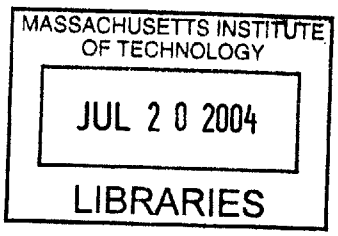
Clique Here: Exploring the Use of Mobile Multi-media to Support Connectedness

by

Joseph Edmund Corral

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2004



Author .....
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by .....
Christopher Schmandt
Principal Research Scientist, MIT Media Laboratory
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Clique Here: Exploring the Use of Mobile Multi-media to Support Connectedness

by

Joseph Edmund Corral

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The practical demands of modern life obligate people to spend time physically seperated from the people they care about. To cope, people turn to communications technology like the telephone, e-mail, and instant messaging (IM) to maintain a connection with social contacts. These communication modalities are limited in their ability to provide social-connectedness by their failure to balance expressiveness, overhead, and social obligation. Clique Here is a mobile communications platform that attempts to address this limitation and support a higher degree of social-connectedness by complementing mobile telephone capability with media rich awareness and multiple lightweight communication modes. The Clique Here system consists of a mobile client, implemented on a camera embedded mobile handset, a home client, implemented on a wireless web tablet, and communication between clients is facillitated by an application server.

Thesis Supervisor: Christopher Schmandt
Title: Principal Research Scientist, MIT Media Laboratory

Acknowledgments

It's time to take a breath.

One random afternoon when I was less than ten years old, my father had the MIT 2.007 design competition on TV, and I sat down to watch it with him. This competition happens every year and the participants are MIT mechanical engineering students that design robots to compete against each other in performing certain tasks. After the show, I looked up at my father and told him I wanted to go to MIT. Today, I can only smile at the thought of how clueless that young kid was about the effort it would take to fulfill the goal he set so casually. I could not have known how important that goal would become in my life, or how far it would stretch me. Regardless, I set it, and I'm done. I could not have done it without the help of many great people in my life. I extend my most sincere thanks to:

First and foremost, my family for their unyielding willingness to encourage and assist me while I spent five years away from them at MIT. It is impossible for me to describe all of the ways my Mom and Dad have helped to get me to where I am. Every day, I become more aware of the effort both of my parents have made to see me happy. Perhaps the greatest gift they have given to me has been a constant sense of what is most important in life, which was especially important during the chaos at MIT. My sisters, Stephanie and Jessica, for always calling me to check in on my life in Boston, for their unfailing love and encouragement, and every other act of kindness that was often taken for granted. My two newest family members, my new brothers-in-law Chad and DaWuan, for taking care of the best two sisters in the world while I was away.

Chis Schmandt, my advisor, for opening the door to the Media Lab for me, for pushing me until we found a thesis that resonated with both of us, and for his enduring effort to help me succeed. Given the chance to do so again, I would not hesitate to make the same choice in selecting an advisor.

Natalia Marmasse, a SIG Ph.D. student, for sharing her vision of "collaborative living" technology that formed the foundation of my research work, for her willingness to lend a hand, even when she was facing the stress of her own deadlines, and for broadening my cultural literacy with stories of life in Israel. This thesis would not have happened without Natalia. It makes sense that she is not called a sabra, since she is sweet both on the inside and out.

Jang Kim, a former SIG student, for helping me to get my foot in the door at the Media Lab, for always looking to help me with my current challenges with wisdom gained from his past experiences, and for being an inspiration for integrity.

Fellow SIG students Assaf Feldman, Zeynep Inanoglu, Stefan Marti, Matt Hoff-

mann, and even our brief visitor, David Spectre, for the quality time spent at Friday tradition, for brainstorming sessions and technical knowledge, and for making the lab a fun place to work. I would especially like to thank Assaf taking the time to bounce technical and conceptual questions back and forth, for building up my knowledge of C, and for waking me up from the floor of my office in the morning.

My co-workers at HP, for reminding me that this knowledge will have to be applied one day. Especially Rich Suyehira, Rachelle Rowland, and Brent Phillips for opening doors, for their inspiring leadership, and for supporting success in both my professional and academic life. Also, Sue Black, Don Dayley, and Joe Reed, for being great teammates in every regard, and for making offsites interesting.

Finally, I would like to thank my close friends from home: Fred, Eric, and BJ, as well as my close friends from Boston: Kurtis, Jorgy, Karnal, and my crazy 5-year roommate, Brent. For reminding me to have a social life, for keeping me humble, and for all the great college memories. I'll never forget the great times had when camping at the spot, or hanging out at the woods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile communication continues to proliferate and expand in capability. Far-reaching cellular voice networks provide the potential for people to make themselves available for phone calls with any person, at any time. Domestic mobile data networks have become more practical in coverage and bandwidth, fostering improvement in offerings that seek to bring the successful communication modalities of the fixed Internet (e-mail, the World Wide Web, instant messaging) to the mobile domain.

The advance of mobile hardware has kept pace with that of mobile infrastructure. Modern handsets ship with high-resolution color displays, processing power on par with lower-end PDAs, polyphonic sound, and most notably, an increasing number of these devices contain integrated digital cameras. InfoTrends Research Group reports that worldwide sales of camera embedded mobile phones will reach 150 Million units in 2004, 25% of the total mobile phone market [7]. While the telecommunications industry has been in the business of connecting people for nearly a century, the rich mobile media, high bandwidth, and ubiquitous connectivity available to today's mobile phones makes it possible for these devices to connect us with the people we care about to an unprecedented degree.

In the past, mobile communications research focused primarily on the use of mobile devices in the professional setting [8, 18, 21], rather than the use of these devices to support social relationships. While the results of many such contributions do further the understanding of mobile communications in the social context, extracting the full

potential mobile devices have for supporting social relationships calls for research to focus on the unique requirements of this context. A research segment that heeds this call is gaining in popularity as a number of recent projects have explored mobile mediated social communication [6, 22, 3, 14, 13]. These projects, focusing on the mobile element of social communication, complement and coexist with work in a more general, burgeoning category addressing social communication mediated by any form of computing technology [12, 16, 4, 9].

Maintaining healthy social relationships leads to the satisfaction of basic human needs. Further, the health of a social relationship between two people depends on the degree of social-connectedness they feel toward each other [12], where social-connectedness is defined here as a feeling of inclusion within a contact's social life. The first of two primary factors that promote social-connectedness is awareness. Awareness, in this context, describes the knowledge a person has, gained by observation, about the state of a social contact's life (e.g. activity, availability, mood). When current awareness information is available to a person's contacts, the general knowledge contacts have of this person's life may allow them to make inferences and expand on the observed awareness. This up-to-date knowledge of a person's life will lead social contacts to feel in-touch with this person, and to feel included in his life. The second primary factor promoting social-connectedness is communication. When two social contacts communicate, they are being directly included in each other's life. The resulting feeling of social-connectedness depends on the amount of practical and emotional information shared, which is dependent on the expressiveness of the communication channel.

When two people share the same physical space, a high degree of social-connectedness is easily achieved. All five senses can be utilized for observation, leading to precise awareness. Communication is more probable since it is easy to identify appropriate moments for communication, communication can be initiated and completed quickly, and real-time aural and visual interactions provide channels for communication unmatched in expressiveness.

The practical demands of modern life, however, require the closest of social group

members to spend a substantial amount of time physically separated. For any level of social connectedness to exist, communication will need to be mediated by a channel that transcends physical space. Beyond the spatial requirement, systems mediating social communication must employ media types expressive enough to convey the affective nature of this form of communication, but they must also present the recipient with this content with minimal disruption and without introducing social obligation. Conventional communication systems fail to balance these two requirements. Synchronous phone calls can certainly be expressive, but the overhead for the sender and the intrusiveness imposed on the recipient become barriers to use for communicating simple affective thoughts such as "I love you" or "I was thinking about you" [4].

In their insightful paper on the expected and unexpected uses of text instant messaging (IM) observed in practice [17], Nardi et.al. showed that the lightweight nature of IM makes communicating simple affective thoughts reasonable, and the ambiguity of message reception creates plausible deniability, shielding the recipient from social obligation and giving her better control over incoming communication. Text IM users do manage to achieve a notable level of expressiveness through evolving conventions [6]. However, the potential for expressiveness of the rich media available for capture with modern mobile devices, specifically photographs and audio recordings, far exceeds text.

While lightweight communication modes are crucial for effectively mediating social communication, powerful non-verbal communications that are possible when the same space is shared, such as a *smile* or a *wave*, are not effectively mediated by traditional lightweight communication. Marmasse et.al. [14] have demonstrated how mobile communications systems can mediate such non-verbal interactions through their work on the WatchMe project. WatchMe is a personal communicator that combines sophisticated mobile awareness capability with lightweight modes of communication. In the WatchMe system, when an inquiring user observes the details of another user's availability information, this indicates that the inquiring user is currently thinking about the observed user. If the two users shared the same space, this "thinking of you" message could be conveyed through a smile. The WatchMe system replicates

this interaction by temporarily displaying a picture of the inquiring user on the screen of the observed user, a communication channel referred to simply as a *smile*.

Clique Here is a mobile messaging system supporting a higher degree of social connectedness by complimenting mobile telephone capability with media rich awareness and several lightweight communication modes. When direct communication is not possible, users can feel included in the lives of their contacts by browsing the images and audio clips their contacts have captured for the purpose of conveying awareness. The interest of the user browsing this awareness information is automatically conveyed to the observed user with a Marmasse *smile*. This rich awareness might entice direct communication, made possible among the constraints of professional life by lightweight communication modes (text IM, voice IM, image messaging). Finally, any combination of these interactions may serve as an evolutionary path to an opportunistic phone call between two social contacts. Clique here makes communication between physically separated social contacts more probable, more expressive, and more welcome.

1.1 A Clique Here Scenario

As the members of contemporary family units age, the dynamics of their interpersonal communication evolve as members make transitions into the different stages of their lives. A son leaving home to attend college at a distant university is an example of a transition that results in a significant shift in family dynamics. This will likely be the first time that the child will spend several months physically removed from the home. The following scenario illustrates how Clique Here could enable such a family to maintain social connectedness, despite the physical separation.

Susan, a mother of three children, has found herself emotional over the recent departure of her son, Josh, to an out of state university. Josh is not the first child to leave the home, but it still does not come easy, and she is adjusting to the absence of the 'little things' that she could enjoy when Josh lived in the house. Josh was a busy person in high school, but many times they could have a quick chat as he packed up

to head to practice or work, or maybe there would just be time for a hug, and he'd be on his way. Once these few minutes in which they could interact as they crossed paths during a busy day were gone, it was clear how valuable they had been. Even more valuable were the times when they both happened to have some free time, and they could take the opportunity to catch up on each other's life.

At school, Josh has been having the typical experiences of any college freshman. He has met a lot of new people, he's found that he has to spend a lot more time doing school work than he did in high school, and he's found that he can enjoy himself a bit more now that he's in charge of his own supervision. Even with all of these new experiences to keep him occupied, Josh still misses his family back home.

On a Saturday, Josh and some friends manage to get their hands on bleacher seats at the afternoon baseball game. As the guys find their seats, and they begin to absorb the unique energy that comes from sitting in a major league ball park, Josh remembers he needs to let some people know where he's at, and he won't mind letting it be known that he scored tickets to the game. So, he pulls out his mobile phone and snaps a picture of the field with the embedded camera and, by default, he sends the image to all of his contacts, all with just a couple of clicks.

Back at home, Susan is sitting with her husband, Jim, and their conversation reminds them of Josh. When their older daughter went away to school, Susan and Jim were also often reminded of her, but they resisted the urge to call her, knowing she might be busy, and instead limited their communication to their weekly Sunday phone call. Today, these parents no longer feel this limit. As Susan and Jim are reminded of their son, Susan pulls out her mobile phone. She scans her contact list, one of which who is Josh, of course, and sees that he has recently sent a new picture. When zooming in on the picture, the parents realize that Josh is at the game. Susan wants to call Josh to hear about it, but Jim tells her to let him enjoy the game, since it's about to start. Before the two can finish the debate, they get a call from Josh. On his own mobile phone, Josh was informed that his mother took the time to check out his image message, and he decided to give them a call in the five minutes before the game got started.

After the fourth inning, a friend of Josh's has seen his picture message and calls him to check on the score, express some jealousy, and to let him know that he is heading to the local bars to meet up with some people for a drink. After the game, Josh looks at his contact list, and sees that his friend has sent an image showing the bar the group has decided on, and knowing it is probably too loud in the bar for a phone call anyway, Josh and his friends head directly to the bar to meet up with the group.

Clique Here allowed the characters of this scenario to maintain social-connectedness, despite physical separation. The mother did not have to wait until the Sunday phone call to hear about the baseball game. Instead, she was made aware of it in real time, with little effort from her son. Even more important, since the son was also aware that his mother was thinking of him, by her looking at his image message, it led to an opportunistic interaction.

Chapter 2

User Interaction

This chapter describes the user interaction experience of the Clique Here system. The system has been designed with the goal of supporting a high degree of social connectedness by leveraging the expressiveness of mobile multi-media, by mapping the successful principles of IM to the constraints of the mobile domain, and by introducing novel non-verbal channels of communication. Clique Here optimizes for the *control* of availability information due to the lack of automatic sensors on the mobile phone that could provide this information, and to mitigate privacy related issues. According to the *overhead vs. control* scale characterized by Milewski, the cost of giving users the control to set awareness information is that the overhead of maintaining the information is transfered to the user. While users have a social incentive to utilize the expressiveness of the multi-media awareness channels to share the interesting events in their life, the user-interface must be designed to minimize the input burden if the users are to be expected to act on these incentives and keep availability information current.

