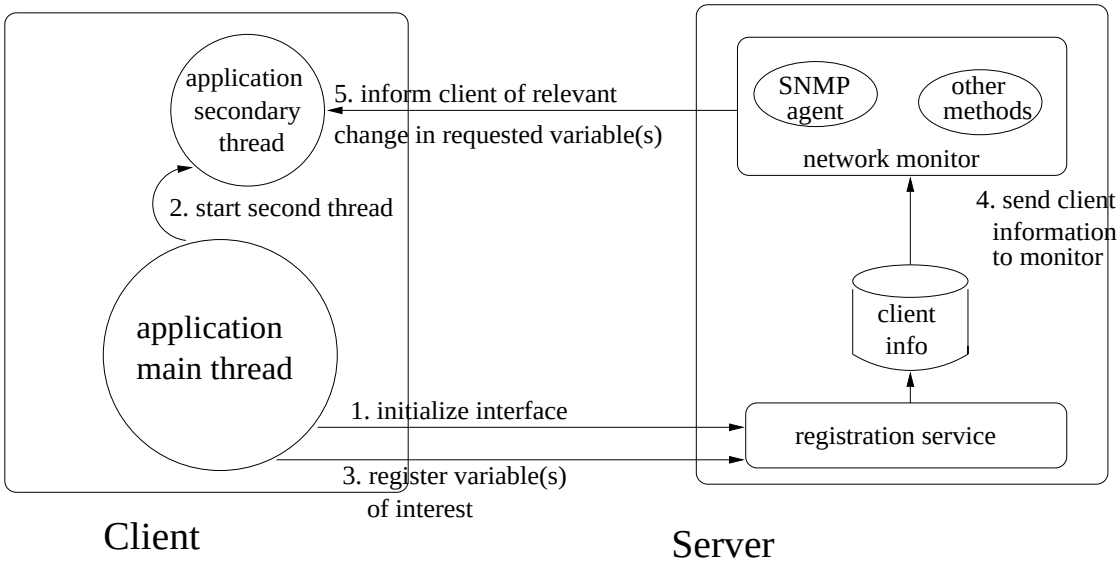

Figure 4.3: Overview of the API and monitors.

4.7 Gathering Network Information

Calliope also needs to have a means of obtaining information about current network conditions, and changing its behaviour accordingly. This information is provided through an application program interface (API), the result of our earlier and ongoing research [19].

The network monitoring system being used through the API allows an application to receive relevant network information through callbacks. The API serves as an interface between the application and a monitoring system which actually tracks network events. The monitoring system includes a Simple Network Management Protocol (SNMP) agent as well as components to track events outside the scope of SNMP, such as CPU load. See Figure 4.3 for an overview of the architecture and basic functions.

The API has three main components: a client library, a client callback function, and

a request-handling server. The client library currently consists of two functions. The first of these is an initialization routine; when the client interface is to be initialized, the client provides this routine with the address of the callback function. The callback function is application-specific, receiving requested network data and determining the appropriate action when notified of relevant conditions. When initialization occurs, a new thread of control is created. This thread opens a port from which to receive network information and then receive-blocks until the server returns requested information.

The second component of the client library is a registration function that allows an application to specify which network conditions it wishes to monitor. There are several components to this request: the network variable of interest (e.g. error rate), a value or range of values of that variable, and an operator (inside or outside range, or equal to value) to specify when a callback related to the variable should be triggered. For example, the client can request a callback when the number of erroneous TCP segments received is more than 50. The port number of the blocking thread is appended to this message, which is then sent to the request server.

The server runs as a heavyweight process on the same machine as the client. When a client request arrives, it is placed in a database, and the server checks all the registered variables (from all clients) to see if they match the conditions for callback. Currently, the server obtains network information by querying the monitoring system, which may obtain variable values through the SNMP agent or other monitors.

If the variable is in the target range, then the relevant information is sent to the callback thread. If it is not in range, then the server checks again after a specified interval or when another request arrives.

The question of which API variables might be most useful for groupware is considered in the following chapter. This chapter also provides a detailed evaluation of how the suggested modifications to Calliope affect its network resource consumption.

Chapter 5

Evaluation

In order to see if changes to Calliope could result in better application behaviour in terms of delays and disconnections, nine elements had to be tested. These elements were the two new features (backup system and document overview), and seven existing features whose implementation and use could be modified (updates, telepointer, scrollbar, locks, public and external annotations, and the scratchpad.). While all these features require evaluation, not all could be throughly assessed. I was unable to come up with a quantifiable measure of usefulness for either the backup system or the document overview. These two features were added to Calliope as part of a plan to convert the entire application so that it would make more effective use of a wireless channel. This would have involved changing the implementation of the other features discussed above so that they could be restricted as channel quality deteriorated. Initial experiences with modifying the Calliope code demonstrated that making it adaptive would have been very difficult, and would not have yielded enough extra information about improvements in channel use to make the effort worthwhile. This experience suggested that adaptability is best imple-

mented during the design phase of application development. Although the new features could not be formally evaluated, they can be said to be useful to some degree.

The backup system is intended to prevent lost work caused by disconnections. Thus, it can only be properly assessed by studying the frequency of disconnections and whether the backup saved work that would have been lost as a result. Certainly, disconnections can be tested, but unless the pattern of disconnection were representative of most wireless systems, the results could not be generalized. Unfortunately, there is no data available on probable disconnection rates for a given wireless environment, as disconnections do not follow a predictable pattern. Therefore, representative disconnection rates could not even be simulated or used as a basis for evaluation.

The current backup interval of five minutes is not likely to be a useful measure in general, as there may be more frequent disconnections in other environments. Furthermore, there may be detectable wireless conditions that threaten disconnection and suggest that a backup should be initiated. A study of these factors would prove very useful for creating an optimized backup system for wireless collaboration.

In the general case, then, all we can conclude is that a backup system will provide some measure of protection against lost work. However, the probability that a backup will be needed is difficult to measure, and the optimal backup frequency is currently undetermined.

The second feature that could not be properly evaluated is the document overview. This feature can be said to be useful if a user saves time by avoiding lengthy delays in receiving relevant text. Therefore, the only way to evaluate the overview is to conduct a series of tests with various users, and see whether people had occasions in which the

overview reduced delays. This sort of field test was too time-consuming for this portion of the research. Furthermore, there is no data available on representative document sizes; such data could have provided an estimate of the proportion of collaborative documents that are large enough to make an overview a useful feature [10]. Thus, at this stage it can only be concluded that the overview will likely reduce delays on those occasions for which it is relevant: editing documents when the majority of the text is not to be used. The probability that that the overview will be useful is currently undetermined.

The other seven features could be evaluated on the basis of how well they reduced network traffic in a collaborative session. These features are those which already existed in Calliope, but which can be used differently or disabled depending on the network conditions. These features were evaluated in a series of tests, described in detail in the next section.

5.1 Evaluating Interface Features

In order to determine when to disable or enable various interface features, there needs to be some information available on what sorts of demands these features make on the network. This information will suggest which are the most bandwidth-expensive features, and thus those that should possibly be disabled first (depending on how vital they are). Those features that consume little bandwidth can be left running through poor conditions.

Tests were conducted on seven of Calliope's features: public and external annotations, telepointer, group scrollbar, locks, scratchpad, and immediate updating. The testing environment (see Figure 5.1) consisted of a two-user Calliope session conducted over a wired network, specifically over two AIX machines linked by a 10 Mbps Ethernet

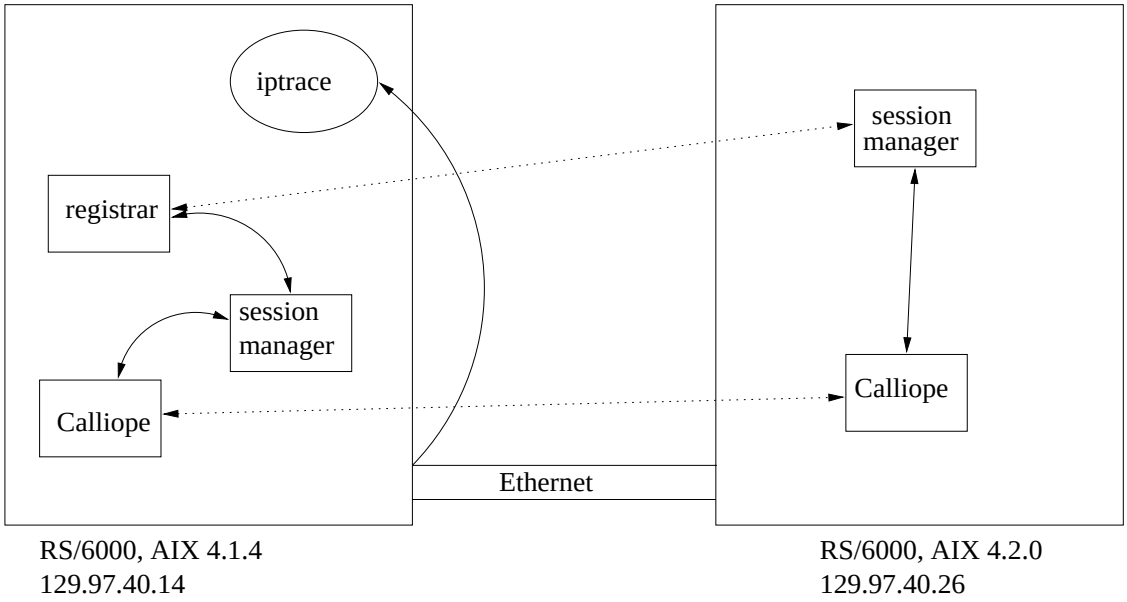

Figure 5.1: Experimental testbed.

connection. One of these machines was running iptrace, a Unix utility that provides substantial information about network packets coming and going from a particular host. Iptrace provides full header information (i.e. IP and TCP), including source and destination ports, and also provides an ASCII representation of the packet contents (see Appendix B for an example of the output from iptrace). The results of the traffic traces are based on two users only, but an examination of Groupkit's multicast mechanism reveals that the communication traffic scales linearly with the number of users. Two tests were performed on every feature, with each machine being evaluated as a sender and a receiver. These bi-directional tests did not reveal any significant asymmetries, except when serialization caused one machine to generate more traffic than it would have otherwise sent (see scratchpad test below).

Figure 5.2 shows a sample of a Calliope session in progress in order to demonstrate the sort of traffic that is transmitted over the network. The session is described in terms of the TCP stream that connects the two users. Some initialization information is exchanged, after which the application-specific commands are sent over the network. The number of commands Calliope sends in a single write to the TCP socket will vary, as this depends on user actions. Groupkit uses a code for every socket write which informs the receiver of how many bytes should be read from the socket: this code begins with RDO and is followed by the number of bytes being sent. Thus, a single write consists of an RDO code plus a series of commands. Groupkit (and hence, Calliope) transmits Tcl commands to be executed on remote machines, so each packet contains such data as "gk_eval [procedure name, parameters]". This data stream may be broken up into several TCP packets, and each packet requires a TCP acknowledgement. Because Groupkit uses an asynchronous communication model, there are no application-level acknowledgements.

Steps 2 through 5 of Figure 5.2 show a single-command socket write broken up into two packets: one for the RDO code and one for the actual Tcl command, each of which is acknowledged by TCP. The use of Tcl commands over the socket makes it very easy to match interface commands with particular packets when iptrace is used to examine the contents, and thus it is simple to determine how much traffic each feature generates. For purposes of calculating the amount of data sent per feature, acknowledgments of data packets were included in the overall count, but initialization and session-concluding commands were excluded.

Because Groupkit sends Tcl commands over the network, the actual procedure names make a small difference to the amount of network traffic. A procedure called "in-

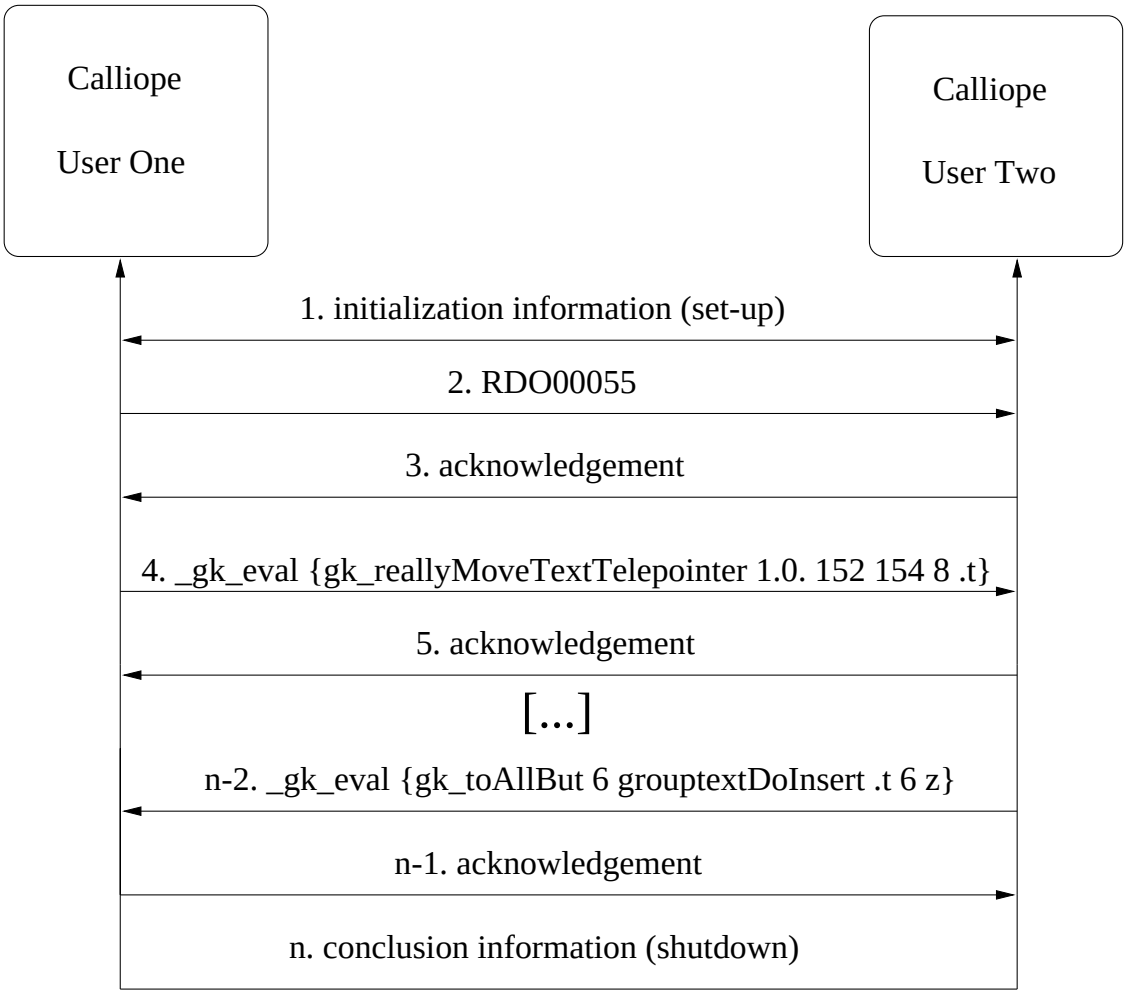

Figure 5.2: Network traffic sent during a Calliope session.

sert_text_block_between_these_points" will send 38 characters for the name alone. Using very short names may cut down on the traffic, but could make the Tcl code very hard to understand. On the whole, this does not seem to be a vitally important problem, but it must be considered if a particularly long command is being invoked several times during a session.

The overall results of the traffic traces are not, in themselves, sufficient to provide specific API variable values that should trigger application changes (e.g. latency of 6 ms. means that the telepointer should be turned off). However, they can be used in a larger interface study to show overall patterns: which features are heavy network users, and which consume few resources. This provides a basis for making tradeoffs between the usefulness of a feature, and its network cost, a tradeoff which must be made with consideration of human factors. Furthermore, the suggestions for feature reduction provide a starting point for a study of the usability of degraded features (e.g. fewer updates) which could be applied to many types of groupware or other applications that need to work within a restricted environment.

5.1.1 Telepointer Test

The telepointer was tested by starting Calliope and moving the mouse on an empty screen. The mouse was moved continuously for several seconds to provide a measure of bytes sent per second of telepointer activity. This test was run with the telepointers on both machines enabled at the same time, and from each of the two machines with only one telepointer enabled at a time, making three tests in total. The motivation behind testing both uni-directional and bi-directional traffic was to discover whether these different

traffic patterns had any effect on the efficiency of channel use.

The results of these tests are in Table 5.1. Both the uni-directional and the bi-directional tests demonstrated similar numbers of bytes sent for each update. The overall number of bytes sent is very large for all tests, showing that there is a great deal of traffic generated by the telepointer. This can be explained by the implementation of the telepointer: for every mouse motion event, Calliope sends an update of the new mouse coordinates to all conference participants.

Calculating an estimate of bytes/second generated by the telepointer is difficult, as the number of telepointer updates will vary depending on such factors as the speed at which the mouse is moved. However, to use the results of Test One as an example, the rate is 1459.04 bytes/second (given that the test lasted 76.61 seconds). This is a very high rate: if the telepointer were used over a 19.2 Kbps wide-area cellular network, then this one feature would consume approximately 60% of the available bandwidth.

These results suggest that the telepointer cannot be easily supported in a low-bandwidth environment. The telepointer can be completely disabled for use over wireless channels if necessary, but there are some implementation changes that could provide some limited use of this gesturing tool.

One small change that could be made is in how the telepointer is toggled on and off.

Test	Number of Procedure Calls	Total Bytes Sent	Avg. Bytes/Call
1 (both tptrs.)	922	111781	121.24
2 (one tptr.)	652	68895	105.67
3 (one tptr.)	468	47306	101.08

Table 5.1: Telepointer test results.

Currently, the telepointer has to be enabled or disabled through a pull-down menu. This means that once a user wishes to stop sending telepointer traffic, they have no choice but to send several mouse position updates by moving the pointer up to the menu bar to select the "Toggle Telepointer" option. If the telepointer state defaults to "off", and a user could hold down a key or button (while moving the mouse) to enable the telepointer, then the telepointer would send less irrelevant traffic, as it would almost always be turned off. Also, decreasing the sampling rate would certainly cut down on traffic. Calliope could be modified to send an update only after a given number of mouse motion events (for example, every fourth) were returned to it by X-Windows.

Neither of these changes appears to be difficult to implement, once a reasonable lower bound for sampling is determined. These implementation changes will cut down on the amount of traffic generated by the telepointer while still allowing it to be used whenever bandwidth permits. Two suggested modifications (defaulting to "off" and decreasing the sampling rate slightly) may not have any detrimental effect on the interface, and if they do not, then this modified telepointer could be used no matter what the current network conditions were. If the API results are to be used to determine whether to reduce the number of updates or completely disable the telepointer, then the most relevant parameter to use is the estimate of remaining bandwidth. This should give an indication of whether the telepointer can be used without it interfering unduly with other features or networked applications.

5.1.2 Annotation Test

The public annotation feature was tested by loading a short text file (1209 characters, 34 lines), then invoking the public annotation feature. This was done from both machines in turn, in separate tests. The public annotation consisted of the phrase "I have nothing to add" with a carriage return at the end, making 23 characters sent in all. (Note that the traffic profile below, as well as the data in the table, only takes into account the traffic generated through the use of the public annotation, and excludes the transfer of the text file data.)

Figure 5.3 below shows both tests on the same plot. Both tests displayed bursts of traffic, which was expected due to the fact that every character is sent after it is typed. However, the size of these packets appeared to be extremely high for a single-character insert. An examination of the trace files revealed that there is a great deal of redundant data being sent along with the characters. The implementation of the public annotation is very inefficient: the annotation window inherits features from the main text window which are never displayed, but are nonetheless updated. The telepointer and group scrollbar coordinates within the smaller annotation window are relayed to all other participants, but are never displayed on the screen. Thus, we have a great deal of irrelevant and useless information being sent over the network. Every time the mouse is moved, for example, to select "Close Annotation," a telepointer update is generated. Also, whenever mouse button one is pressed or released, a group scrollbar update is sent. The basic text insertion mechanism, as described above, is also a source of heavy overhead, sending several packets with one character in them. With a packet overhead of 40 bytes (20 bytes for each of the TCP and IP headers), this is a very inefficient way to transmit data.