The system contains two client devices, one designed for the mobile user, implemented on a mobile phone, and one designed for users in the home, implemented on a wireless web browsing device. Communication between all clients is mediated by a server. The mobile client contains the bulk of the features supporting the goal of maintaining social-connectedness. The home client focuses on providing the home users a social-awareness of their contacts outside of the home. Each client was designed

with a particular user segment in mind. While considerable effort and iteration was devoted to maximizing the usability of the mobile interface, it was decided that the technology barriers may still be too great for user segments containing middle aged parents, or even grandparents. The home client provides such users with a simple way to view image messages from their loved ones, allowing them to infer availability from a rich medium, and if desired initiate a synchronous interaction via the home phone.

The following sections discuss the details of Clique Here user interaction by first describing the communication channels available through the system, and going on to describe how each client supports these channels and optimizes their ability to maintain social-connectedness.

2.1 Communication Channels

Clique Here offers several communication channels with varying levels of attentional demand, overhead in establishment, and expressiveness. These options give users the flexibility to match the appropriate communication channel with contact availability, inferred using Clique Here awareness information. These options also allow the user to make incremental steps toward more heavyweight communication channels.

2.1.1 The Phone Call

The mobile Clique Here client is implemented on a mobile handset, and the system acknowledges the fact that the venerable phone call remains an important channel for the purposes of task-oriented-communication and social-communication alike. The synchronicity of the full-duplex phone call, along with the expressiveness of audio, results in information exchange capabilities that approach real-time, face-to-face verbal interactions. The attentional demands of the phone call for both parties, however, along with the overhead of establishing the communication, make it important for people to effectively manage their phone communications. The importance of managing communication over the phone is further emphasized as the pervasiveness of

mobile phone service makes it possible for people to be reached through this channel at nearly every moment of their day. Clique Here allows people to better manage telephone engagements with their social contacts by providing availability information, which may enable smarter choices on when and how to initiate communication. In addition, Clique Here provides alternate communication channels, which may be just as effective as the phone call in achieving certain communication goals, but might be less burdensom to the parties involved.

2.1.2 Asynchronous Multimedia

Clique Here augments the traditional capabilities of instant messaging by including image and audio media as new data types. Adding such media to the lightweight communication framework of IM affords the use of this media as a means of communication. Clique Here users can take advantage of the minimal input burden of capturing image and audio media, and avoid the text input limitations of mobile devices[1], to send lightweight, expressive messages to their contacts. In addition, because they can send pictures, voice, and/or text, senders can tailor their messages to best capture the content they wish to communicate, and choose a form that is most appropriate in terms of the recipient's availability.

2.1.3 Non-Verbal Channels

A Clique Here user may browse a contact's awareness and availability information for several reasons. He may wish to know if the contact is available for communication, and if so what might be the most appropriate channel. Alternatively, he may just wish to know what this person is up to. People sharing the same space make similar queries to maintain social-awareness and determine availability, with similar motivations.

When the same space is shared, however, these non-verbal queries can be interac-

[1]The most prevalent text input method on mobile phones is multi-tap, a method where each numeral key maps to the letters of the alphabet printed on the keys. Tapping the numeral once will input the first letter on the key. Tapping the numeral twice quickly will enter the second letter, and so on.

tive. Natalia Marmasse's [14] scenario of the exchange of smiles between two people sitting on a couch is a perfect example. One person may glance at the other out of curiosity, to determine their mood or to determine their availability. The other person may notice, look up, and smile at the other person, or perhaps never notice it at all. If the person that has been glanced at should look up and smile, the other might choose, based on his inference of non-verbal cues, to smile back, and go about his business, or strike up a casual conversation.

Clique Here provides two non-verbal communication channels, *smiles* and *waves*, to replicate non-verbal interactions that naturally take place when the same physical space is shared. When a user uses either the home or the mobile client to observe the details of a contact's awareness information, this generates a *smile*. The observed contact is notified of this event on his mobile client by a temporary pop-up displaying the latest image received from the inquiring user. *Waves* are created by a Clique Here user when he wishes to directly query a contact to determine if he is available for communication, and do so in as lightweight of a form as possible. Contacts receiving wave events are also notified by a temporary pop-up screen that displays the latest image received from the waving user, but the screen does not timeout and must be manually dismissed. This image displayed in the pop-up may provide the wave recipient with an intuition of the sender's current situation, and help him to better decide if he wants to engage in communication. Sections 2.2.1 and 2.2.4 describe the appearance of both *smiles* and *waves* within Clique Here screens.

2.2 Mobile Client

The most valuable aspect of Clique Here is the mobile client. A Nokia 6600 smartphone was selected as the hardware platform for the mobile client (Figure 2-1). With no keyboard input and an above average, but still cramped 176X208 pixel display, the 6600 imposes the same user interface design constraints as most mobile phones. The handset has an embedded VGA resolution digital camera and captured images can be clearly displayed on the 16-bit color display. The 6600 also features audio

Figure 2-1: The Nokia 6600 displaying the Clique Here *Contact List Screen.*

capture and playback functionality, utilizing the embedded microphone and speaker. The Nokia 6600 is a Symbian Series 60 device and it provides open APIs for native Symbian in C++ and the Java MIDP 2.0 platform. The MIDP 2.0 platform was chosen for development because its `javax.microedition.lcdui` toolkit provided the best balance between efficiency, flexibility, and extensibility. An important addition to MIDP 2.0 from its predecessor, MIDP 1.0, is the `CustomItem` class. This abstract class allows items with custom painting and behavior requirements to be added to `Form`s, and in doing so combines the efficiency of the high-level `Form` API with the flexible low-level `Canvas` constructs. The MIDP 2.0 high-level (form based) UI API supports both traversal based selection and pen based selection for mobiles with pointing devices. However, the Nokia 6600 has no such device; instead, the five-way navigation button allows users to traverse and select the standard GUI components (e.g. text boxes, buttons, lists) that lie on a given form. This selection limitation makes button-based commands less than ideal, since a user would have to navigate to the button in a serial fashion before he could invoke a command, and the issue becomes worse as the size of the form increases. For this reason, Clique Here command and control is accomplished through the soft-key menu interface provided by the 6600. Like many MIDP devices with soft-keys, the Nokia 6600, incorporates the

program specified `Command` objects, which encapsulate the actions available to the current screen or selected item in the program, into their soft-key menu interface. The Nokia 6600 soft-key implementation, fortunately, was done relatively well. The 5-way 6600 joystick enables quick selection of soft-key menu items, and if a command is associated with an item that has been selected, pressing the "fire" button displays a pop-up menu listing those options associated with this particular item.

The following sections describe the mobile client user interface each screen at a time, along with descriptions of the relationships between the screens.

2.2.1 Contact List Screen

Once successfully logged onto the Clique Here application, users are presented with the *Contact List Screen*. Figure 2-2 shows the *Contact List Screen* on the Nokia 6600. The structure of this screen is very similar to the *buddy-list* concept seen in traditional IM systems. Each user has a unique *roster* of contacts on the Clique Here system. For each contact on the user's roster, the *Contact List Screen* contains a *Contact List Item*, a selectable area displaying the contact's username, along with several icons and visual variables, each of which will be discussed in this section. The general purpose of the *Contact List Screen* is to serve as the entry point for communication with each contact, and provide a breadth of contact awareness and availability information across the roster.

Clique Here incorporates the awareness and availability information offered by traditional IM systems, e.g. online/offline status and idle/active state. Clique Here defines a user as offline when they are not logged into any client application; if a logged in user has temporarily lost wireless service, they are not considered to be offline. If a contact is on a user's roster, but currently offline, such as the Joseph contact in Figure 2-2, the background of the contact's *Contact List Item* becomes gray and the contact username is displayed in plain black italic font. In contrast, the *Contact List Item* background is drawn with a light color when the user is online, and their username is displayed in bold black font if the user is active, and gray bold otherwise. The Assaf, Natalia, and Chris contacts in Figure 2-2 are examples of

Figure 2-2: The *Contact List Screen* (on left) provides availability information. The options menu for this screen is on the right.

online contacts.

Unlike traditional IM systems, Clique Here users can send messages to their contacts, even if they are offline. Upon logging onto the system, the user can quickly scan each *Contact List Item* for the *envelope* icon, which indicates that a new message has been received from this contact. To the right of the icon, a number indicates how many new messages have been received, and the contact who sent the original message need not be currently online in order for the user to retrieve messages that have been sent. In its example view of the *Contact List Screen*, Figure 2-2 provides an example of the envelope icon's appearance in the interface.

When browsing the *Contact List Item* of a particular contact, the usernames of online contacts are proceeded by an icon indicating the type of client they are logged on to. If a user is logged in from a Clique Here mobile client, such as the Natalia contact from Figure 2-2, a small mobile phone icon appears. If a user is logged in from a Clique Here home client, such as the Chris contact from Figure 2-2, a small house icon appears. Knowing this single bit of information, along with knowledge of the daily habits of close contacts, could allow users to make important inferences

about availability.

The most visible component of each *Contact List Item* is a thumbnail image, appearing at the far left. This thumbnail displays the most recent image sent to the user as part of a Clique Here message. This thumbnail is updated in real time as new messages arrive to the user from each contact, which means that the user may be able to infer current availability based on the contents of the picture. The *newness* of the thumbnail is communicated through the saturation level of the thumbnail border. When the image is brand new, the border is a fully saturated blue, and the saturation decreases linearly in time, leaving the border color gray after 1 hour. The thumbnail image displayed for the Natalia contact in Figure 2-2 is quite new since the image is bordered with an almost completely saturated blue.

From the *Contact List Screen*, the user can navigate to the *Message Detail Screen* to observe a more detailed view of a contact's availability. The navigation to this screen by a user, will create one of the *smile* events described in Section 2.1.3. The smile is communicated to the contact that has been observed through a *Picture Alert Screen* pop-up appearing on his mobile client display. After this screen is dismissed, or after it times out on its own, the observed contact can determine that he has been smiled at from his *Contact List Screen*, as a smiley face icon appears on the far right of the *Contact List Item* associated with the smiling contact. Adjacent to the icon is an integer that reflects the number of minutes that have passed since the smile took place. The smile icon disappears after 15 minutes. The smile icon is kept visible for 15 minutes to make the user aware of the smile within a time frame when it might be useful as a means to infer availability, augmenting other availability information provided, or it may provide value simply as an indication that the sender was thinking of them.

As described in Section 2.1.3, Clique Here allows the user to *wave* to a contact, and in contrast to the smile, the wave is manually initiated by the sender. A wave, by convention, has a stronger connotation in terms of the sender's desire for verbal communication. Acting as a lightweight preamble to verbal communication, the wave is an unintrusive way to deal with any ambiguity left after making inferences based

on availability information. When a user has been waved at, just as with the smile, the user is notified via a *Picture Alert Screen* pop-up appearing on their mobile client. However, in the case of the wave, this screen does not time out; it appears until the user dismisses it.

From the *Contact List Screen*, users can navigate to other screens and access additional Clique Here functionality using the options menu, shown if Figure 2-2. The "Roster message" option will bring up the *Roster Message Screen*, allowing the user to send a Clique Here message to everyone on his roster. The remainder of the commands on the option menu are associated with the *Contact List Item* currently selected. Selecting "View details" will open the *Message Detail Screen*, which will display the full details of the selected contact's thumbnailed message. If the user wishes to exchange Clique Here messages with a selected contact, invoking the "Chat with user" command will cause the *Chat Screen* to appear. These screens, along with screens that support their intended purpose (e.g. *Audio Capture Screen*, *Image Capture Screen*), will be described in full detail in the following sections.

2.2.2 Image Capture Screen

An important goal of Clique Here is to enable images captured on a camera phone to be used as a means for communication. To this end, the *Image Capture Screen* was designed to capture images in as few clicks as possible. When the screen first appears, the viewfinder video feed is displayed in the center of the screen, surrounded by a red border (Figure 2-3). The user can capture the image shown in the viewfinder at any moment simply by pressing the *fire* button, or they can select the "Capture" command from the options menu. The captured image will then be shown in a preview mode, shown in Figure 2-4, but some latency exists between the point when the capture command is invoked and when the image preview appears. This latency is masked with an animated progress indicator.

In preview mode, the captured image is shown in the center of the screen. If the user is dissatisfied with the image, he may select the "Re-take" command, and return to capture mode. Otherwise, the user can choose to accept the image by pressing the

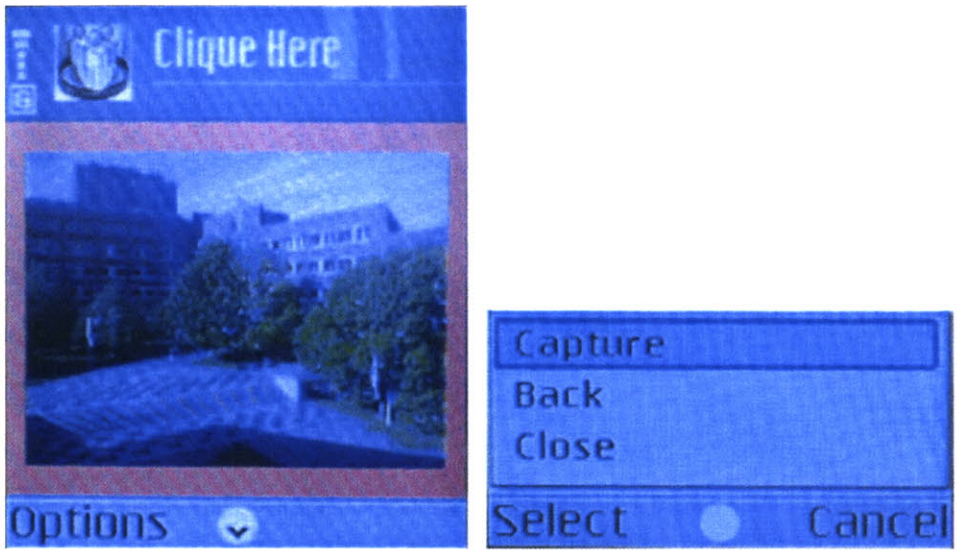


Figure 2-3: Capture mode of the Image Capture Screen displays the video viewfinder on the majority of the available screen.

fire button or selecting the "Accept" command from the options menu. Invoking the "Accept" command will take the user back to either the *Roster Message Screen* or *Chat Screen*, whichever was navigated from to create this image. At that point, the image will be added to a Clique Here message which is being composed. Alternatively, if the user wishes to accept the photo and automatically send the CliqueHere message being composed, they may select the "Accept & Send" command from the options menu, and the image will become part of a Clique Here message.