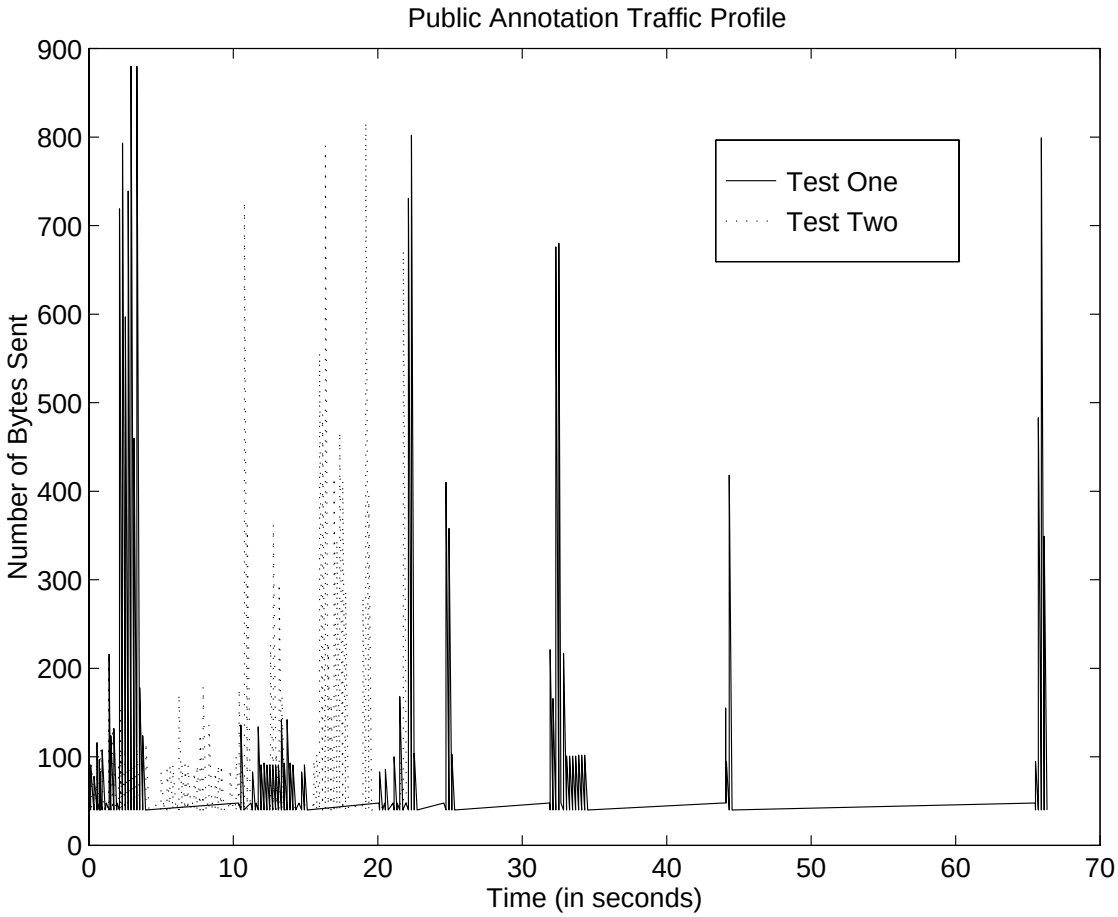

Figure 5.3: The two public annotation tests.

The total bytes per character are summarized in Table 5.2 below. The average data rate over both tests is 829.41 bytes/character, which represents an enormous amount of traffic for such short messages. Obviously the amount of traffic will fluctuate depending on how a person uses the mouse during the annotation-writing period. (However, there is no indication on the screen of these updates, so the user is unaware of all the traffic being sent.)

It would appear to be an implementation problem that allowed the telepointer and group scrollbar traffic to be sent without ever being used. It is conceivable that this information could have been used in a later model of the annotation window, if the telepointer were likely to be used during the creation of an annotation, and if the annotations were long enough to warrant the use of a group scrollbar to ensure a shared view. However, the current version of the public annotation mechanism neither needs nor uses these features, and so the implementation needs to be changed to get rid of the extraneous traffic.

Furthermore, it may not be necessary for the annotation to be sent keystroke by keystroke. If the annotation were sent after the user finished writing it, then the heavy overhead of keystroke updates would be eliminated. In many situations, users would not need to see an annotation being written, and thus sending the whole annotation at once this would be a simple way to cut down on heavy overhead. However, this change should only be made after a study of how people use annotations. It is possible that the

Test	Characters Sent	Total Bytes Sent	Avg. Bytes/Char
1	23	20659	898.22
2	23	17494	760.61

Table 5.2: Public annotation test results.

keystroke updates might have to be left as a user option, even though it makes for inefficient channel use. The addition of this feature should be very simple, as provision for block updates already exists in the private text feature and could be adopted for use in writing public annotations.

If the keystroke updating is left as an option, then the API could be used to suggest whether a user ought to be considering using it. If there is little remaining bandwidth, then these updates may interfere with another feature or application that the user considers more important than sending each keystroke of their annotation text.

5.1.3 External Annotations

The external annotation was tested without loading a text file: each user simply selected the feature and entered the URL "http://www.somewhere.com/~user". This was a very simple test, and the resulting traffic is summarized in Table 5.3.

There is very little traffic generated by the external annotation. Unlike the public annotation, the characters are not sent one at a time. Thus, all the text is delivered at once, probably in one packet. It is unlikely that very much text will be sent through the external annotation, but the small amount of text that is sent is sent as efficiently as possible. The test results demonstrate how few resources are consumed by this feature: beyond the basic text for the URL, an external annotation carries an overhead of approximately 400 bytes. (This figure could increase if the full URL could not fit in one TCP packet, but this is highly unlikely.)

The well-designed implementation of the external annotation, along with the fact that little text is sent, allow this feature to be used under all but the very worst network

Test	Characters Sent	Total Bytes Sent
1	30	433
2	30	433

Table 5.3: External annotation test results.

conditions. The only possible change that could be suggested is that users avoid sending external annotations when the channel quality is extremely poor.

5.1.4 Keystroke Updates

It is useful to have a measure of how expensive immediate updates are in relation to block updates, so two sets of tests were performed to compare both sets of updates. The first set of tests involved sending 167 characters in the main text window, so that each character was sent immediately. Each test consisted of one machine being sender and the other receiver, and was done twice to allow each machine to perform both roles.

Table 5.4 below shows the total bytes sent and the number of characters for each test. Averaged over the two tests, the data rate for keystroke updates is 132.21 bytes/character. The trace data reveals that for every character insertion, there is an approximate overhead of 216 bytes plus the number of characters. Thus, five single-character insertions use 1085 bytes, and one five-character insertion uses 221 bytes. As with the public annotations, this represents a very high overhead for single-character updates. As typing is likely to be a very common activity in Calliope, this inefficiency is a very important factor for wireless collaboration.

One option that was suggested above was to use block updates. It is difficult to say whether this decision will work from a human-computer interface perspective, but it will

certainly lead to better use of the channel. This option was tested with basically the same test as the keystroke update test above, except that the text was entered into the private text window, and thus was only sent when the user selected the “Insert [text]” option.

The results are dramatically different from the keystroke updates. There were only four packets sent: one to send initial Groupkit codes, one with the whole text message, and one acknowledgment for each of these two packets. Overall, the block update data rate is far better than the keystroke update rate, which can be seen in Table 5.4. Averaged over the two tests, the data rate is 2.31 bytes/character, which is 2% of the traffic that keystroke updates produce.

If block updates could be used reasonably in the context of groupware interfaces, then the remaining bandwidth variable returned from the API would be of great use in providing information as to whether keystroke updates were proving too bandwidth-expensive and needed to be replaced by block updates. It is also possible that some sort of keystroke bundling could be done at the application layer, so that a group of characters would be sent at a time. For example, assuming an average word size of five characters, sending text one word at a time would generate one-fifth of the traffic of single-keystroke updates. This would allow more frequent updates to the shared text without bearing as

Test	Characters Sent	Total Bytes Sent	Avg. Bytes/Char
Keystroke Updates 1	167	22374	133.98
Keystroke Updates 2	167	21785	130.45
Block Updates 1	167	385	2.31
Block Updates 2	167	385	2.31

Table 5.4: Keystroke versus block update results.

heavy a burden of overhead as single-character updates would carry. Again, the API information on bandwidth could suggest an appropriate number of characters to be bundled together for transmission, so that more frequent text updates could be supported as network conditions permitted. Adding a variable-size block update (such as a word or line) is moderately difficult because Calliope has many components, such as locks, that assume that keystroke updates always occur. Furthermore, features such as document saving must take into account the fact that the document copies may be slightly inconsistent. Another difficult portion of this modification is determining an appropriate block size, given values from the API and what is the least confusing for the session participants. The implementation effort suggests that it might be simpler to build an entirely new editor rather than modify Calliope; considering the enormous amount of traffic likely to be sent through keystroke updates, this effort seems worthwhile.

5.1.5 Group Scrollbar

The group scrollbar was tested by loading a large text file (10,259 characters, 345 lines) so that several screens of text appeared. There were two tests, one from each machine with User One following User Two's scrolling. The scrolling was fairly constant, paging from the top to the end of the text. There was some variation in the number of scrolls this took, as the same selection point on the scrollbar (i.e. where the mouse was clicked) could not easily be chosen twice. Thus, Test One used 13 clicks to page through the text, and Test Two used 15 clicks. However, because the scrollbar updates are bound to mouse button presses and releases, more scrollbar updates can be sent than the user intends. If, for example, a mouse button were accidentally pressed and released twice on the same

spot of the scrollbar, the screen view will not change, but an update will be sent. (Again, the traffic analysis below does not include the loading of the large file, only the group scrolling traffic.)

Figures 5.4 and 5.5 below show that the traffic consists mainly of several packets of about 115 bytes each, along with several packets of about 40 bytes (acknowledgments). The 115-byte packets are scrollbar updates. There was more traffic in this trace than anticipated, and the trace revealed that the scrollbar updates came in pairs: whenever User One would send, User Two would send an update right back. An examination of the implementation revealed that the group scrollbar is designed to send updates whenever its position is changed, regardless of whether or not the user is the one controlling the shared view. Suppose that Bob has chosen to follow Alice's scrolling. When Alice scrolls, her new coordinates are sent to Bob to allow his copy to shift to the shared view. However, Bob's copy then will send an update back to Alice with his new scrollbar coordinates, which is completely unnecessary, given that Alice is not going to change her view to match his in any case, and would not even need to, as the views already match.

Thus, this implementation leads to twice as much traffic as necessary for a group scrollbar update. For each invocation of the scrollbar update procedure, there is an average of 201.89 bytes (average over the two tests). This is not a lot of traffic from each procedure call, considering that most of the data sent is the name of the Tcl command. However, there are too many of these update requests being sent. If the implementation were changed so that only the necessary scrollbar updates were sent, then the traffic would be cut in half.

If still further traffic reductions needed to be made, then a user could be allowed to

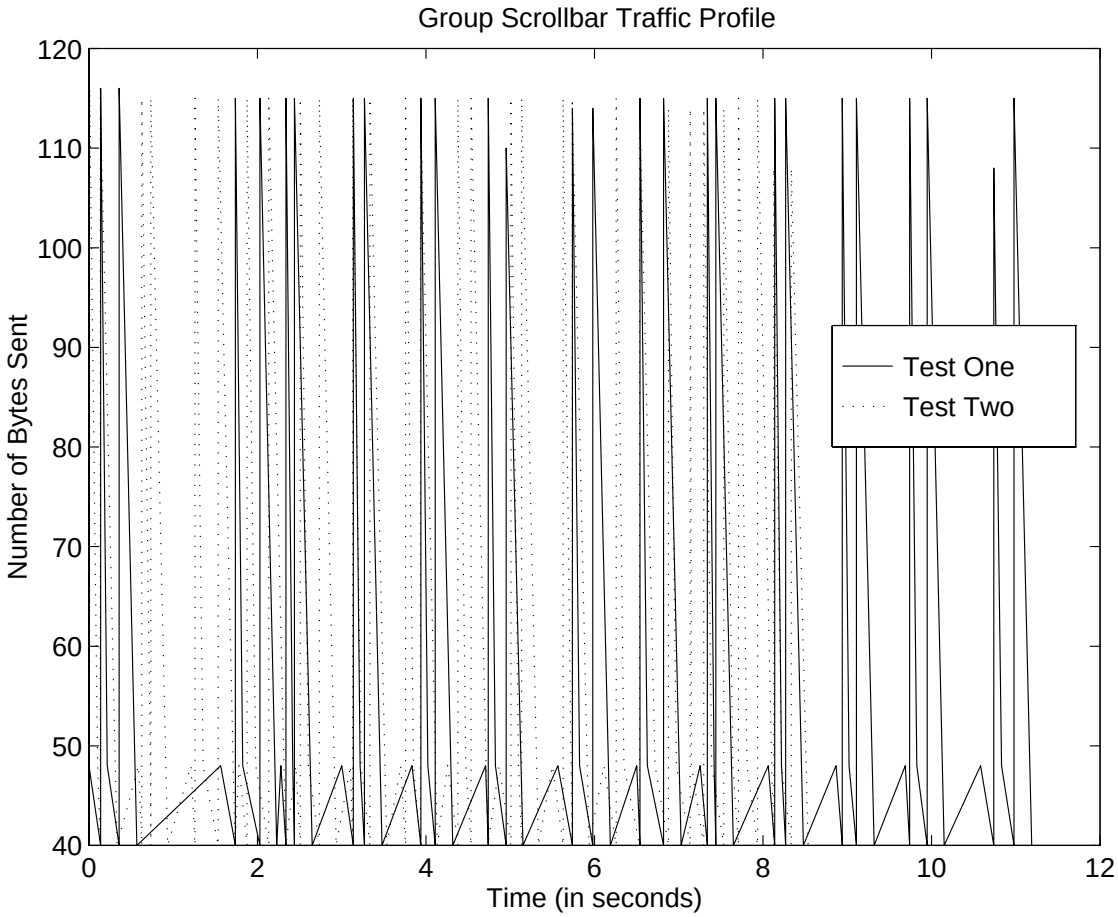

Figure 5.4: Group scrollbar traffic tests.

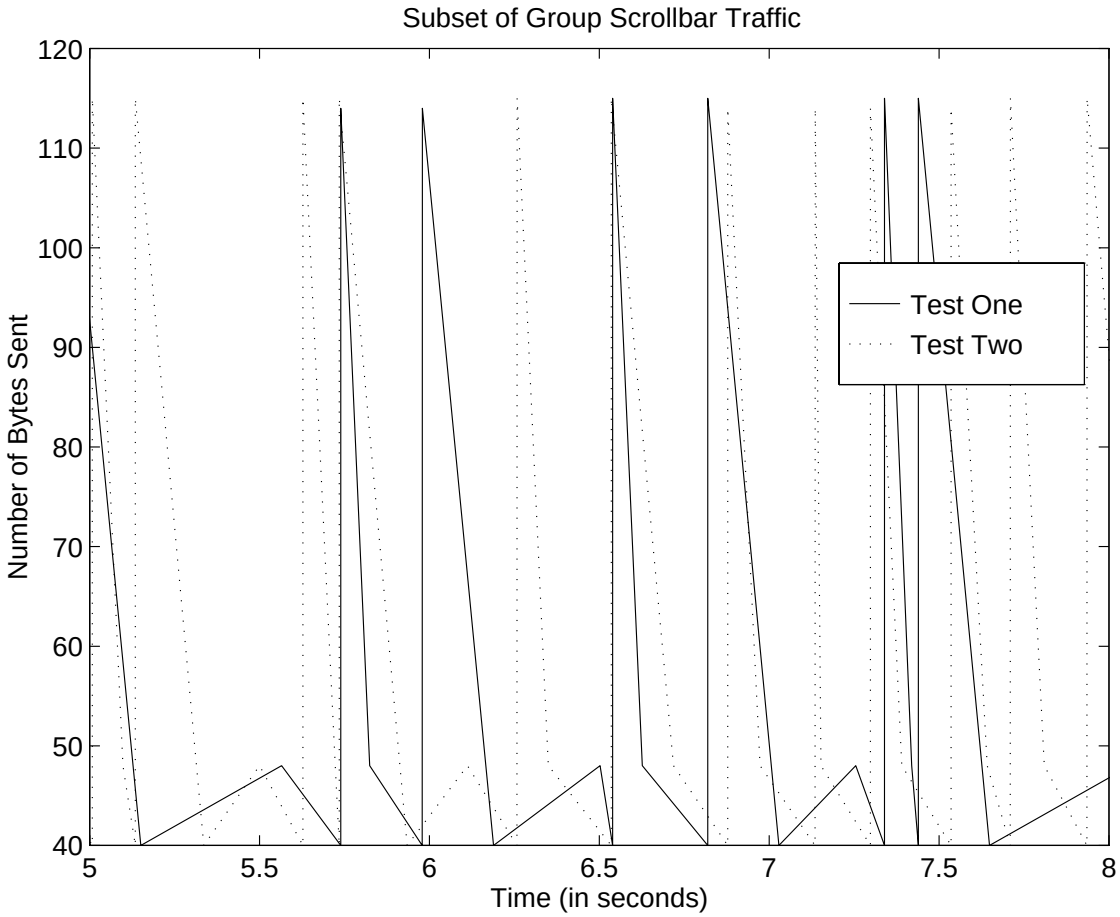

Figure 5.5: Closeup on section of group scrollbar plot.

send an update only when they had arrived at the screen they wished to work on. When one person is scrolling through a long text file, it may be irrelevant for other parties to be following along. Once the right section has been found, then others can be sent the proper coordinates so that they can share the same view. This is a very simple option to add, and would cut down on updates that serve little purpose. However, as this sort of scrolling may prove confusing or jarring, it needs to be investigated in a user interface field study.

5.1.6 Concurrency Control

There are four types of locks that had to be tested: selection, word, line, and paragraph. Each lock was tested by loading in a short text file (1209 characters, 225 words, 34 lines), then selecting the lock type and grabbing the appropriate region. The selection lock was used to lock the first paragraph, while the word lock was used on the second word. The line lock was used on the first line of the second paragraph, and the paragraph lock was used on the second paragraph.

These tests were done on both machines, with one machine's user being the lock selector each time. Table 5.6 displays the amount of traffic sent. The amount is very similar for most tests, except for the lock selection test. The lock selection mechanism

Test	Scrollbar Procedure Calls	Total Bytes Sent	Avg. Bytes/Call
1	31	5786	186.64
2	30	6514	217.13

Table 5.5: Group scrollbar test results.

Test (Lock Size)	Region Size (chars)	Total Chars Sent
Selection 1	114	4450
Selection 2	114	5533
Word 1	6	305
Word 2	6	463
Line 1	64	587
Line 2	64	463
Paragraph 1	141	577
Paragraph 2	141	463

Table 5.6: Variable-size locking test results.

sends a lock request for every change in the selection point; that is, when the user drags the mouse across the text to select it, a lock request is sent for each change in the lock region. Thus, to select a region of 114 characters by dragging across the text took an average, over the two tests, of 43.79 bytes/character in region selected. This will vary, depending on how the text is selected (how many changes to the lock region are made), but it is still a high per-character overhead.

If the lock selection implementation were changed to allow the user to choose selection start and end points before making the lock request, then the amount of traffic generated would be similar to that of the word, line, and paragraph lock requests.

For locks that surround a pre-defined unit of text (word, line paragraph), the amount of text sent is very similar, as each lock request merely sends the appropriate set of lock start and end locations. Thus, these lock requests are fairly efficient, with similar amounts of data being sent over the channel regardless of region size. The average amount of data sent, over all tests of word, line, and paragraph locks, is 374.67 bytes per lock request for every participant. However, there is an implementation problem that

generates unnecessary traffic: there are group scrollbar updates sent along with the lock requests. This is because the scrollbar updates are bound to mouse button presses and releases, so clicking to select start and end point for a lock (either a pre-defined unit or selection lock) will also send scrollbar traffic. This traffic is sent regardless of whether the conference participants are using the shared scrollbar, so this can be construed as unnecessary traffic. Changing this implementation detail will help make the lock requests more efficient. Without the extraneous scrollbar updates, the number of bytes sent for a single lock request would be approximately 227 bytes.

Once these implementation changes are made, the change suggested in Chapter 4 can be used to reduce traffic further. This change involves reducing lock updates by suggesting to a user that they select larger locks rather than requesting a series of small locks to control the same area. However, as the current lock negotiation method does not generate very much traffic, this may be too much programming effort and may complicate the interface unnecessarily for a very small increase in channel use efficiency. This would have to be investigated further within the context of a field study of interface changes.