2.2.3 Audio Capture Screen

Analogous to the image capture requirements, it was crucial to ensure users could capture audio media as simply as possible. The *Audio Capture Screen* went through significant iteration in order to meet this requirement. The earliest versions relied on GUI drawn buttons to invoke commands. However, with the 6600 only supporting traversal based selection, this approach had serious usability issues.

The next iteration moved the commands to the options menu, and the user would select the "record" command from the menu to begin audio capture. Capture would

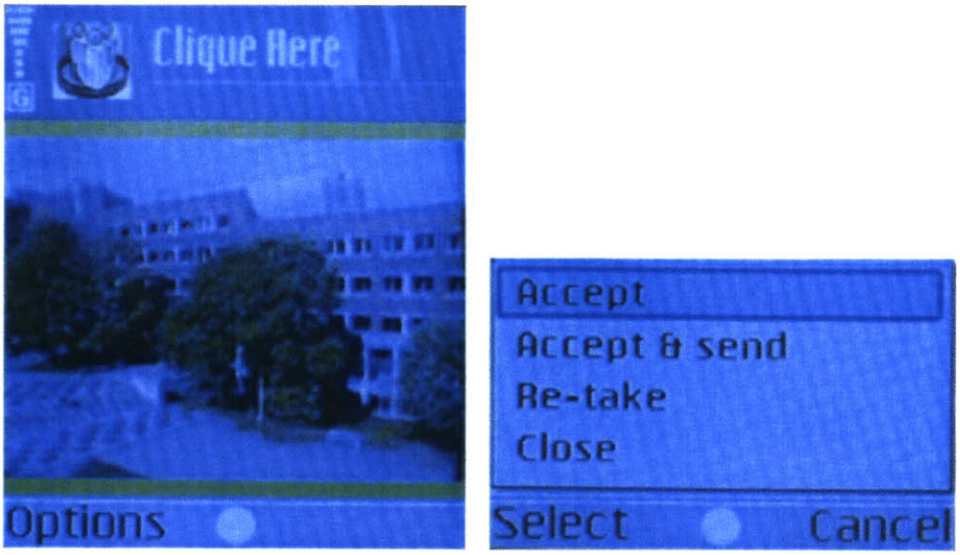


Figure 2-4: Preview mode of the Image Capture Screen displays the image that has been captured, giving the user the option to accept the picture or re-take it.

end when the user selected the "stop" command from the menu, or after a specified maximum time. The user could then review the message by selecting "play", and send the message by selecting "send". This approach took too much time for the user to end the audio capture, since she had to both open the options menu, and select "stop". Also, allowing the user to replay the message after it was recorded added an extra step with marginal value. The common use of voice-mail shows that people are used to recording audio and sending it without review.

The current iteration simplifies the process further. To begin recording audio, users press, and hold the *fire* button. Recording is stopped by releasing the *fire* button, or automatically if a maximum duration is reached. Once capture completes, the clip is automatically accepted, and as in the *Image Capture Screen*, the clip is added to the Clique Here message being composed from either the *Roster Message Screen*, or the *Chat Screen*. At these screens, the user has the option to review the audio message, and discard it if they wish. This means that if they still want an audio clip included in the Clique Here message, then they will have to re-navigate to the *Audio Capture Screen*. Note that the differences in how audio and photos are

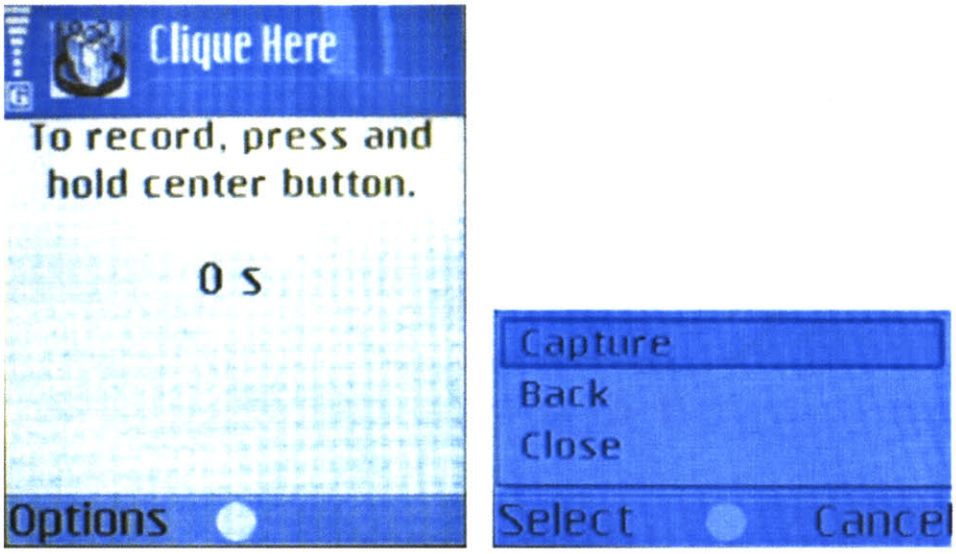


Figure 2-5: The Audio Capture Screen was designed for speed and simplicity. Users start recording audio by pressing and holding the *fire* button, and release it when they are finished.

accepted reflects the observations that users are much more likely to get an audio recording right the first time.

2.2.4 Picture Alert Screen

When a user receives a smile, wave, or a Clique Here message (sent only to him), he is alerted via the *Picture Alert Screen*, shown in Figure 2-6. The behavior of this screen is slightly different for each of these three events, but the common feature shared by each is that the screen pops-up and displays a near screen sized image, leaving room for additional information above it. The image displayed is the most recent image received from the user that has sent the smile, wave, or Clique Here message. Above the image, a string reads "<username> smiles(or waves)". For a smile, the alert dismisses itself after 5 seconds, but a wave causes the *Picture Alert Screen* to remain visible until the user manually dismisses it. When receiving a Clique Here message from a contact, the *Picture Alert Screen* displays icons indicating if the message contains an image, audio clip, or text. These icons are followed by the string

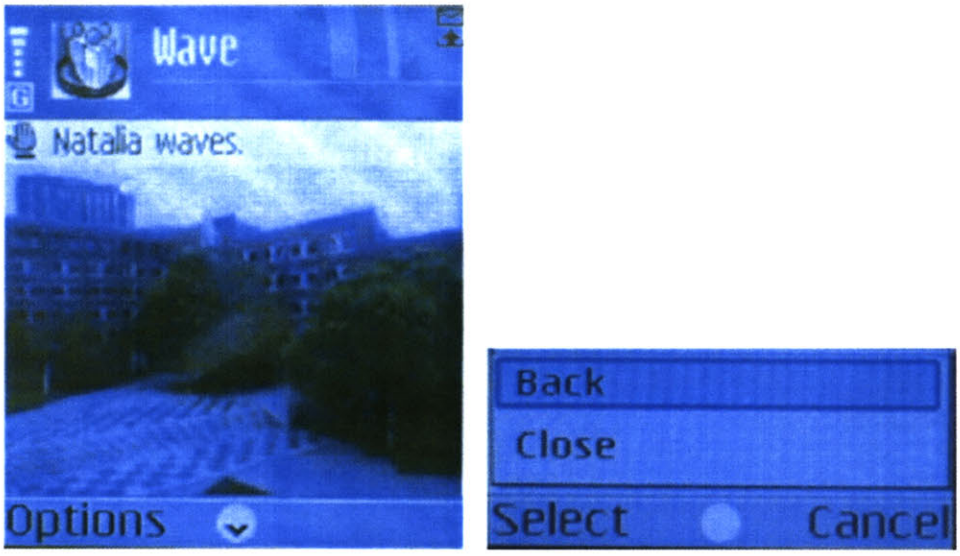


Figure 2-6: The Picture Alert Screen notifies the user of incoming *smiles*, *waves*, and *Clique Here messages* (if sent directly to the user).

"Msg from: <username>".

2.2.5 Chat Screen

Users wishing to communicate one-on-one with a contact may navigate to the *Chat Screen* from the *Contact List Screen* by selecting the "Chat with user" command from the options menu. The *Chat Screen*, shown in Figure 2-7, serves two primary functions: first, to display the history of Clique Here messages sent to and received from this contact, and second, to provide an interface for composing new Clique Here messages to be sent to this contact. Message history information appears at the bottom of the screen. Messages are sorted chronologically with the most recent at the top. Each message entry features the username of the message sender, written in red font if the sender was the local user, and blue if the contact sent the message. A timestamp follows the sender username, indicating the time the message was sent. On the lines below, a thumbnail image appears if the message included an image attachment, a black loudspeaker icon appears if the message included an audio clip attachment, and any text that was included with the message appears.

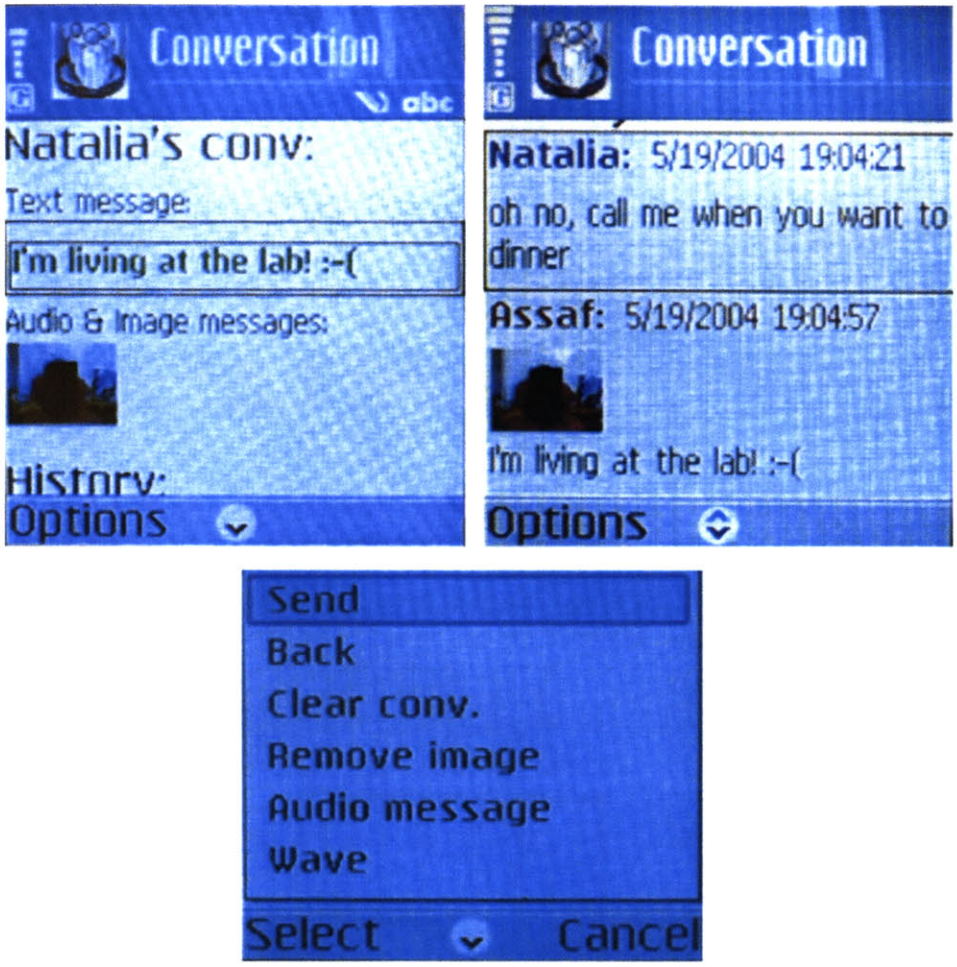


Figure 2-7: The Chat Screen is similar to a traditional fixed-Internet IM chat window, but it has been optimized for the limited mobile device display and augmented to support multi-media messaging.

Users browsing the Clique Here message list can view the image attachment, if one exists, at full screen and listen to the audio clip, if one exists, by selecting the "View details" command from the options menu. This takes the user to the *Message Detail Screen.* Also, the entire message history can be cleared by the user by selecting the "Clear conv." command from the options menu.

The top portion of the screen is dedicated to message composition. A standard text box appears, allowing the user to enter a text message. Multi-media components may be added to the message by invoking either the "Image message" or "Audio

message" commands from the options menu. These commands bring up the media capture screens described in the sections above. As long as at least one media type (text, image, or audio) is added to the outgoing message, it may be sent via the "Send" command.

2.2.6 Message Detail Screen

Abbreviated representations of Clique Here messages appear both in the *Contact List Item* of the *Contact List Screen* and in the message history list of the *Chat Screen.* Selecting the "View details" command while either of these abbreviated forms are selected brings up the *Message Detail Screen*, shown in Figure 2-8. This screen allows the user to view the full detail of a Clique Here message. The top line of the screen displays the sender username, followed by the message timestamp. Clique Here message content appears below this line. If the message contains an audio clip, a loudspeaker icon appears to indicate the attachment's existence, and the user can hear the clip by selecting "Play voice IM" from the options menu. The next component that may appear is a full-screen version of the image attachment, if one exists. Finally, if a text component of the Clique Here message exists, it will appear at the bottom of the screen.

2.2.7 Roster Message Screen

Clique Here allows users to communicate with *all* of their contacts in an unintrusive fashion using text, audio, and image media to provide expressive availability information, to share important events, or to share a random glimpse into their daily life. This broadcast function is provided to the user through the *Roster Message Screen.* This screen is a simplified version of the top portion of the *Chat Screen*, allowing users to quickly create and add content to Clique Here messages. The important difference in the function of this screen is that these *Roster Messages* are intended to communicate information to all of the user's social contacts, and thus the messages are delivered to each contact on the user's roster. Also, when a contact receives a

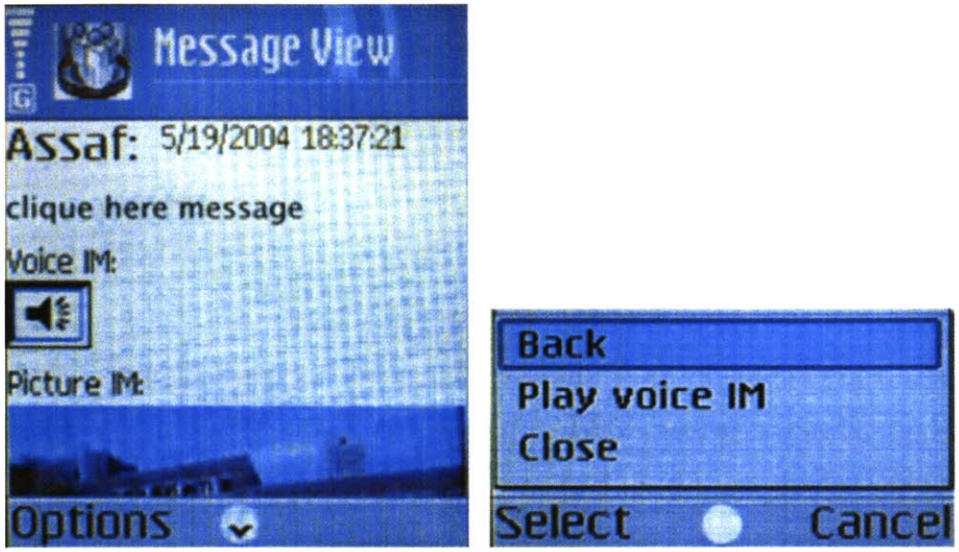


Figure 2-8: The Message Detail Screen presents all components appearing in a given Clique Here message (image, audio, or text).

Roster Message, the *Picture Alert Screen* is not displayed because it would be distracting to do so in response to each broadcast message. The user's *Contact List Item* is updated, but the recipient is not interrupted with an alert. This way, recipients will notice the information when they are interested in the user's availability and they decide to browse the user's *Contact List Item.*

2.3 Home Client

The Clique Here home client interface was designed to address the low technology threshold of its target usergroup, parents and grandparents. Both Mynatt et.al. in the Digital Family Portraits project [16] and Lakshmipathy et.al. in the TalkBack answering machine project [10] make use of a public display, in the form factor of a typical picture frame, to minimize the technology barriers of their systems. Incorporating the user interface into a familiar household object makes the presentation of information natural and unobtrusive. The Clique Here mobile client has adopted this user interface concept.

The home client runs on a wireless tablet-style device, shown in Figure 3.5 that

Figure 2-9: The Roster Message Screen allows the user to quickly create a Clique Here message and send it to the contacts on his roster.

was originally designed to provide consumers with a convenient, portable way to browse the web. With the touch-screen and the ability to run custom Java applets, this tablet device was an ideal candidate to support a simple, accessible home client. Ideally, the client would be placed in a visible, but out-of-the-way, place in the home, allowing the content being displayed to provide awareness, without being intrusive. Once logged onto the home client, the latest images sent from each of the user's contacts are part of a looping slide-show that runs full-screen on the device. Below each image, the contact username is printed, followed by a text message, if one was included in the Clique Here message. If no text message was attached, the contact username is followed by a timestamp indicating when the message was created. When a new message arrives, a subtle audio icon is played, to notify the user, in case they're available and interested. Should the user wish to inspect a particular message, they simply touch the client screen when the image message appears. This will halt the image scrolling, and any audio attached to the message will be played. When the user touches the screen a second time, the normal scrolling mode continues.

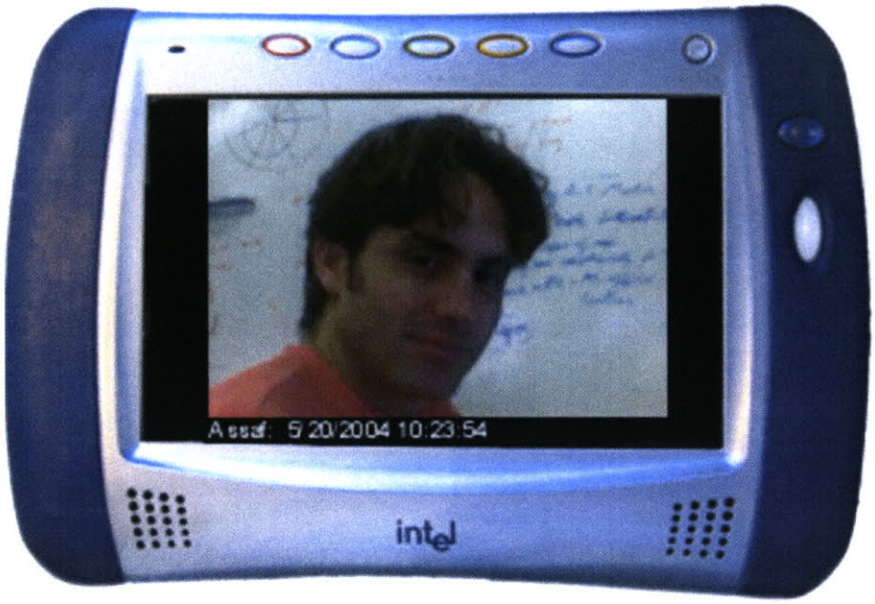


Figure 2-10: The Clique Here home client.

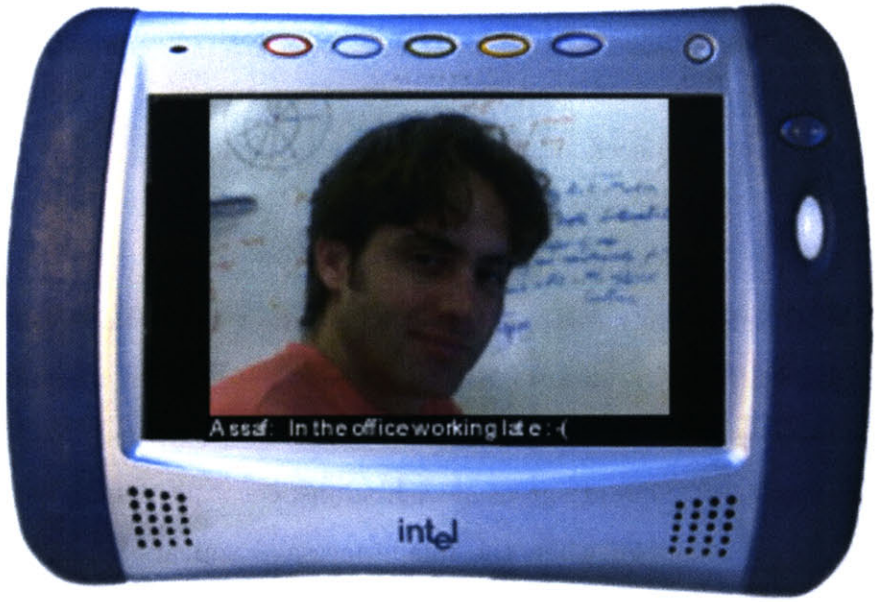


Figure 2-11: If a user has included text in a Clique Here message, it will be shown in the Clique Here home client below the user's current image.

Chapter 3

Design and Implementation

This chapter describes the software architecture of the Clique Here system. At a high level, Clique Here is a client-server messaging system. Many systems fitting the same description have been designed very well and implemented for the fixed Internet. As a multi-media messaging application for the mobile Internet, the Clique Here design space is less mature. Many of the Clique Here system requirements map to computational, bandwidth, and API capabilities that have only recently become available, and thus, few best practices exist to assist design decisions.

This chapter is divided into five sections. The first section discusses system-wide design decisions and rationale at the software level. The second section discusses design decisions at the protocol level, and the relationship between these decisions and decisions presented in section 3.1. The third section begins the discussion of each Clique Here component by introducing the Clique Here server. The fourth section describes the focus of the design effort for this project, the design of the Clique Here mobile client. Finally, this chapter concludes by explaining the Clique Here home client design.

3.1 System Wide Constructs

The Clique Here system is a 100% pure Java implementation. The Clique Here server is implemented with Java Servlets, the Clique Here home client is a Java

Applet, and the Clique Here mobile client is a Java MIDlet. Details on the design and implementation of each of these individual components is provided in sections to follow. In this section, we discuss the system constructs that reap the design benefits of Clique Here's consistent use of Java.

3.1.1 Java Package Structure

Figure 3-1 shows the Java package structure of the Clique Here system. The pure Java system implementation allowed classes used by all components to be "factored out", and placed in the `cliquehere` package. Component classes are placed in packages below the `cliquehere` package, each depends on the `cliquehere` package, but there are no dependencies between component packages. Component packages also depend on the `cliquehere.exceptions` package. Each component compiles with a different JDK, J2ME MIDP 2.0 for the mobile client, J2SE 1.3 for the home client, and J2EE 1.4 for the server. This Clique Here package structure simplifies system builds and incremental updates to shared classes since if an update is made to a shared class, the change does not have to be copied somewhere so that other components can build against the change. Instead, components build the source within their package with their respective JDK, as well as the source in the shared `cliquehere` and `cliquehere.exceptions` packages.

3.1.2 ChatConstants Interface

The Clique Here package structure not only allows the system to share important classes for reuse and modularity, but it also provides a mechanism to maintain consistency in constant values that must be used throughout the system. The `ChatConstants` interface does exactly that. Several classes in each component package implement the `ChatConstants` interface to gain access to Clique Here globals.

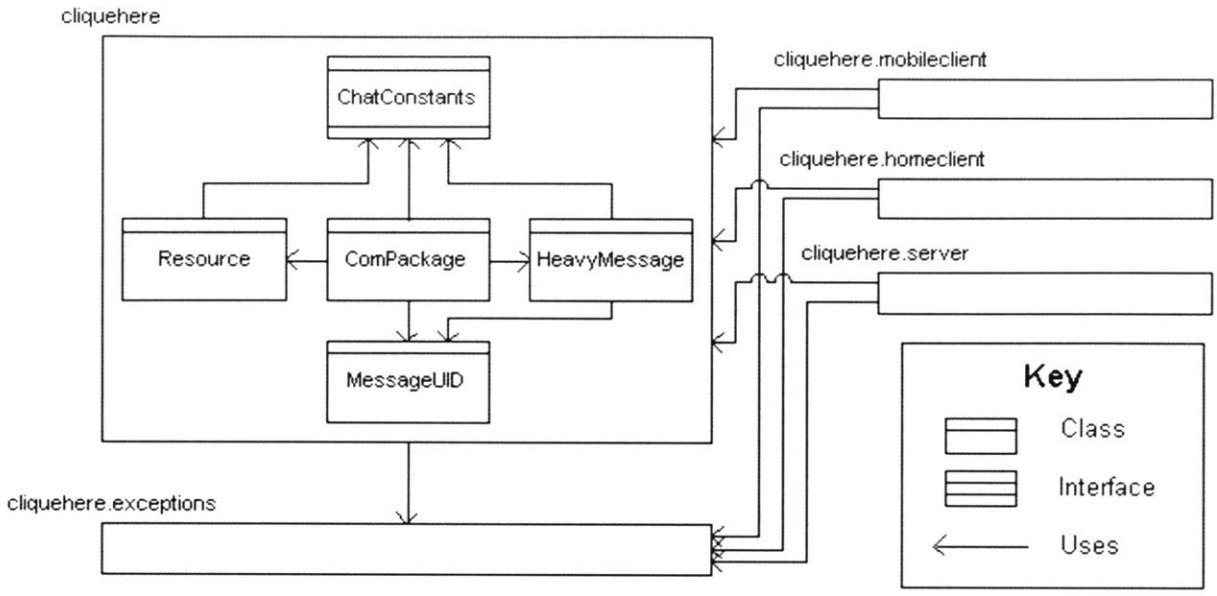


Figure 3-1: The Clique Here package structure takes advantage of the fact that each Clique Here component has been implemented in some form of Java.

3.1.3 ComPackage Class

The `ComPackage` class encapsulates the different forms of data that must be communicated between client and server to support the functionality of the system. In general, a `ComPackage` has two components, a package type and payload data. The package type identifies the purpose for the communication, and the payload data may be different, depending on the package type. For example, when a client sends a `ComPackage` of type `SMILE_MSG` to another client, via the server, there is now payload data added to it, but when the recipient processes the packages, the semantics of the package are clear just from the type. Also, if a client sends a `NORMAL_MSG` to another client, via the server, when the recipient processes the message, she will know tho expect a `HeavyMessage` object as payload. Table 3.1 lists each possible `ComPackage type`, and the entities that send and receive each. MIDP 2.0 Objects do not provide their own serialization. To cope, serialization methods were defined manually for `ComPackage` objects, and for all objects contained within.