5.1.7 Scratchpad

The scratchpad supports several types of unstructured activity: pasting text from the main document, moving and editing text fragments, inserting new text, and freehand drawing. These components make it impossible to create a standard test, as such testing actions as dragging text and drawing cannot be reproduced exactly in repeated tests. Instead, the tests are meant to demonstrate the amount of traffic generated by a single session of use. The first two tests were performed by having each user manipulate the scratchpad

in turn. Text (nine characters) was entered into the scratchpad, and this fragment was then dragged to the bottom of the scratchpad. Some lines were then drawn, and the text was erased and replaced with another 13 characters. (Note that a documented bug stated that pasting text into the scratchpad did not always work, and this proved to be true in these tests. Thus, this feature could not be tested with the others.) The third test required both users to work on the scratchpad at once. The same actions as above were performed, except that the text was erased and replaced with 9 other characters. The actions were interleaved, and each action (drawing, dragging, etc.) was performed by both participants. The traffic patterns are displayed on Figures 5.6, 5.7, and 5.8 below.

The differences in the three tests are striking, which reflects the difficulty of making a standardized test. Despite the fact that tests one and two are essentially the same test, the traffic pattern is radically different. The trace files reveal why this is so: the second test required serialization of the text insertion. If there are two or more users engaged in a conference, one of the conference processes can be used to serialize an action, to ensure that in-order delivery can be maintained for all users (see the Groupkit description, Chapter 3). In test two, the local conference was not selected as the serialization copy, so all inserts were sent to the second user to be serialized. Then the insert commands were sent back to the first machine, and the text was then inserted into the scratchpad.

This arrangement leads to small bursts of traffic as each insertion was sent to be serialized (with keystroke updates), followed by large packets with all the actual insertion commands bundled together. Only the text insertion uses serialization, so the drawing and dragging commands do not suffer from this extra traffic. Unfortunately, if an application needs to use serialization, then this extra traffic must be sent. If this traffic

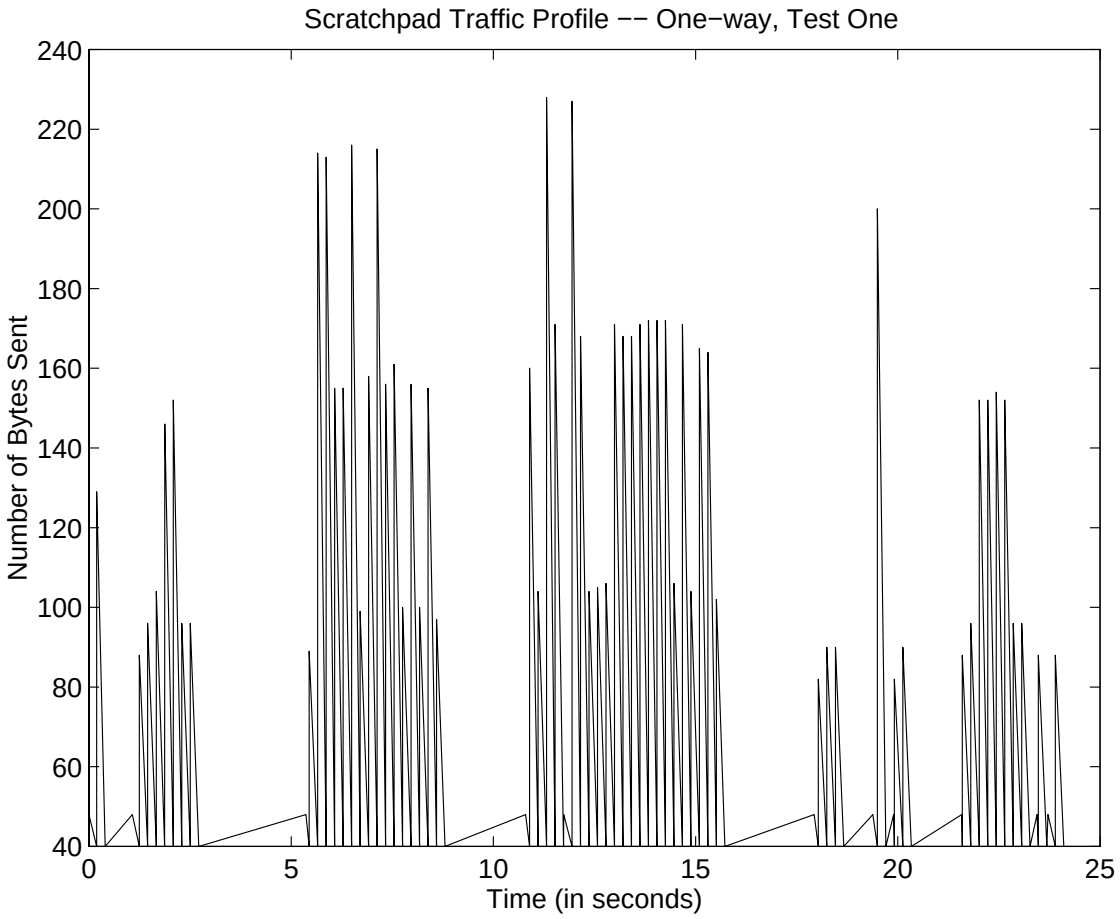

Figure 5.6: First scratchpad traffic test.

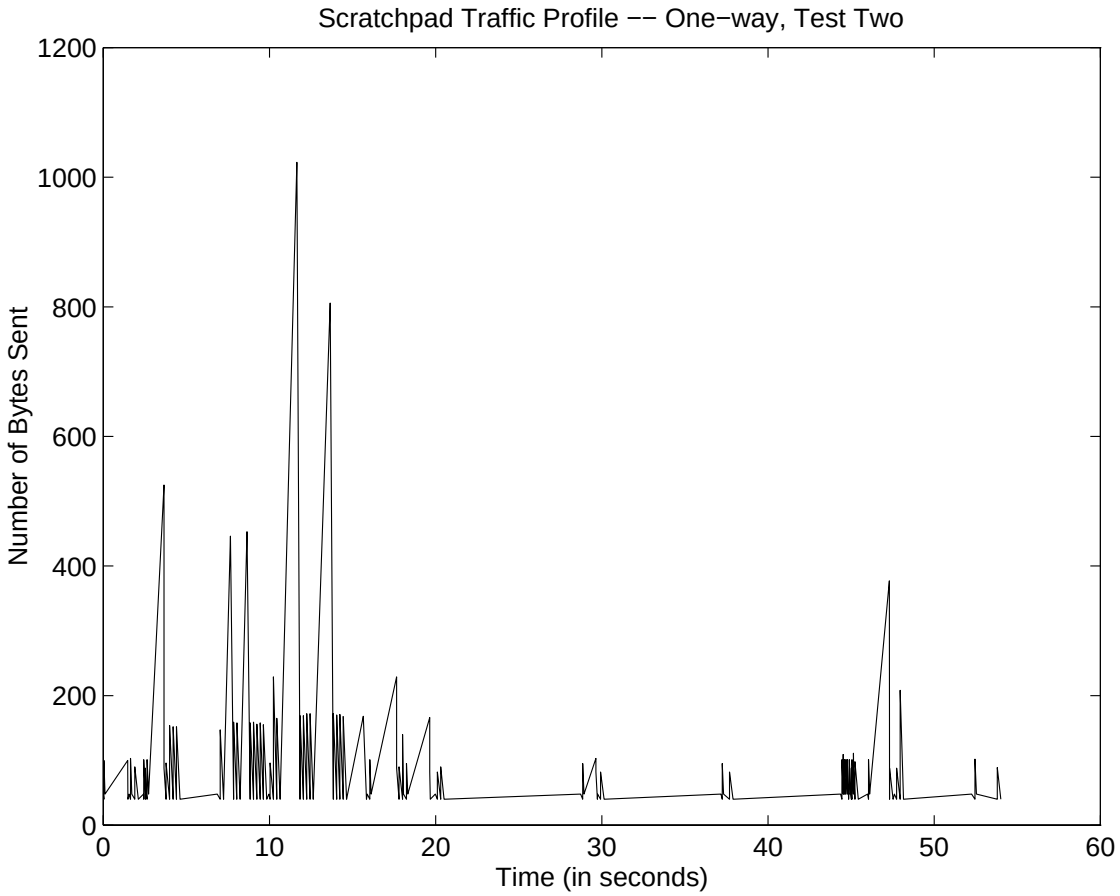

Figure 5.7: Second scratchpad traffic test.

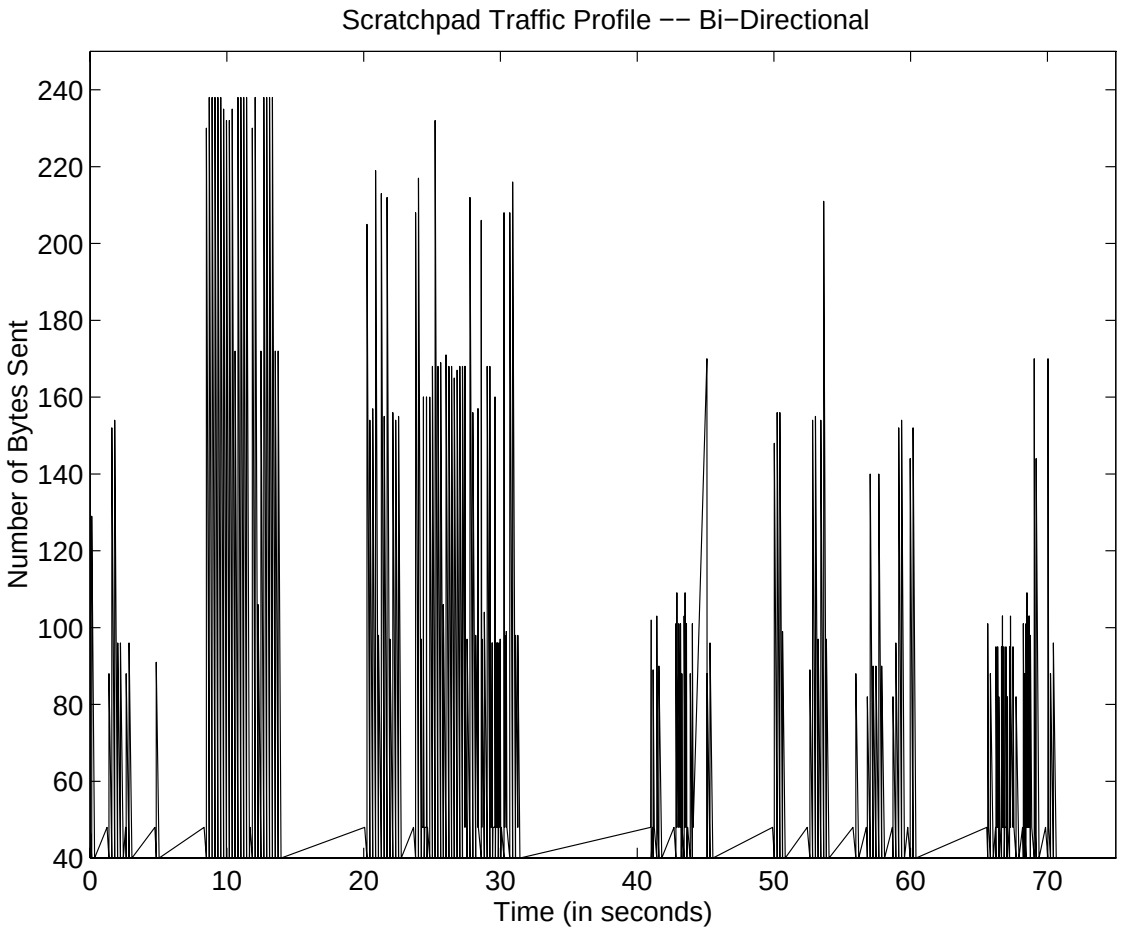

Figure 5.8: Third scratchpad traffic test.