The functionality the the `ComPackage` class provides is a classic fit for utilizing the power of object-oriented design with inheritance. Instead of these `ComPackage`

Type ID:	Sent By:	Sent To:
GROUP_MSG	client	group of clients [1]
NORMAL_MSG	client	one contact client
SMILE_MSG	client	one contact client
WAVE_MSG	client	one contact client
ROSTER_REQUEST	client	server
ROSTER	server	client
SERVER_MSG_UIDS_REQUEST	client	server
SERVER_MSG_UIDS	server	client
COM_PAK_BATCH	client OR server	server OR client [2]
POLL_FOR_NEW_COM	client	server
RETRIEVE_OFFLINE_MSGS	client	server
LOGIN	client	server
QUERY_ERROR	server	client
RETRIEVE_MSG	client	server
DELETE_MSGS	client	server
MSGS_DELETED	server	client
ACK	server	client

Table 3.1: `ComPackage` types and their movement.

objects having a type field, and different payload objects that are dependent on this field, each package type could be implemented as a subclass of `ComPackage`. This approach was not selected, however, because it would result in approximately twenty more classes to be added to the system. Issues relating the number of classes in the system to JAR file size and performance are discussed in Section 3.4.

3.1.4 HeavyMessage Class

In contrast to `ComPackage` objects that support communication between client and server, `HeavyMessage` objects encapsulate the data that makes up a Clique Here message. `ComPackages` of type `GROUP_MSG` and `NORMAL_MSG` contain a `HeavyMessage` object as payload. Each of the Clique Here system components (server, mobile client, home client) depend on HeavyMessage objects to move Clique Here messages between each other. Image, audio, and text attachments are stored as member variables, along with meta-data (e.g. sender, timestamp, etc.). Since `HeavyMessage` objects store the multi-media attachments as member variables, they consume a significant

amount of heap resources. To address this, clients implement a wrapper class around `HeavyMessage`, which will be discussed in section 3.4.

3.1.5 Exception Strategy

With the complexity of the Clique Here system, error handling and exception strategy could not be an afterthought if the system was to be maintainable. In Java, exceptions are the subject of many debates. The language construct offers a powerful tool for dealing with error conditions, but exceptions are not used consistently in practice. In an article published on *JavaWorld* [5], Brian Goetz presents a novel technique he calls "Exception Chaining" to balance the trade-offs between different exception strategies.

All Java exceptions inherit from the class `Exception`. Progeny of this class may be general, such as the `IllegalStateException` or they may be very specific, as in the `NoRouteToHostException`. The issue addressed by the article is the relationship between the specificity of exceptions thrown by developers in the methods they write, and the burden placed on developers who call these methods.

It is cumbersome to catch several exceptions when calling a method. If the burden is too great, it could lead to the poor programming practice of catching the superclass `Exception`. However, if more general exceptions are thrown, then crucial information describing the error condition will not be available at runtime. Exception chaining resolves this issue by allowing more general exceptions to be thrown, while preserving import debugging information. This is done by writing a wrapper around the `Exception` class. This class has the same interface as the `Exception` class, except for an additional constructor that can take an exception as an argument. In Clique Here, this wrapper class is `ClientException`, and all other user defined exceptions in the system are sub-classes.

With exception chaining, a developer implementing a method can catch the specific exception thrown in his code, and *translate* it to something more general by using that exception as an argument to construct a `ClientException`, or one of its sub-classes. If the exception is thrown at runtime, the stack-trace of the general exception

will include the stack-trace of the translated, specific exception as well.

3.2 Communication Protocol

This section describes the protocols selected for client-server communication.

3.2.1 Network Transport Protocol: HTTP vs. Sockets

Clique Here is a client-server messaging application. Clients must send messages asynchronously to the server for routing, and appropriately, recipient clients must receive these messages asynchronously from the server. For these basic requirements to be met, Clique Here clients need both a *pull* and a *push* from the server. This need alone points the Clique Here network transport protocol choice toward two-way channels such as TCP sockets; instant messaging solutions of the fixed-Internet have gone this route. However, this choice required a careful evaluation against the important design constraint created by the limited data capabilities of the mobile device.

TCP is a connection-based protocols, meaning that even when application data is not moving between client and server, bits will be moving to maintain the connection. This is undesirable when bandwidth is small and costly, as with the mobile device. A second issue with TCP is their support is only *optional* in the MIDP 2.0 specification, the API used to develop the mobile client. The Clique Here mobile client hardware does claim to support sockets, but developer forums have cited many issues in practice. Also, mobile data connections do not maintain consistent availability. Walking into basements, elevators, or some buildings can cause the connection to be lost. Since TCP sockets need to maintain a constant connection, if the network is not available, the socket connection will be lost, and the connection will have to be re-established when the device has better data service.

HTTP support, however, is *required* for devices implementing the MIDP 2.0 specification, and connections are terminated after after the request-response exchange is complete. This means that the protocol will fail more elegantly when the network

connection is unavailable; if an HTTP request fails, it can simply be resent periodically, until it does become available. Also, many MIDP 2.0 forums recommend the use of HTTP over TCP or UDP for client server applications. The difficulty with HTTP is that it does not provide a mechanism for the server to *push* data to the client asynchronously. Considering each of these trade-offs, HTTP was selected as the network transport protocol for Clique Here, and the application uses a common technique to resolve the *push* issue. The client polls the server at short (5s) intervals to determine if new data is waiting, and if so, the data is downloaded. The means that bits are moving across the connection, event when data isn't being sent, just like with TCP sockets. With HTTP polling, however, the application has control of this data movement, which is important because should it be desired for network resources to be conserved, for example, the polling interval could be increased.

3.2.2 Application Protocol: Binary vs. XML-RPC

Two major directions for the Clique Here application protocol were considered. The first was to implement a custom binary protocol, and the second was to leverage popular XML based open message passing protocols such as SOAP or XML-RPC. The trade-off between these choices is overhead versus flexibility and extensibility. XML driven protocols create overhead in two different areas. First, an XML parser does not ship with MIDP 2.0 implementing devices, and thus one must be added to the application package. This will add approximately 100k to the jar file size[19]. The second area of increased overhead is in the size of messages. With these XML protocols, a message with zero data payload, even with XML compression standards such as WBXML, the message size will still be on the order of one kilobyte in size[19]. This is a significant issue since the choice of HTTP as a network transport layer requires the Clique Here client to poll the server with a minimal data message, every few seconds. After reviewing the trade-offs between these two directions, it was determined that the flexibility benefits of these XML protocols did not outweigh the costs. Thus, Clique Here implements a custom binary application protocol designed to preserve as much flexibility as possible.

3.2.3 Recursive Serialization

A `ComPackage` may have the type `COM_PAK_BATCH`. In this case, the payload of the `ComPackage` is a vector of other `ComPackages`. This allows many `ComPackages` to be sent as one request. This becomes important when communicating over HTTP because the round-trip time per request is a significant cost, 8s on the Nokia 6600 to the server when only a trivial amount of data is sent. Batching these packages limits the round-trip cost to one request. When a `COM_PAK_BATCH` package arrives at the client or server, it is deserialized to extract the payload `ComPackages`, then these are individually sent to the switch statement for processing based on type.

3.3 Clique Here Server

The choice to use J2ME MIDP 2.0 to develop the Clique Here mobile client was based on an evaluation of that API against the project requirements. The Clique Here server platform choice, J2EE 1.4 Servlets, was made to best support the client API decision. Compared to the Clique Here client implementation, the Clique Here server is rather simple, serving two primary purposes, to appropriately route Clique Here messages, and to store Clique Here message state.

The server is deployed with the Apache Tomcat 4.0 application server. Tomcat is open-source software that offers typical webserver functionality, though not optimally, and supports client-server applications developed with either Java Servlets or JavaServer Pages.

With HTTP selected as the Clique Here network transport protocol, and with serialized `ComPackage` objects selected as the application protocol, the general communication flow is as follows. All requests made by the client to the server are done via HTTP Posts, and a serialized `ComPackage` object is written to the output stream. The `ComPackage` type may be any of those listed in Table 3.1 which are indicated as being sent by the client, or if the client has no particular request to make, `POLL_FOR_NEW_COM` requests will be made at a periodic interval. The server will receive the request, deserialize the bytes into a `ComPackage` object, and identify its

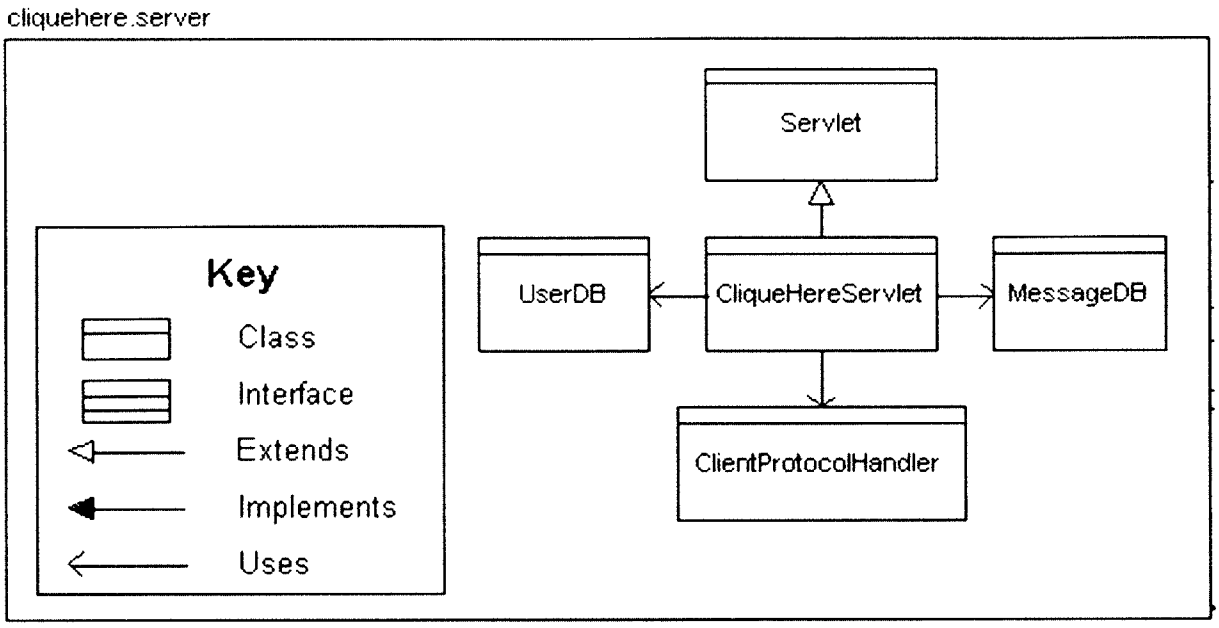


Figure 3-2: This Module Dependency Diagram shows the uses, implements, and extends relationships between the classes making up the Clique Here Server.

type. Based on its type, the server will perform the requested action, if provided valid data. The server closes the loop by creating the appropriate response `ComPackage`, serializing it, and writing the bytes to the response stream.

This section describes the role that relevant Clique Here server classes have in the process described above, implementation details of the server, and provides rationale for design decisions.

3.3.1 CliqueHereServlet Class

Receiving and responding to HTTP requests is done by the `CliqueHereServlet` class, which extends `Servlet`. Java Servlets give you multi-threading for free, meaning that with each new request, a new thread of execution is forked off to handle and respond to the request. When the `CliqueHereServlet` object receives an HTTP request, it quickly passes the input and output streams to the `ClientProtocolHandler`, where the processing of the request is done. Once the `ClientProtocolHandler` has processed the `ComPackage` and written the response to the output stream, execution is returned to the servlet class, and the thread terminates.

3.3.2 ClientProtocolHandler Class

To hand-off `ComPackage` processing to the `ClientProtocolHandler`, the `CliqueHereServlet` calls the handler's `doPost()` method. The handler first reads the bytes from the input stream and deserializes them into a `ComPackage` object. The package type is then extracted and applied to a large switch statement containing clauses to handle every possible incoming `ComPackage` request. These handler methods generate the appropriate `ComPackage` response, serialize it, and write the bytes to the output stream.

To support the "virtual push", accomplished through small interval client polling, packages sent to a user need to be added to a queue on the server that can be emptied when a `POLL_FOR_NEW_COM` request arrives. If the handler receives a package for a user, and that user is online, then the `ComPackage` is added to such a queue. When a user polls for messages, and the queue is not empty, the `ComPacakge` objects in the queue are sent to the user as a `COM_PAK_BATCH`.

If the recipient is offline, however, the `ComPackage` is not stored in this queue. This queue is solely for the purposes of supporting the "virtual push", and is thus irrelevant to offline users since they won't be polling. Regardless of if a user is online or not, if a package arrives for him that contains a `HeavyMessage` object, then the message is stored in the `MessageDB` object for later retrieval.

3.3.3 Media Transcoding

Audio recorded on the Nokia 6600 is encoded with the 3GPP Adaptive Multi-Rate (AMR) codec, an open speech codec which significantly compresses the size of speech audio files. The Clique Here home client hardware can only play 8kHz 8-bit audio, encoded in Sun's AU format. To allow audio clip portions of Clique Here messages to play on the home client, the server transcodes the audio from AMR to the AU format with the correct parameters.

When the server queues a message for a user, it determines if the user is logged into a home client, and thus needs the message audio transcoded. If so, the server

shells out to two binary audio conversion utilities in serial. First, the server calls a Sony-Ericsson AMR to WAV converter. Then, the server uses the SOX utility to make the WAV to AU conversion. The transcoded audio is then copied to a folder accessible by the webserver so that the home client Applet can retrieve the audio.

Image media may also require transcoding. Images captured on the 6600 have VGA resolution (640X480). This data is sent to the server at full resolution so that it may appear with the best possible quality if routed to a home client. However, contacts logged onto mobile clients have no need to receive the image at full resolution since their display can only display images as large as 174X143 pixels. Therefore, to improve resource efficiency, the server scales down images that are addressed to mobile clients.