is to be eliminated, then applications must deal with possible out-of-sequence actions, which may make the application confusing to use. Besides extra traffic, serialization can cause an additional problem for mobile users. If some action, for example, inserting text, must be processed at another machine before the results appear on the screen, then communications delays must be considered. Extensive delays, such as occur over wireless networks, may prevent a user from seeing her own actions for a very long time, which can lead to frustration and confusion.

It is difficult to tell from reading the graphs, but the second test generated more traffic than the first, due to the serialization. As is to be expected, the third test generated about twice as much traffic as any of the one-way scratchpad tests. Table 5.7 shows the total bytes sent, and the total time for each test. A bytes/second rate calculated from this type of test can be highly variable, as it includes pauses and composition time in the total time. Thus, the more pauses, the lower the overall data rate, which is not a reasonable basis for drawing exact conclusions about the efficiency of the scratchpad. These results can only be used to give an estimate of the amount of traffic that might be generated by use of the scratchpad.

Analyzing the results generally leads to the conclusion that there can be a lot of traffic generated by using the scratchpad. Because there are so many different aspects

Test	Total Time	Total Bytes Sent	Avg. Bytes/Second
1 (one-way)	24.103240	11696	485.25
2 (one-way)	54.006093	16775	310.61
3 (two-way)	70.647741	35350	500.37

Table 5.7: Scratchpad test results.

of this one feature, small changes could be made to each of its subcomponents in order to balance resource consumption with a rich interface. The changes suggested above (block updating, reducing update frequency, sending pre-defined shapes) should serve to reduce this heavy traffic. Once these first two modifications are developed for other features, they can easily be extended for use with the scratchpad. The addition of a set of basic drawing shapes requires more work, but is not complicated: the most difficult part would be to determine a reasonable basic set of shapes to present to the user. However, the scratchpad is still a very complex feature that consumes many resources. While these changes may allow the scratchpad to be used when bandwidth drops, for very poor network conditions, it is likely that the feature could not be used at all.

5.1.8 Summary

Figure 5.8 gives a summary of the features that are already implemented in Calliope, along with suggestions for how they could be made adaptive in order to perform better under various levels of connection quality.

To make these conditions more specific would require an integrated test of the user interface and how network delays caused by these features influence a group session. The central purpose of this research is to reveal ways in which group editing can make better use of a channel, which should cut down on delays and make a session more coherent. However, the notion of delay in the user interface is complicated: it is not enough to reduce delay by changing the interface, as this may make an application more difficult to use. Furthermore, it is not a very simple matter to determine how much delay is detrimental to the interface [23]; this is an issue which would require further

	Suggested Changes Under Given Condition		
Feature	Slightly Poor	Fairly Poor	Extremely Poor
Public Annotations	No change	Use block updates	Use sparingly
External Annotation	No change	No change	Use sparingly
Telepointer	Decrease number of updates	Disable	Disable
Group Scrollbar	No change	No change	Send final scroll only
Locks	No change	No change	Request large locks
Scratchpad	Decrease number of updates and use pre-defined shapes	Use block updates	Disable
Keystroke Updates	Send few characters at once	Send more characters at once	Send entire block at once

Table 5.8: Summary of Calliope features and suggested changes.

experimentation with users of the application.

5.2 Comparison of Original and Adaptive Calliope

In order to gauge the usefulness of the adaptive version of Calliope, it is helpful to present a scenario in which both the original and the adaptive editors are used to do the same actions. This scenario can be used as a basis for comparing traffic generated by both models, as well as providing a sense for what the user will experience while using an adaptive editor.

The scenario is that of two people, Alice and Bob, collaborating on a short memo. Alice is working on a laptop, and the wireless channel she is using degrades from good quality to fairly poor quality during the session. The actions are conducted on the text in Figure 5.9, which was composed prior to the start of the session. For the puposes of this

scenario, it is assumed that the channel quality decreases between actions 1) and 2).

Memo text:

Internal Memorandum

From: Head of Supplies
To: All Departments
Subject: Change in Order Forms

There has been a recent change in the Office Supply Ordering Form, which is now Form I-6785 rather than Form I-7856. All those who wish to order supplies must now use use Form I-6785, including any person wishing to order Office Supply Ordering Forms (Form I-6785).
Note that all copies of obsolete Form I-7856 will be collected at the end of the day. Following departmental regulations, any person found in possesion of Form I-7865 after this date shall be destroyed immediately along with the requisioned forms.

Figure 5.9: The document being modified in scenario.

Actions (after document is loaded)**:**

1) Alice changes “immediately” to “summarily” (9 chars. deleted, 7 inserted).

2) Bob requests a lock to change “who wish to order” to “wishing to order”; lock is granted and text is changed (4 chars. deleted, 3 inserted).

3) Alice changes “posession” to “possession”(1 char. inserted).

4) Bob requests a lock to change “any person” to “all persons”; lock is granted and text is changed (1 lock request, 2 chars. deleted, 3 inserted).

5) Alice uses a public annotation to send “Replace ‘departmental regulations’ with ‘official procedures’?” as suggestion to Bob (62 chars. inserted).

6) Bob requests lock to change “departmental regulations” to “official procedures”; lock is granted and text is changed (24 chars. deleted (whole phrase deleted), 19 inserted).
7) Alice adds “office” before “supplies” in line 3 of main text (6 chars. insrted).

If the original Calliope application were used in this situation, there would be keystroke updates for the main text and annotation, and there would be no suggested lock sizes. For the purposes of the scenario, it is assumed that line locks are requested for actions 2), 3), and 6).

For these actions, we can construct an estimate of the traffic sent. It is useful to consider two different versions of non-adaptive Calliope: the original, inefficient implementation, and an improved version which no longer performs such actions as sending redundant data. The actions performed are the same in both cases, but the amount of traffic for each of these actions will differ.

For both versions, every character update carries an approximate overhead of 216 bytes plus the number of characters. This traffic analysis for the more efficient version assumes that the cost of an annotation text update is the same as an update in the main text window. (Note that a deletion is the same as an insertion: Calliope “inserts” a blank space.) The annotation is more costly with the original implementation, with an average cost of 829 bytes per character. Finally, the original Calliope lock requests require an average of 375 bytes, while the improved version requires only 227 bytes.

The actions under the inefficient implementation Calliope give the results shown in Table 5.9, while the improved version results are presented in Table 5.10. The improvements make an enormous difference to the amount of traffic generated: the more efficient

version generates only 45% of the bytes used by the original Calliope. As the data being sent in the original version is redundant, there are no detrimental effects caused by removing it from the communication channel. Thus, this scenario demonstrates that such inefficiencies can have serious effects on the usability of an application, particularly if it is to be used in a resource-poor environment.

Action	Bytes Sent
1	3472
2	1894
3	217
4	1460
5	51398
6	9706
7	1302
	Total: 69449

Table 5.9: Traffic sent by original Calliope.

Action	Bytes Sent
1	3472
2	1746
3	217
4	1312
5	13454
6	9558
7	1302
	Total: 31061

Table 5.10: Traffic sent by improved but non-adaptive Calliope.

The situation is different when adaptive Calliope is used. The first action is the same,

but when the channel quality drops to fairly poor, the actions are modified to suit these new conditions. The keystroke updates are no longer used: instead, an update is sent for every tenth change. (Note that the user would need to explicitly request an update be sent if their changes involved less than ten characters.) The annotation now uses a block update mechanism for the entire text, so Bob will not see the annotation text until Alice finishes typing it. As the annotation is not so important that he needs to follow along with its development, this is not a significant reduction in the feature's usefulness. Finally, Calliope will recommend that the user be optimistic and select large locks if they will be making multiple changes to a region. Thus, Bob requests a paragraph lock at action 2), as he intends to change other parts of this region (action 4)). This action could, under other circumstances, lead to a reduction in concurrent actions on adjacent text regions; if Alice wanted to change line 1, she would be able to do so if Bob requested a line lock on line 2, but not if he had a lock on paragraph 1. In this particular scenario, there is no such conflict.

These changes lead to the results in Table 5.11. The total traffic sent by adaptive Calliope is approximately 21% of the traffic sent by the non-adaptive version, and there is not a significant reduction in the usability of the editor. For the price of making users wait a short time for updates, and possibly interfering with concurrent text changes in a region, the number of bytes sent is dramatically reduced. This result suggests that the adaptive version of Calliope makes a reasonable trade-off between interface features and bandwidth consumption, and is a suitable model for editors being used over a wireless channel.

Action	Bytes Sent
1	3472
2	450
3	217
4	437
5	278
6	1350
7	222
	Total: 6426

Table 5.11: Traffic sent by adaptive Calliope.

5.3 API Variables

The second part of this research involved selecting a representative set of network conditions of interest to adaptive groupware. These variables can be used for two purposes: those designed to allow the application to adapt, and those that can be relayed to the interface to give the user more information about network conditions that could affect a collaborative session.