3.3.4 MessageDB Class

CliqueHere messages are stored on the server until they are deleted in order to support the Clique Here message caching strategy, which is described in detail in section 3.4.3. Message persistence on the server also allows users to retrieve messages received while offline. The `MessageDB` class maintains this state. Messages are stored in this object within a member variable collection object, which means that these persistent messages are stored on the server heap. This strategy is adequate for the purposes of this project. However, a production system would need to store its state in a relational database.

3.3.5 UserDB Class

Like persistent message data, user data is stored in a class object, namely `UserDB`. The `ClientProtocolHandler` object uses data stored in the `UserDB` object to determining who is on each person's roster, what groups contacts are placed in, etc. Again, if this were a production system, this state would need to be stored in a database.

3.4 Clique Here Mobile Client

The Clique Here mobile client is implemented as a J2ME MIDP 2.0 MIDlet running on a Nokia 6600. Among the components of this system, the mobile client has both the broadest functional requirements and the tightest resource constraints, making it the most important subject of design in the system. Best practices offered by MIDP device manufacturers stress the importance of minimizing the size of a MIDlet's JAR file[20], a file containing the compiled application. Some older devices will not even accept a JAR file over 50k. The size of a JAR file, of course, depends on the total amount of code contained in the file, but more importantly, it also depends on the number of classes in the file, regardless of the amount of code contained in each class. For this reason, these best practices suggest avoiding the use of design patterns that require additional classes and to limit the use of Java interfaces. Unfortunately, following this best practice would be a severe hindrance to effective software design, and it would leave an application, of the complexity of the mobile client, inflexible and unmaintainable.

The approach taken to design the mobile client was to ignore this best practice regarding JAR file size and to use whatever principles of software design that were judged to yield a robust, flexible, and maintainable application. The only exception to this is the implementation of the `ComPackage`, described in Section 3.1.3. Also, the catch to this plan was that if performance issues appeared due to the size of the JAR file, then an effort would be made to combine classes, but only as an optimization step after the system was functionally complete. In practice, the JAR file size never became an issue, and the software design efforts that went into the client were preserved.

This section highlights the important design decisions made in the development of the Clique Here mobile client.

3.4.1 Model View Controller

Model View Controller (MVC) is a popular design pattern among GUI applications. Introduced by Steve Burbeck at Xerox PARC [1], the pattern calls for the software

to be separated into three components (Model, View, and Controller) and to allow only certain dependencies between each component. The role of the *model* is to manage the behavior and data of the application domain, respond to requests for information about its state, and respond to instructions to change state. In terms of dependencies, the *model* has no knowledge of either the *controller*, or the *view*. The *view* component manages the visual display of state provided by the *model*, and the *controller* component manages the user's interaction with the system.

The Clique Here mobile client implements a common adaptation of the basic MVC pattern. The *view* and *controller* components are combined into one piece. This is done implicitly because MIDP 2.0, like most modern languages, has a GUI toolkit, `javax.microedition.lcdui`, used to create both *view* and *controller* functionality, and the toolkit defines the relationship between the two.

The Clique Here mobile client module dependency diagram, shown in Figure 3-3, graphically describes the implementation of the MVC pattern. The *model* and *view/controller* are only connected by one dependency, `CliqueHereMIDlet`, the central *view/controller* class, depends on `Client`, the central *model* class. The *model* does need to push data to the *view/controller* in some circumstances, but the `MessageListener` interface maintains the abstraction.

The MVC pattern was implemented by the Clique Here mobile client to provide the application with flexibility. With no dependence existing from the *model* to the *view/controller*, a significant amount of re-use could be achieved if the client had to be ported to a different MIDP device, especially if that device was a MIDP 1.0 device. Since most MIDP 2.0 improvements over 1.0 are in UI functionality, porting the client to a MIDP 1.0 device would require a significant rewrite of the *view/controller*. The Clique Here mobile client *model*, however, only depends on the MIDP 1.0 API, and it could be completely re-used.

3.4.2 ResourceConsumer and ResourceHandler Classes

As the Clique Here mobile client is used to capture messages, the audio capture device will be loaded and unloaded several times. Similarly, on many occasions the

audio playback device will be started, stopped, loaded with new data, etc. The image capture device goes through similar cycles. The MIDP 2.0 API's used to control the states of these multi-media devices are convoluted and inconsistent among devices. This leads the common task of performing such state changes to be burdensome and error prone. The `ResourceHandler` interface was created to address this issue by providing a clear and consistent interface for setting the state of objects that must acquire scarce resources, and complying to a consistent contract for the result of such state changes.

The `UNREALIZED` state is the lowest resource consuming state of the interface. The `UNREALIZED` state is different than the others since the only method that can be called to promote the object from this state is `realize()`. This is because, by definition, an object in the `UNREALIZED` state is not associated with any state data (e.g. an audio clip to play, a certain format to use for audio or image capture, etc.). The `realize()` method will be called with the required state data as an argument, and since the object cannot be promoted without this data, `realize()` is the only method that can promote the object to the next highest state, `REALIZED`. The `REALIZED` state is defined as the lowest resource consuming state where the object is associated with the any required state data. The `unrealize()` method, however, can be called when the object is in any state, and the result will be that the object is demoted to the `UNREALIZED` state.

The `prefetch()` method promotes the object from the `REALIZED` state to the `PREFETCHED` state. This state is defined as the state where all prefetchable scarce resources have been acquired, and the thus the state offers the shortest latency for starting the object. To return to the `REALIZED` state from the `PREFETCHED`, the `deallocate()` method must be used. The next promotion, from `PREFETCHED` to `STARTED` is achieved by calling the `start()` method. The `STARTED` state definition is intuitive (audio recording, audio playing, image being captured). The object is demoted back to the `PREFETCHED` state by calling the `stop()` method.

In the description of this interface, all method calls have been introduced as either promoting up one level or demoting one level. The contract for a promotion method,

however, is that if the current state is less than the promotion state, then the result of calling the promotion method will be moving the object to the promotion state. This means that if the object is in the `REALIZED` state, and the `start()` method is called, `prefetch()` is implicitly called and the object moves to the `STARTED` state. The only exception to this, described above, is when the object is promoted from the `UNREALIZED` state. Demotions carry an analogous contract. Also, if a demotion method is called when the object's current state is less than or equal to the demotion state, then the call is ignored. The same goes for promotion methods, if the object is currently in a higher state than the promotion state, the call is ignored.

The `ResourceConsumer` is intended to be a common interface to screens that contain the multi-media components. It is almost identical to the `ResourceHandler` interface, only it does not support the `STARTED` and `STOPPED` states. Screens have this subset of functionality because it does not make sense for a screen to have a public `start()` and `stop()` method.

3.4.3 Cache Strategy

The Clique Here mobile client provides the user with persistence of their Clique Here messages. To mitigate the storage burden associated with this feature, while optimizing the latency users will have to tolerate in order to view their Clique Here messages, the Clique Here mobile client implements a three level caching strategy for multi-media messages. When Clique Here messages are transported over the network, they are delivered to the client as `HeavyMessage` objects within the payload of `ComPackage` objects. The `HeavyMessage` objects contain multi-media data that was sent with the message. If the Clique Here mobile client were to simply keep a collection of `HeavyMessage` objects to provide users with persistent messages, heap resources would quickly run low rather quickly. Instead, the mobile client defines a wrapper class around `HeavyMessage`, `LightMessage`.

Soon after `HeavyMessages` arrive at the `Client`, a `LightMessage` is constructed, using the `HeavyMessage` as a parameter. The `LightMessage` object offers the same interface to the user as the `HeavyMessage`, but there are two significant differences

between the classes. First, if the HeavyMessage contains an image, a thumbnail image is created, and added to the `LightMessage`. This small amount of image data is always part of the `LightMessage`, and can be used to represent the image, regardless of where the full image is in the cache hierarchy. The second major difference is that the full-size multi-media may not always be stored as a member variable of the `LightMessage` object, and thus stored on the heap. The data will lie in one of three places in the multi-mdedia data cache hierarchy. Clique Here message persistence is provided by the `Client` maintaining a collection of `LightMessages`. The smallest level of cache, but most expensive, is the heap. When the `LightMessage` is first created, the `HeavyMessage` object is stored as a member variable. When, the `getAudio()` or `getImage()` methods of `LightMessage` are called, the `LightMessage` object simply calls the methods of the same name in the`HeavyMessage` object, which is stored as a member variable. The `Client` class keeps track of which `LightMessages` have their data stored on their heap, and it tracks LRU data. When a new `LightMessages` is created, if the maximum number of `LightMessages` storing their data in the heap is reached, then the LRU has the `HeavyMessage` removed from its heap, its member variable storing the message is set to `null`.

The second level of cache in the system is local persistent memory. When a `LightMessage` is created by a client, if it contains multi-media data, the associated `HeavyMessage` object is serialized and written to local memory. The system also has a maximum number of messages that can be stored in local memory, and again, when that maximum is reached, the LRU is removed from local memory. If the `getImage()` or `getAudio()` methods are called when the data is stored in local memory cache, the `LightMessage` object will first check its local heap to determine if it has the message, if it does not, then it will call a method from the `Client` object that will return the cached `HeavyMessage`, if it is present.

The third level of cache is storage over the network on the server. Clique Here messages are stored on the server until they are deleted. A maximum number of messages could be set for this level as well, and the system could just force the user to delete messages if the max was reached, but the maximum should be considered to

be a large number since storage on a server is cheap. If the `getAudio()` or `getImage()` methods of a `LightMessage` object are called, and the multi-media data is neither stored on heap as member variable, nor in persistent phone memory, then the `Client` sends a `RETRIEVE_MSG ComPackage` to the server, and by contract, the server will return the appropriate `HeavyMessage`.

3.4.4 ComEngine Interface and HttpComEngine Class

Section 3.2.1 describes the rationale behind the decision to use HTTP as the network transport protocol. However to maintain flexibility in the system, the mobile client was designed to abstract the choice of network protocol. This abstraction is provided through the `ComEngine` interface. The `Client` class depends only on the `ComEngine` interface for communications capabilities which provides the `Client` object with every function necessary for communicating with the server, but the interface methods are only dependent on the communication protocol at the application protocol level, since the interface references `ComPackage` objects. The `HttpComEngine` class implements the `ComEngine` interface, and provides the actual communication over HTTP. If future requirements and capabilities change such that communication over TCP sockets becomes more advantageous, the flexibility provided by this abstraction would significantly reduce the programming effort required to update the mobile client software.

3.4.5 Screen Navigation

Windows based OS, found on most desktop computers, allow the application developer to create modal dialogs, windows that perform a certain function and return control to a parent window upon completion. Most mobile devices, including the Nokia 6600 adopt a screen based UI, which is similar in function to a web browser. Only one screen may occupy the display at a time, and there is no mechanism provided to support a parent-child relationship between two screens. Many times in the mobile client UI, however, the user will navigate to a screen and he will wish to return to the screen he came from once he dismisses his current screen. Small MIDlet appli-

cations, with few screens, can get away with hard-coding the application's behavior upon screen dismissal. The scope of the Clique Here mobile client required a more general approach.

The `Utils.BreadCrumbTrail` interface is implemented by every `Displayable` class in the mobile client, and by the `PresentLifeMIDlet` class. Also, when every screen is created, it is passed the `Displayable` object that considered its parent, casted as a `Utils.BreadCrumbTrail`. The only exception is the root screen, `ContactListScreen`, which is passed the `PresentLifeMIDlet` object, casted as a `Utils.BreadCrumbTrail`, when it is created. The `PresentLifeMIDlet` object maintains a stack representing the navigation path between the root screen, `ContactListScreen`, and the current screen. The `PresentLifeMIDlet` object is able to maintain this navigation history stack because all navigation in the system is done by calling `Utils.BreadCrumbTrail` methods. These methods provide several navigation options (e.g. moving back one screen, moving back to the root screen, moving to a new screen, moving to a new screen and crushing the history stack, etc.). The implementation of each of these methods within the `Displayable` objects is trivial, each method just calls the same method of its parent `BreadCrumbTrail` object, this idea is the source of the name "BreadCrumbTrail". This means that the method call will move all the way down the navigation stack until it reaches the `CliqueHereMIDlet`'s implementation of `Utils.BreadCrumbTrail`. At this point, the midlet appropriately changes the displayed object based on the method call and the navigation history stack. The bread crumb trail concept has enabled effective screen based navigation for the Clique Here mobile client, in a general fashion.

3.4.6 StatusAlert Class

Significant latency exists between many events in the mobile client. Sending an HTTP post to the server with a multi-media containing Clique Here message will take several seconds. Getting a response from the server will take even longer. Acquiring resources for audio capture and playback can take several seconds. Even switching to screens that are not constructed until a user navigates to them will take at least one second

to load. The mobile client attempts to mask this latency with effective thread use. If a user starts a high latency process, and his next actions are not dependent on the results of this process, then the process is forked off into a new thread and control is returned to the user. An example of this case is when a user sends a large Clique Here message. The user trusts the the message will be sent successfully, so he does not have to wait for confirmation that it was. Even if the send did fail, the system can retry on its own without user intervention.

The client also attempts to mask latency by forking off threads to prefetch scarce resources when it is probable that they will soon be used. An example of this case in the client implementation is when the *Roster Message Screen* is loaded, the audio capture object is prefetched, since there is a good chance that the it will be needed. However, many occasions exist where thread usage cannot adequately mask latency. In these cases, the user must wait until a process finishes before they can move on. If the length of time the the user waits is greater than 200ms, then the user will notice the delay and perceive the system to be unresponsive, unless an indicator of system status exists[2].

The `StatusAlert` class implements a progress bar that can be used to provide the user a responsive update of system status during periods of latency. A global `StatusAlert` object is created when the system is initiated to allow the screen to be immediately displayed when needed. The MIDP 2.0 does ship with the `Guage` class, which provides minimal progress bar like functionality, but the implementation on the Nokia 6600 contained known issues that rendered it unusable.