The most important condition that affects a collaborative session is bandwidth. Whether there is enough bandwidth to support certain actions is the key factor for groupware adaptability, and a quantitative measure of remaining channel capacity is necessary in order to adapt the application appropriately. This will lead to more efficient channel use, with delays reduced along with bandwidth consumption.

Thus, an accurate measure of current channel capacity is necessary for appropriate adaptation of the groupware. Furthermore, a measure of what the remaining channel capacity is going to be in the near future would be extremely useful. Currently, appli-

cations can only request recent bandwidth information (bytes sent and received), which may not necessarily reflect what the connection quality is by the time the application is notified. An estimate of bandwidth consumption in the next few seconds may not be very accurate, but is worth investigating. It may need to use other API variables to derive an appropriate estimate, such as error rates or signal strength.

If the idea of delayed updates is supported (as in Koch's IRIS editor), then a measure of latency might be the most useful variable to use. This would give an idea of how long it would take for an update to be received, and thus a user might choose to wait until the delay were shorter before attempting to send or receive. A person might wish to spend some time composing a portion of a document while waiting for low latency which would allow them to receive rapid feedback on their text. This could be preferable to sending an earlier version under high-latency conditions and waiting a long time for responses from others.

Another useful variable, if it could be obtained, would be a measure of "probability of disconnection." This could be derived from the strength of the channel signal: if the signal is weak, then the mobile may be about to cross a cell boundary or disconnect. This could suggest to a user that they may want to backup a document in order to weather a disconnection. This feature may not be reliable or possible, however, and because disconnections are unpredictable, it would not be able to warn of all disconnections.

There are a number of variables whose results could be passed up to the user interface to make for a more coherent and coordinated conference session. Latency, for example, would be a valuable piece of information to give to the user. If a user knows that updates are slow, they will not expect other users to respond immediately to a correction or a

suggestion in an annotation. While extensive delays cannot be completely eliminated over a slow channel, this sort of feedback should help to reduce the frustration that can be caused by allowing the user to adjust his expectations of response time. Currently, latency is measured as the time to the default router, which for a mobile user would be the time over the slow wireless link. However, a more appropriate measure might be round-trip time to each of the other conference participants' machines. The time to complete the wireless segment of the network might not be the best time to measure if participants are very far away from one another. In such a case, the wired portion of the network would represent the bulk of the round-trip time. However, round-trip time is likely to be difficult to measure, as it would require test packets to be sent to all other machines in order to monitor this variable continually. It is possible that sending this test traffic will cause more problems than it solves, as it will be consuming the scarce wireless network resources. This is a problem which requires further study and testing of alternative timing methods in order to be implemented.

Information about signal strength could also be included in the interface, if it could be sent to other users. Then an overall picture of the stability of a session could be used to make the session more coherent. If everyone knows that a particular user is not strongly connected, then they will know that she may be out of contact for a few moments. This information can be found in the SNMP WaveLAN MIB [25], which has not yet been integrated with the API.

User interface components which include information from other participants' networks would require extra network resources for sending out the updates. For example, information about signal strength might be updated every 10 seconds, and each user

might send this information to all other participants every time it was updated in order to maintain a complete and recent picture of the stability of the session. This network information traffic might consume too many network resources and interfere with the editing session. Thus, if it were to be implemented, it would have to be included in the overall adaptive interface; if, for example, conditions worsened, fewer updates could be sent, or a user could request that updates stop being sent to their machine.

In terms of the current variables of the API, this research suggests that many of the variables are not useful for groupware. The variables that measure the wireless channel use (bytes_rx, bytes_tx) are very useful, and the measure of latency is fairly useful, if limited. If some of the extensions suggested above are implemented, then it is conceivable that they will be derived from other SNMP variables that are themselves not explicitly required by groupware. Hence, merely because a variable was not included in this short list of useful measures does not mean that it should not be monitored, as it may be used to calculate a variable needed by groupware. It is conceivable that the application could receive these primitive variables and perform the necessary calculations (e.g. averages over time) to create a derived value. However, this task seems more suited to the capacities of the API, so the application designer could decide to only use those values returned directly through API variables. This might provide a limited number of variables, but ensures that the application is not burdened by calculating derived values. How to obtain these derived variables useful for groupware (future bandwidth estimate, round trip time, disconnection probability) is a problem that requires further investigation.

The following chapter concludes the thesis and proposes areas of future investigation that were suggested by this research.

Chapter 6

Conclusions and Future Work

There are number of changes that can be made to a group editor to make it run more effectively over wireless channels. Many components of a group editor, such as group interface features and the frequency of updates, must be able to adapt to changing conditions. In many cases, when editing occurs over a wireless channel, these features need to be scaled back to avoid delays caused by contention for a low-bandwidth link.

Significant reductions in traffic generation can be seen when certain interface features are disabled or when updates are made less frequently. These reductions allow a group editor to operate in a difficult, variable wireless environment without restricting the user interface unnecessarily. This research describes a number of application changes and the network conditions which should trigger a change to a different level of resource consumption. These changes can be generalized to other editors as well as to other type of groupware; developers of any application that balances the richness of a group interface and frequency of updates with efficient use of a limited channel can use many of these recommendations and results.

Another issue which arose from this research was that the design of groupware is not always appropriate if the application is to be used over a wireless channel. Details that seemed to cause little overhead in an application designed for use on a wired network can cause problems when the application is brought into the wireless environment. Changing these implementation details after the application is completed is a very difficult task, as assumptions about the underlying network are deeply ingrained in the whole structure of the program. This suggests that if an application is to be used in a wireless environment, the designer must ensure that communications over the network are as efficient as possible.

This thesis has also suggested an appropriate set of API variables that adaptive groupware would need to use. Remaining channel capacity and latency are very important variables, while many of the other network variables were not required. This research also suggested a list of variables to add to the API for better support of adaptive groupware, including signal quality information and an estimate of remaining channel capacity in the near future.

The recommendations suggested by this research lead to a number of areas of future research. One topic which requires further research is integrating the API with Calliope. Adaptation must be invoked by some method, but how the user interacts with this adaptation must be resolved. Possible methods include using configuration files, or suggesting recommended actions and allowing the user to perform these actions whenever certain conditions obtain. Also required is an investigation of how to reveal network condition information (such as latency) through the interface in order to support a more coherent conference session.

Because user interface changes can affect the ease of use of an application, the adaptive interface needs to be studied with users. Such an investigation should indicate whether some of the "restricted interface" changes make Calliope too difficult to use, or whether they improve performance without diminishing the usability of the application.

Several changes need to be made to Calliope in order to correct implementation details that lead to unnecessary traffic. It is conceivable that the extent of the changes would be so great that it would be preferable to build another similar editor completely rather than modify existing Calliope code.

Not all of the variables recommended for adaptive groupware are included in the API monitoring list. Network variables such as round trip time to other conference users' machines, disconnection probability, and remaining bandwidth estimate would prove very useful, but all require further research. Methods for obtaining these variables cannot be found easily, so it is not a trivial matter to extend the API to monitor them.

Further work is also needed to integrate the API with the WaveLAN MIB. There are a number of variables in the MIB which appear to be very valuable for use with adaptive applications, including several which deal with signal quality. When this new MIB is added to the API, investigations of their use (along with values that can be derived from this set) can be initiated.

Appendix A

API Variables

Here is the complete list of variables whose values can be monitored and returned to an application through the API, presented in two tables. These tables list the API variables selected from the SNMP MIB-II as well as the variables defined by our Mobile Computing Project which are derived from SNMP variables or collected by other means than an SNMP agent. This variable list is complete as of August 1, 1997. As the API is undergoing frequent changes, this list may not be correct after that date. The descriptions of the SNMP variables below are based on descriptions in [16]; consult this document, or [24], for further details.

SNMP Variable	Description
sysDescr	Textual description of entity, such as hardware, networking software, etc.
sysObjectID	Vendor's authoritative identification of network-management subsystem contained in the entity
sysUpTime	Time (in hundredths of a second) since network management portion of system was last re-initialized
sysName	Administratively-assigned name for this managed node (by convention, the node's domain name)
sysLocation	Physical location of the node
sysServices	Value indicating the set of services that this entity primarily offers

Table A.1: SNMP system group variables.

SNMP Variable	Description
ipInReceives	Total number of input datagrams received from interfaces, including those received in error
ipInHdrErrors	Number of input datagrams discarded due to errors in their IP headers
ipInAddrErrors	Number of input datagrams discarded because the IP address in destination field was not valid to be received at this entity
ipForwDatagrams	Number of input datagrams for which this entity was not final IP destination, causing an attempt to forward datagram
ipInUnknownProtos	Number of locally-addressed datagrams received successfully but discarded due to unknown or unsupported protocol
ipInDiscards	Number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g. for lack of buffer space)
ipInDelivers	Total number of input datagrams successfully delivered to IP user protocols
ipOutRequests	Total number of IP datagrams that local IP user protocols supplied to IP in requests for transmission
ipOutDiscards	Number of output IP datagrams for which no problems were encountered to prevent their transmission to their destination, but which were discarded (e.g. for lack of buffer space)
ipOutNoRoutes	Number of IP datagrams discarded because no route could be found to transmit them to their destination
ipRoutingDiscards	Number of routing entries which were chosen to be discarded even though valid (e.g. to free up buffer space)
udpInDatagrams	Total number of UDP datagrams delivered to UDP users
udpNoPorts	Total number of received UDP datagrams for which there was no application at destination port
udpInErrors	Number of received UDP datagrams that were undeliverable due to reasons other than lack of an application at destination port
udpOutDatagrams	Total number of UDP datagrams sent from this entity

Table A.2: SNMP IP and UDP group variables.

SNMP Variable	Description
ifNumber	Number of network interfaces present on system
ifIndex	Unique value for each interface
ifDescr	Information about interface: includes manufacturer, product name, version of hardware interface
ifType	Type of interface, distinguished according to physical/link protocol(s), e.g. fddi, ethernet-csmacd, etc.
ifMtu	Size of largest datagram, in octets, that can be sent/received on interface
ifSpeed	Estimate of interface's current bandwidth in bits per second. Despite description, this value is generally static, i.e. nominal bandwidth
ifInOctets	Total number of octets received on interface, including framing characters
ifInUcastPkts	Number of subnetwork-unicast packets delivered to higher-layer protocol
ifInNUcastPkts	Number of non-unicast packets delivered to a higher-layer protocol
ifInDiscards	Number of inbound packets discarded despite having no detectable errors that would make them undeliverable to higher-layer protocol (e.g. buffer overflow)
ifInErrors	Number of inbound packets discarded due to errors that would make them undeliverable to higher-layer protocol
ifInUnknownProtos	Number of packets discarded due because of unknown or unsupported protocol
ifOutOctets	Total number of octets transmitted on interface, including framing characters
ifOutUcastPkts	Total number of packets that higher-level protocols requested be transmitted to subnetwork-unicast address, including those discarded or not sent
ifOutNUcastPkts	Total number of packets that higher-level protocols requested be transmitted to a non-unicast address, including those discarded or not sent
ifOutDiscards	Number of outbound packets discarded despite having no detectable errors that would make them undeliverable to higher-layer protocol (e.g. buffer overflow
ifOutErrors	Number of outbound packets that could not be transmitted due to errors
ifOutQLen	Length, in packets, of output packet queue

Table A.3: SNMP interfaces group variables.

SNMP Variable	Description
tcpRtoAlgorithm	Algorithm used to determine timeout value for retransmission (e.g. Van Jacobsen)
tcpRtoMin	Minimum value for retransmission timer, in milliseconds
tcpRtoMax	Maximum value for retransmission timer, in milliseconds
tcpMaxConn	Limit on total number of TCP connections the entity can support
tcpActiveOpens	Number of active opens that have been supported by this entity
tcpPassiveOpens	Number of passive opens that have been supported by this entity
tcpAttemptFails	Number of failed connection attempts that have occurred at this entity
tcpEstabResets	Number of resets that have occurred at this entity
tcpCurrEstab	Number of TCP connections whose current state is either ESTABLISHED or CLOSE-WAIT
tcpInSegs	Total number of segments received, including those received in error
tcpOutSegs	Total number of segments sent, excluding those containing only retransmitted octets
tcpRetransSegs	Total number of segments retransmitted

Table A.4: SNMP TCP group variables.

Mobile Project Variable	Description
netLatency	Round trip time to default router
avgInIPPkts	Running average of incoming unicast and broadcast IP packets (SNMP-derived)
cpuLoadAvg1	Average length of run queue, as reported by kernel, over the last minute
cpuLoadAvg5	Average length of run queue, as reported by kernel, over the last 5 minutes
cpuLoadAvg15	Average length of run queue, as reported by kernel, over the last 15 minute
ethErrsAvg.5	Number of Ethernet frames received with errors over last 0.5 minutes (SNMP-derived)
ethErrsAvg1	Number of Ethernet frames received with errors over last minute (SNMP-derived)
ethErrsAvg5	Number of Ethernet frames received with errors over last 5 minutes (SNMP-derived)
ethInAvg.5	Number of Ethernet frames received by this machine, averaged over last 0.5 minutes (SNMP-derived)
ethInAvg1	Number of Ethernet frames received by this machine, averaged over last minute (SNMP-derived)
ethInAvg5	Number of Ethernet frames received by this machine, averaged over last 5 minutes (SNMP-derived)
ethOutAvg.5	Number of Ethernet frames sent by this machine, averaged over last 0.5 minutes (SNMP-derived)
ethOutAvg1	Number of Ethernet frames sent by this machine, averaged over last minute (SNMP-derived)
ethOutAvg5	Number of Ethernet frames sent by this machine, averaged over last 5 minutes (SNMP-derived)
deviceList	List of device names and numbers and their status (up or down)
bytes_rx	Size of incoming socket buffer, in bytes
bytes_tx	Size of outgoing socket buffer, in bytes

Table A.5: Variables defined by Mobile Group.

Appendix B

IPtrace Sample Output

Here is a sample of the output from iptrace, the program that was used to study the traffic generated by Calliope. The relevant fields are marked by numbers and described below.

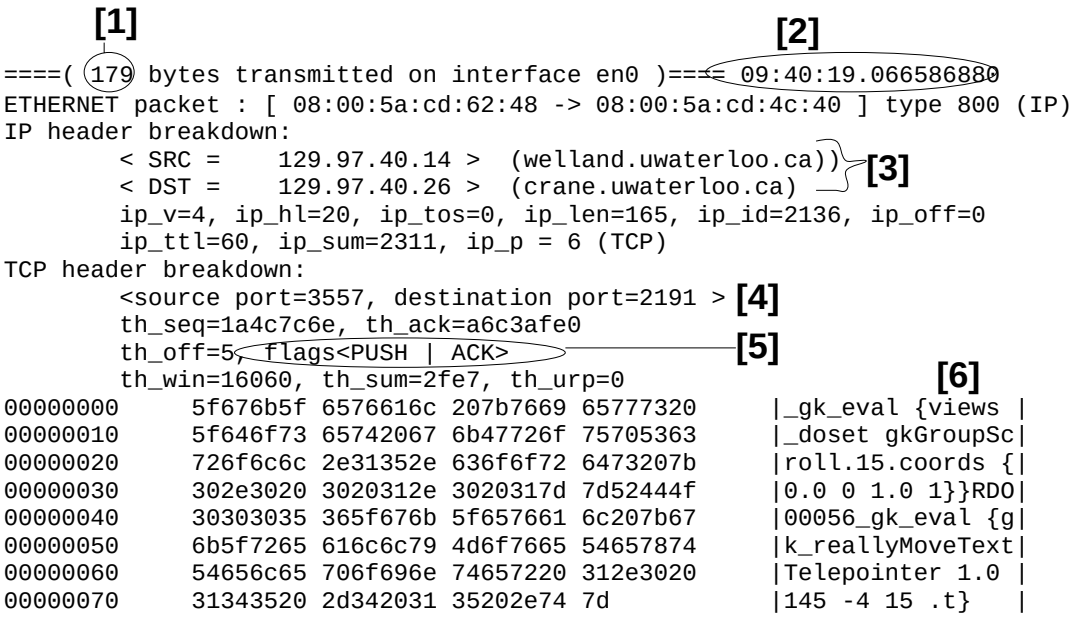

Figure B.1: One IP packet, as described by iptrace.

Relevant fields:

[1] The number of bytes in the packet including header information. This figure was needed to tell how much traffic was being sent over the network.

[2] The time when the packet was seen on the network interface. This was used to estimate the amount of traffic generated over a given duration.

[3] The source and destination machines, with both names and IP addresses. This was needed in order to trace the direction of communication, i.e., which machine was sending and which was receiving.

[4] The source and destination ports. These were used to filter out packets travelling between the two machines that were not generated by Calliope.

[5] The TCP flags. These were used to identify acknowledgments of data packets, by searching for an ACK flag.

[6] The packet contents, in ASCII. This information was used to track which action (such as a telepointer update) generated the packet.

Bibliography

[1] R. M. Baecker. *Readings in Groupware and Computer-Supported Cooperative Work*. Morgan Kaufman, San Mateo, CA, 1993.

[2] T. Brinck, R. Hill, and J. Patterson. F2 Tutorial Notes – Groupware for Realtime Collaboration. Cited in Rada, R., ed., *Groupware and Authoring*. Academic Press, London, 1996.

[3] N. Davies, G.S. Blair, K. Cheverst, and A. Friday. A collaborative multimedia application for a mobile environment. Technical Report MPG-94-14, Lancaster University, 1996.

[4] C.A. Ellis, S.J. Gibbs, and G.L Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, January 1991.

[5] T. Fallmyr. Adaptable mobile systems. In *Proceedings of the Fifth IEEE Workshop on Future Trends in Distributed Computing*, pages 363–368, Cheju Island, Korea, August 1995.

[6] G.H. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, April 1994.

[7] S. Golub. Software can be found at http://www.cs.wustl.edu/~seth/txt2html.

[8] S. Greenberg, C. Gutwin, and M. Roseman. Semantic telepointers for groupware. In *Proceedings of the OzCHI '96 Sixth Australian Conference on Computer-Human Interaction*, Hamilton, New Zealand, November 1996. Available at http://www.cpsc.ucalgary.ca/projects/grouplab/papers.

[9] S. Greenberg and M. Roseman. Groupware toolkits for synchronous work. Technical Report 96/589/09, University of Calgary, Department of Computer Science, 1996.

[10] L. Honeycutt. Electronic correspondence with author, March 17, 1997. Honeycutt is maintainer of the Online Bibliography of Computer-Supported Collaborative Writing.

[11] T. Imielinski and B.R. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37(10):19–27, October 1994.

[12] M. Knister and A. Prakash. Issues in the design of a toolkit for supporting multiple group editors. *Computing Systems – The Journal of the Usenix Association*, 6(2):135–166, 1993.

[13] M. Koch. The collaborative multi-user editor project IRIS. Technical Report TUM-I9524, Technische Universität München, August 1995.

[14] M. Koch. Design issues and model for a distributed multi-user editor. *Computer Supported Cooperative Work – An International Journal*, 3(3–4):359–378, 1996.

[15] H. Y. Lo. M-mail: A case study of dynamic application partitioning in mobile computing. Master's thesis, Department of Computer Science, University of Waterloo, 1997.

[16] K. McCloghrie and M. Rose. RFC1213: Management information base for network management of TCP/IP-based internets: MIB-II. Network Working Group, March 1991.

[17] A. Michailidis and R. Rada. A review of collaborative authoring tools. In R. Rada, editor, *Groupware and Authoring*. Academic Press, London, 1996.

[18] A. Mitchell. Communication and shared understanding in collaborative writing. Master's thesis, Department of Computer Science, University of Toronto, 1996.

[19] M. Nidd, T. Kunz, and J.P. Black. Wireless applications and API design. In *Proceedings of the 4th International IFIP Workshop on Quality of Service*, pages 83–85, Paris, March 1996.

[20] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Massachusetts, 1994.

[21] M. Roseman and S. Greenberg. Building real-time groupware with Groupkit, a groupware toolkit. *ACM TOCHI Transactions on Computer Human Interaction*, 3(1):83–85, March 1996.

[22] M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1):26–33, February 1996.

[23] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Reading, Massachusetts, 1987.

[24] W. Stallings. *SNMP, SNMPv2, and CMIP: the practical guide to network management*. Addison-Wesley, Reading, Massachusetts, 1993.

[25] AT & T. SNMP MIB for AT & T WaveLAN, Version 2.0, May 1995. Available at http://www.networks.digital.com/dr/wireless/mibs.

[26] T. Watson. Application design for wireless computing. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 91–94, Santa Cruz, CA, December 1994.