3.5 Clique Here Home Client

The Clique Here home client is implemented on an Intel Web Tablet, a consumer electronics device that was never released to the marketplace but was made available to the Media Lab for research purposes. With a wireless networking capability, embedded speakers, a touch-screen, and a specified ability to display J2SE 1.3 Java Applets, the device was and ideal fit against the functional requirements of the home

client application.

While developing the software to run on the web tablet, however, it was discovered that the web tablet's implementation of the J2SE 1.3 Java Runtime Environment was incomplete. Among many minor inconsistencies, two major absences from the J2SE 1.3 specification were the support of Java Swing UI components and support for socket connections. To work around these issues, the Clique Here home client uses the older AWT GUI toolkit and communicates with the server via HTTP in a similar fashion to the mobile client.

The home client runs on the web tablet as an `Applet` embedded in an HTML document. The HTML document is deployed from a web server on the same host that deploys the `CliqueHereServlet`. Deploying the applet from the host that runs the server is a requirement because the Java Applet security model will only let Applets establish connections to the host that deployed the Applet. The home client is started by sending a request for the HTML document containing the Applet, from the browser.

3.5.1 HomeApplet Class

The relatively low complexity of the Clique Here home client did not necessitate an implementation of the MVC pattern, as in the mobile client. Instead, the `HomeApplet` class both manages the client state and controls the user-interface. `HomeApplet` extends the `Applet` class, and when the `init()` method is called, the client logs into the server the same way the mobile client would. With the simplicity of the design, the only intended interaction users are to have with the interface is touching the screen to stop the image scrolling, browsing the selected image, and listening to the associated audio clip, should any exist. If the user does this, then a smile message will be sent to the user that is being observed.

3.5.2 Communication Engine

Communication in the Clique Here home client is done almost exactly as it is in the mobile client. The `HttpComEngine` class, the class that provides the client with the methods to perform server communication, is implemented exactly the same in the home client, as in the mobile client. The `HttpPoster` class, the class that actually sends and receives data via HTTP, however, is different, as the J2SE 1.3 connection API much different than that of MIDP 2.0.

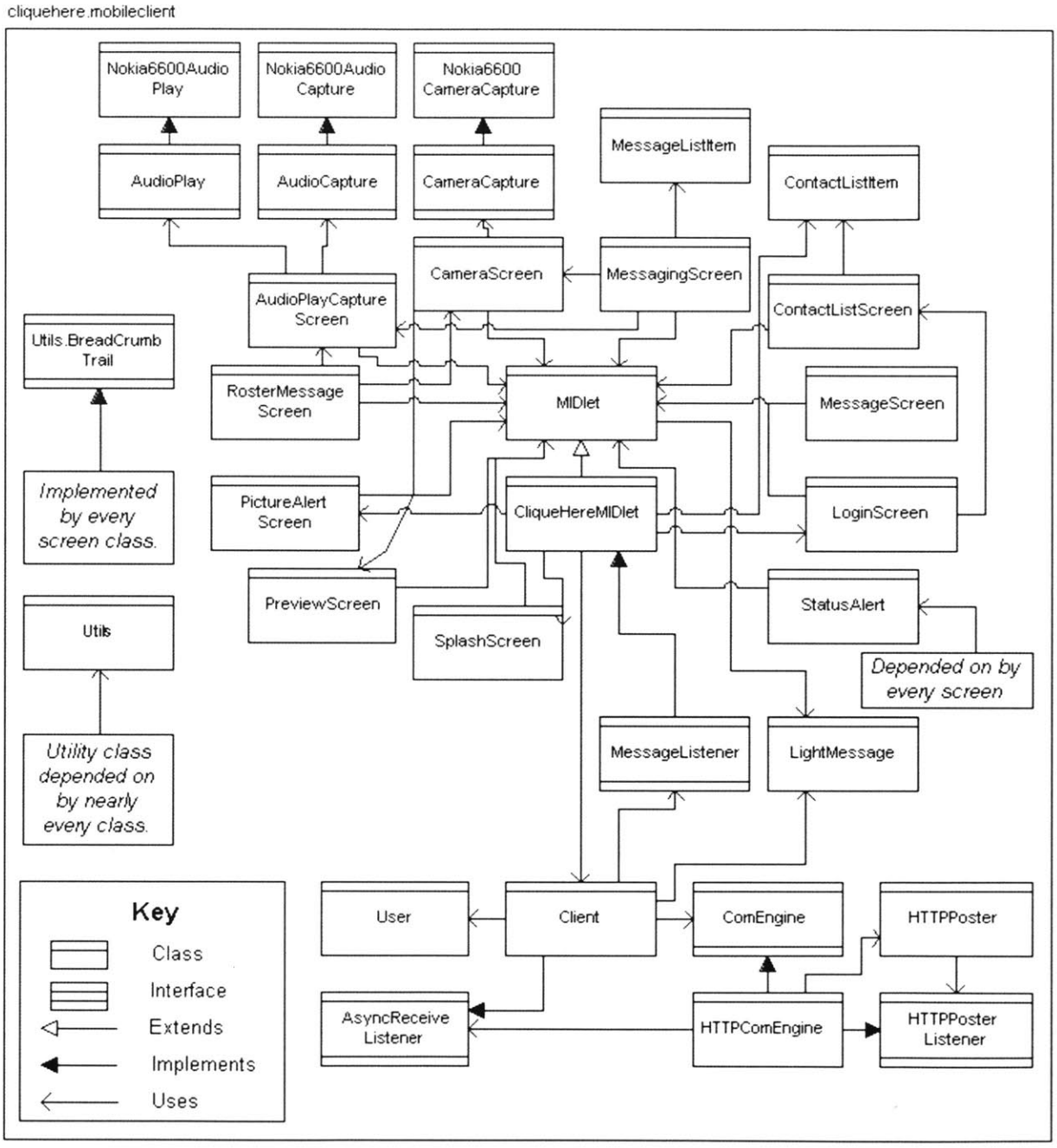


Figure 3-3: This Module Dependency Diagram shows the uses, implements, and extends relationships between the classes making up the Clique Here mobile client.

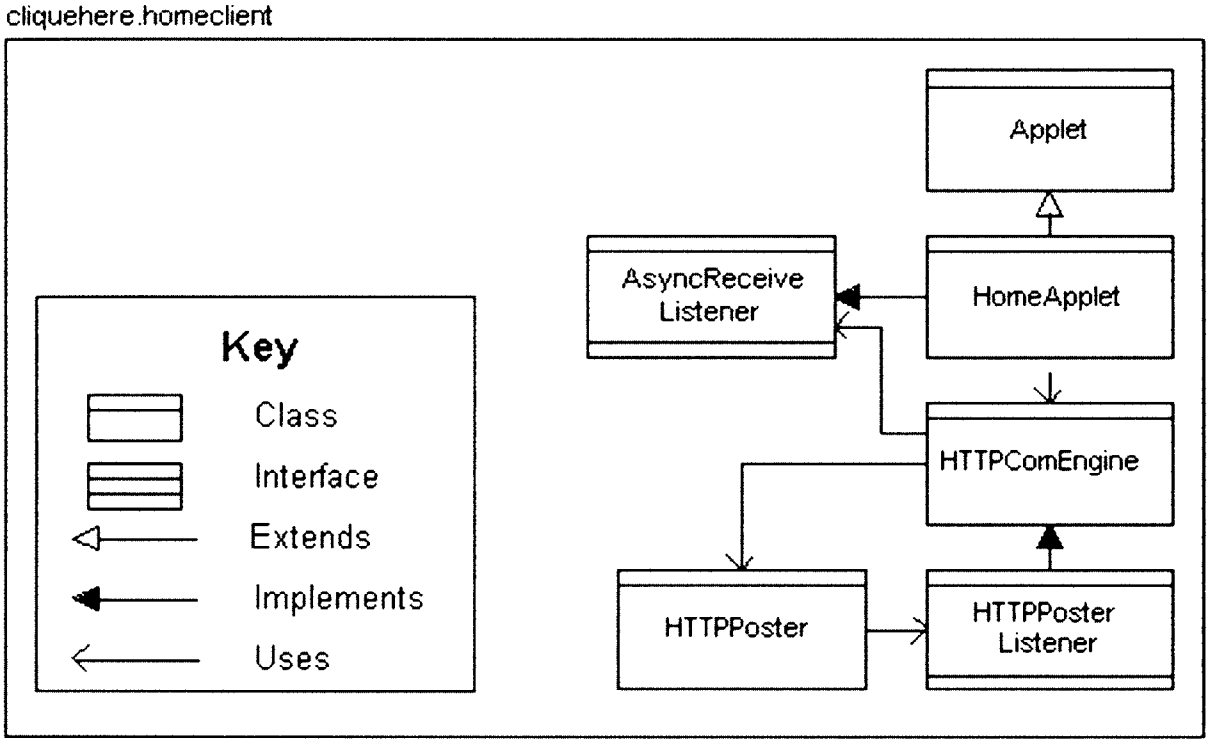


Figure 3-4: This Module Dependency Diagram shows the uses, implements, and extends relationships between the classes making up the Clique Here home client.

Chapter 4

Related Work

This chapter explores related research in order to ground the motivation for this thesis.

4.1 Instant Messaging

Instant messaging (IM) has become a popular communication modality used by people for social and professional purposes alike. Nardi et.al., in their substantial contribution [17], present results of an ethnographic study observing the expected and unexpected uses of IM in the workplace. The practical uses of IM show it to complement synchronous modes of communication by fostering a greater efficiency for particular communication tasks, and by fostering interaction that many not occur at all without the system.

The study identified several features of IM that, offered together, improve upon the shortcomings of traditional communication platforms. The list of such features is paraphrased in the list below.

Awareness: A value of awareness information is that it can assist users in making better decisions about when to initiate communication, which becomes more important as the intrusiveness of the desired method of communication increases. IM systems introduced the concept of a *buddy list*, a text list containing the usernames of contacts who are currently logged onto the system. This list is augmented with

icons and visual variables to indicate if a contact has recently been active on his PC, termed "idleness" by the system, and if he has set an away message. Together, these three pieces of data (online status, idleness, away message state) provide IM users with a sense of awareness of their contacts availability.

Plausible deniability: While the buddy list provides IM users with a sense of awareness of their contacts availability, the exact level of availability remains ambiguous. This means that if a user sends a message to one of his contacts, but does not get a reply right away, the instant message recipient could have ignored the message, or could just be away from his desk. The ambiguity provides the recipient with plausible deniability, and effectively redistributes the control of communication initiation evenly between sender and receiver, shielding the recipient from social obligation.

Immediacy: Given a capable network connection, instant messages arrive to recipients near instantaneously. When messages arrive, they typically invoke an audio alert, and a chat window is shown on the foreground of the PC display. Senders can have confidence that if the recipient is available, they will observe the incoming message. This offers an advantage over e-mail, where a recipient only sees new messages when she manually checks her inbox, and it even offers an advantage over the telephone where callers have to wait for a number of rings before they are connected.

Low attention demand: The asynchronous nature of IM, along with the ambiguity mentioned above, leads the system to afford multi-tasking. Users could talk on the phone and IM in parallel, or work directly on their computer and respond to incoming messages intermittently, as they choose.

The last three features highlighted above denote IM as a *lightweight* mode of communication. Lightweight communication can be generalized as communication that places minimal burden, of any kind, on either the sender or recipient. Lightweight communication makes communicating with social contacts more probable. Even despite the time constraints imposed by deadlines and necessary task oriented communication at the workplace, a person may find himself thinking about a family member in a brief moment of free time, but without lightweight communication, the family member being thought of probably would never know about it. It would be extremely

difficult for this person to call his family member to connect, but limit the conversation time to one minute. Even if he could, the phone conversation would require his full attention. Systems supporting lightweight communication, such as IM, enable people to make brief connections with their social contacts. Communications as simple as "good morning" or "I miss you" become reasonable, even in the presence of work obligations.

Lightweight communication can even make synchronous communication with social contacts more probable. With no awareness information about a social contact, a person might hesitate to call, knowing they might be busy, unless they had something urgent to discuss. Lightweight communication provides a socially acceptable way to query a contact's availability in the absence of adequate awareness information. For example, if a girlfriend IMs her boyfriend at work saying, "I need to ask you something when you have a chance", then the boyfriend will not be bothered by the query, and he can decide for himself how to respond to the communication. If he is very busy, he can just ignore the message, taking advantage of *plausible deniability.* He may also choose to respond via IM to find out more information, and maybe finish the conversation over IM. Alternatively, the message may have arrived during a period of free time, in which case he might pick up the phone to engage in a synchronous conversation.

Though Nardi's study focused on PC instant messaging clients, the lightweight and awareness features would be just as valuable to the mobile user. However, it is important to note that the overhead of text IM communication would depend on the input method available on a user's mobile phone. The most prevalent text input method on mobile phones is multi-tap, a method where each numeral key maps to the letters of the alphabet printed on the keys. Tapping the numeral once will input the first letter on the key. Tapping the numeral twice quickly will enter the second letter, and so on. If multi-tap were the only option, then text IM may require a significant amount of effort. This is one reason why voice IM was selected as a capability to include in the system. Of course, text input issues are irrelevant to voice IM, and it has the added benefit of indicating the emotion and urgency of the message more

effectively, allowing the recipient of the message to make a more informed choice of how to respond.

Systems similar to IM deployed in the mobile domain are SMS and MMS. These systems exhibit the features of lightweight communication, but a significant absence is the awareness feature. A buddy list concept is not employed, rather SMS and MMS rely on phone numbers and e-mail addresses to route messages. Despite the lack of awareness information, SMS has gained significant popularity, especially in Europe. A study conducted by Grinter et.al. [6] looked at the SMS usage patterns of teenagers in the UK. This study showed that the immediacy of SMS led many users to prefer it for communication, just as Nardi showed the same feature to promote the use of IM for desktop users. The consistency between the value users find in lightweight communication over SMS and the value IM users find in their system is not surprising. The absence of awareness support in the most common mobile messaging platforms, however, leaves a substantial gap in the value offered to users by these systems. Clique Here seeks to fill this gap by providing both awareness information and expressive lightweight communication to the mobile user. The next section describes the Hubbub project, which takes a different approach to filling the same gap.

4.2 Hubbub

Hubbub is an extension on the traditional instant messaging platform which explores the use of sound to improve support of awareness and opportunistic conversations [8]. Hubbub introduces two novel uses for sound within the IM context. First, each user is associated with a unique *Sound ID*, which provides the user identity in the audio space. Second, Hubbub clients can send *Sound Instant Messages* (SIM's), which are earcons sent asynchronously between clients through the system, like text IMs, and by convention, each earcon has an associated meaning. SIM's have meanings such as "HI" and "Thanks". When a SIM is received by a Hubbub client, the sender's *Sound ID* plays, followed by the SIM. Also, when the *Sound ID* plays, the GUI also displays

the username of the sender, to help the recipient to learn her contact's *Sound IDs.* Once she has learned these ids, the attentional demand of the system is decreased, since she does not have to be looking at the display to understand who sent the SIM, and what the meaning was.

A third new feature in Hubbub that is not offered in traditional IM systems is an activity meter associated with each *bub* (contact) on the contact list. Issacs et. al. cite the conclusions of Bonnie Nardi et. al. [17] that IM conversations commonly begin with "Are you there?" as and indication that the awareness provided by the *idleness* feature of IM is not sufficient. In traditional IM systems, one has to be inactive for ten minutes in order for their screen-name to appear idle in the buddy list. This delay does lead to ambiguity since a user may leave his desk, but he will remain active, according to IM, for the next ten minutes. The Hubbub activity meter indicates the user's level of activity within the previous fifteen seconds. Augmenting activity awareness in this way, however, would not be valuable for Clique Here. A certain amount of ambiguity in the system is embraced by Clique Here as a tool to shield users from social obligation, or as Nardi et. al. describe it, *plausible deniability.*

The strongest conclusion of the Hubbub project that relates to Clique Here is the fact that participants in the user study expressed that the auditory cues led to a stronger feeling of connectedness, even when the users were logged in across the country. This supports a portion of the primary goal of the Clique Here project, supporting social-connectedness with *awareness* and communication.

4.3 Live Addressbook

Milewski and Smith [15] defined two important trade-offs for designers of awareness systems. The trade-off of Overhead vs. Control characterizes the tension between easing the user burden of maintaining status information with automation and providing the user with control over the availability information being communicated. Systems that have chosen to emphasize minimization of user overhead have accomplished this by gathering data about the user from sensors and analyzing the data to

infer availability.

In the Clique Here system, since there will be no external sensors for the automatic inference of availability, the user has been given complete control regarding the availability information he wishes to communicate. To mitigate the overhead burden associated with requiring users to manage their own availability data, Clique Here allows users to communicate their context and availability by taking a picture with the digital camera equipped mobile device. Taking a picture is quick and easy, and other members of a user's social group who know the user very well, and possibly his surroundings, will be able to extract a rich amount of information from this picture. To give the user further control, the system will allow him to communicate context and availability via text and recorded voice.

A second trade-off defined by Milewski and Smith is Informativeness vs. Privacy. Privacy is an important issue to many users, and the design choice to give users complete control over their context information allows them to retain as much or as little privacy as they choose.

4.4 Watchme

The WatchMe project [14] is part of a growing research segment that explores how mobile communications technology can mediate communication within social groups and family units [3][4][11]. WatchMe combines detailed awareness information with multiple modes of lightweight communication to form a personal communicator that can keep intimate friends and family always connected. A user's awareness information is gathered in a highly automated fashion to minimize user burden. A sensing and classification module worn by the user tracks location, acceleration, and ambient audio; this information is then used by machine learning algorithms to determine current activities. A user's family members and intimate friends, a group of people called *insiders* by Marmasse et. al., can use WatchMe to view these classified activities in an iconic form and infer availability based on this information. This "always on" awareness provided by WatchMe enables *insiders* to initiate communication at

opportune moments and over the most appropriate channels (e.g. text IM, voice IM, phone call).

Clique Here draws a large portion of its inspiration from the WatchMe project. The Clique Here system, like WatchMe, seeks to mediate social communication by combining awareness capability with multi-mode lightweight communication. Unlike WatchMe, however, the Clique Here mobile client has been designed as a mobile phone application, operating on an un-modified mobile handset, rather than a custom hardware platform. Further, Clique Here focuses on the use of image media as a data type for communication.

Because Clique Here runs on conventional mobile hardware, the automated capture of detailed awareness information, provided by sophisticated sensors and machine learning, is not feasible. Instead, the Clique Here approach to awareness capability is to provide the user media rich channels to manually convey awareness (e.g. images and audio). Though this manual capture approach does place a larger burden on the user for maintaining current awareness information, capturing content for these multi-media channels is significantly easier than text input on mobile devices, and the richness of these channels allows a large amount of information to be conveyed

Marmasse et.al. recognized the importance for a person to know when his *insiders* take a moment within their day to think about him. Knowledge of these events would lead social group members to feel more connected, and might lead to opportunistic interactions. Such a simple and possibly frequent event, however, would not be reasonable for communication over mobile communication channels, even over the lightweight communication modes offered by both WatchMe and Clique Here. To address this limitation, Marmasse et. al. developed a non-verbal channel to communicate the concept of "thinking of you" between physically separated parties. The idea is that when an inquiring user observes the details of an *insider's* availability information, this indicates that the inquiring user is currently thinking about this *insider*. The observed *insider's* WatchMe client makes him aware of this event by displaying a static picture of the inquiring user that will automatically dismiss after a few minutes. In the excerpt below, Marmasse details four possible outcomes of this

mediated non-verbal communication.

- *The picture popping up may go unnoticed, especially since it disappears after a couple of minutes, so the "viewing" insider is not interfering with the "viewed" individual in any way.*

- *The "viewed" person notices the picture but decides not to reply or divert attention from his current action.*

- *The "viewed" individual notices the picture and responds by querying the availability of the other user, which causes his or her picture to appear on the others watch, sort of like an exchange of glances without words.*

- *The "viewed" insider decides to phone the "viewer" or engage in another form of verbal communication, i.e. text or voice messaging.*

The Clique Here system supports the communication of these innovative "thinking of you" events, referring to them as *smiles*, to utilize their ability to maintain connectedness and entice further communication. Further, because the Clique Here system uses image data to convey awareness information, these current images replace the static picture used in WatchMe to indicate a *smile*.

Chapter 5

Conclusion

Clique Here has been designed and developed to address current limitations of mobile communication to support social relationships, specifically in terms of maintaining social-connectedness. The Clique Here project has demonstrated the feasability of implementing a rich media awareness and lightweight messaging platform using existing mobile technology. In addition, this working, usable system provides a platform for research and discourse on the subject of mediating social connectedness with mobile devices.

The rapid improvement in camera phone technology, which has been driven by remarkable market demand, has out paced the development of applications that make best use of the technology. MMS and other current picture messaging solutions do not utilize the full potential images have to form an expressive communication channel. Following an iterative development process, the Clique Here system provides users with a simple interface to not only share interesting events in their life with social contacts, but also to use multi-media to make contacts aware of the their day-to-day life and encourage interaction at opportune moments.

No system that mediates communication will be able to replace the intimacy and expressiveness of communication within the same physical space. Systems like Clique Here, however, that support social-connectedness between physically seperated social contacts diminish the negative consequences of a common pitfall associated with modern life, a forced absence from those we care about. Today's parents must leave

the home more often to travel for business, children frequently leave the home to attend college, and both parents are employed for an increasing number of households. The current weak job market may lead young graduates to permanantly move away from the home to meet their own professional demands. With each of these trends testing the integrity of the modern family, solutions such as Clique Here will become crucial to lend support.

Future Work

Clique Here has made a demonstration of feasability, and provides a research platform that may be used to deepen the understanding of mobile mediated social communication. Future work on the system would benefit from a user evaluation to validate the value of Clique Here functionality, to identify areas in the system that need improvement, and to identify opportunities in the system for expansion. Subjects of the study would need to be part of the same social group, and they would need to spend a substantial amount of time physically apart from each other.

The next logical implementation task for the Clique Here project would be to develop a PC client. Many people spend a significant number of hours per day in front of a personal computer. Unlike the home client, the PC client should be designed for a user with a relatively high technology threshold who is comfortable with PC use. Unlike the mobile client, the PC client does not have to be designed around resource constraints and the feature set may be different since the user will not be mobile.

A significant design improvement could be made to the Clique Here system if a solution better than HTTP polling could be found to yield a *push* from the Clique Here server to the mobile client. While the polling strategy meets the functional needs of the system, in practice, activating the GPRS radio every ten seconds to make the request results in a noticeable decrease in battery life. Increasing the polling interval does mitigate the battery life issue, but at the cost of the immediacy of new message notification. Even when TCP sockets become more widely supported, the constant connection required may result in the same affect on battery life. One

possible alternative might be to incorporate an SMS gateway in the Clique Here system, and achieve a push to the mobile client by sending an SMS message to the client handset on a non-standard port. MIDP 2.0 allows MIDlets to request notification of incoming SMS messages arriving on any specified port.

Bibliography

[1] S. Burbeck. Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc). Available at: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html, 1987.

[2] S. Card and T. Moran. User technology–from pointing to pondering. In *Proceedings of the ACM Conference on the history of personal workstations*. ACM.

[3] S. Counts and E. Fellheimer. Supporting social presence through lightweight photo sharing on and off the desktop. In *Proceedings of ACM CHI '04 Conference on Human Factors in Computing Systems*, pages 599–606. ACM, 2004.

[4] K. Go, J. Carroll, and A. Imamiya. Familyware: Communicating with someone you love. In *Proceedings of the IFIP HOIT Conference*. ACM, 2001.

[5] B. Goetz. Exceptional practices, part 2: Use exception chaining to preserve debugging information. Available at: http://www.javaworld.com/javaworld/jw-09-2001/jw-0914-exceptions.html.

[6] R. Grinter and M. Eldridge. y do tngrs luv 2 txt msg. In *Proceedings of the ECSCW '01 European Conference on Computer Supported Cooperative Work*, 2001.

[7] INC InfoTrends Research Group. Worldwide camera phone sales to reach nearly 150 million in 2004, capturing 29 billion digital images. http://www.marketingtechie.com, March 2004.

[8] E. Issacs, A. Walendowski, and D. Ranganthan. Hubbub: A sound-enhanced mobile instant messenger that supports awareness and opportunistic interactions. In *Proceedings of the CHI'02 Conference on Human Factors in Computing Systems*, pages 179–186. ACM, 2002.

[9] K. Kuwabara, T. Watanabe, T. Ohguro, Y. Itoh, and Y. Maeda. Connectedness oriented communication: Fostering a sense of connectedness to augment social relationships. *IPSJ Journal*, 43(11):3270–3279, 2002.

[10] V. Lakshmipathy, C. Schmandt, and N. Marmasse. Talkback: a conversational answering machine. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 41–50. ACM, 2003.

[11] O. Liechti and T. Ichikawa. A digital photography framework enabling affective awareness in home communication. In *Proceedings of the International Workshop on Handheld and Ubiquitous Computing HUC '99.*

[12] P. Markopoulos, J. van Baren, N. Romero, W. A. IJsselsteijn, B. de Ruyter, and B. Farschian. Keeping in touch with the family: Home and away with the astra awareness system. In *Proceedings of the CHI '04 Conference on Computer Supported Cooperative Work*, pages 3–4. ACM, 2004.

[13] N. Marmasse. *Watchme.* pending Ph.D. thesis. 2004.

[14] N. Marmasse, C. Schmandt, and D. Spectre. Watchme: communication and awareness between members of a closely-knit group. To appear in *Proceedings of the Sixth International Conference on Ubiquitous Computing*, Sept. 2004.

[15] A. Milewski and T. Smith. Providing presence cues to telephone users. In *Proceedings of the CSCW 2000 Conference on Computer Supported Cooperative Work*, pages 89–96. ACM, 2000.

[16] E. Mynatt, J. Rowan, A. Jacobs, and S. Craighil. Digital family portraits: Supporting peace of mind for extended family members. In *Proceedings of the CHI*

'01 Conference on Human Factors in Computing Systems, pages 333–340. ACM, 2001.

[17] B. Nardi, S. Whittaker, and E. Brander. Interaction and outeraction: Instant messaging in action. In *Proceedings of the CSCW 2000 Conference on Computer Supported Cooperative Work*, pages 79–88. ACM, 2000.

[18] L. Nelson, S. Bly, and T. Sokoler. Quiet calls: Talking silently on mobile phones. In *Proceedings of ACM CHI'01 Conference on Human Factors in Computing Systems*, pages 174–181, 2001.

[19] Forum Nokia. Optimizing the client/server communication for mobile applications, part 3. Available at: www.forum.nokia.com, October 2003.

[20] Forum Nokia. Efficient midp programming. Available at: www.forum.nokia.com, 2004.

[21] J. Tang, N. Yankelovich, J. Begole, M. VanKleek, F. Li, and J. Bhalodia. Connexus to awarenex: Extending awareness to mobile users. In *Proceedings of the CHI '01 Conference on Human Factors in Computing Systems*, pages 221–228. ACM, 2001.

[22] A. Taylor and R. Harper. Age-old practices in the 'new world': A study of gift-giving between teenage mobile phone users. In *Proceedings of ACM CHI '02 Conference on Human Factors in Computing Systems*, pages 564–571. ACM, 2002.