TattleTrail: An Archiving Voice Chat System for Mobile Users Over Internet Protocol

by

Jang Soo Kim

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 2002

Author ....
Department of Electrical Engineering and Computer Science
May 22, 2002

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

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

TattleTrail: An Archiving Voice Chat System for Mobile Users Over Internet Protocol

by

Jang Soo Kim

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Mobile communication has become a significant part of everyday life. The advent of mobile phones and PDA's has reinforced the necessity for people to be connected without being physically tied down. Communication is key, but current modalities are restricted to simple protocols such as telephony, and less expressive mediums such as text messaging. TattleTrail is an archiving audio chat application for mobile users that attempts to address current limitations in mobile communication by exploring novel communication modalities using voice as a medium over the flexible Internet Protocol (IP) network. TattleTrail acts as an application server within a mobile audio framework that supports different combinations of synchronous and asynchronous communication channels, archiving of chat messages, and new methods of interactively browsing speech. New transitions between synchronous and asynchronous communication, as well as hybrid channels concurrently supporting both modalities, have been explored, which point to new possibilities for mobile communication.

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

Acknowledgments

I have come further than I have ever expected.

I have had an extremely humble beginning: from being exposed to Computer Science (and programming) for the first time as a sophomore at MIT, to always feeling that my late exposure kept me leaps and bounds behind my peers, and then to earning a Masters in that same field. I could not have done this alone without the help of others. I extend deepest thanks to:

First and foremost, my family for always believing in me more than I did. My parents for always demanding my best, and for their hard work dedicated to put me where I am today. My brother, for his unselfish love, for always being a role model, and for his priceless advice about life. My sister, for her unfailing love and encouragement, for her passion for helping others, for her thousands of acts of kindness and caring that I so often take for granted but never should. I love my family, and think they are the best someone could wish to have. I dedicate all of my research work and this thesis to you all.

Chris Schmandt, my advisor, for opening the world of audio and speech interfaces to me, for working consistently with me during my research, for his natural instinct for exploring audio interfaces and communication modalities, and most importantly, for giving me a chance.

Kwan Lee, a former SIG student, for giving me the opportunity to work with him on his thesis that has become the foundation for my research work, for teaching me more than I ever expected over the past year and a half, and for always being willing to give me advice. I would never have been able to join the Speech Interface Group without his help, for which I am grateful.

Fellow SIG students Natalia Marmasse, Stefan Marti, Sean Wheeler, Gerardo Vallejo, Vidya Lakshmipathy, and Ivan Chardin for their support, encouragement, and brainstorming sessions. I would especially like to thank Natalia for her unselfishness, for always helping me in any way possible, and for making my stay at the Media Lab that much more comfortable. And Gerardo for teaching me the wonders of Mexico and the expressive language of Spanish.

Sunil Vemuri for helping me with iPAQ issues.

My close friends at MIT that made my college experience memorable and enjoyable; we all went through hard times, but we survived. Especially Shane Cruz, who took the journey with me through the Computer Science undergraduate and graduate programs at MIT. I know I could not have made it without your help, support, and friendship through the past five years at this school.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile communication has become a significant part of everyday life. The advent of mobile phones and PDA's has reinforced the necessity for people to be connected without being physically tied down to a workstation or desk. Cell phones, in particular, have revolutionized the lifestyles of millions of people by providing audio communication anywhere that a strong enough signal can be obtained.

Another popular technology to keep people connected is the use of online chat systems that provide text messaging capabilities to users in disparate physical locations. Online chat systems generally fall into two categories: instant messaging between two users, or among a group of users. Instant messaging has popularized a new communication protocol that enables users to instantly open communication channels with other users over Internet Protocol (IP), and to manage buddy lists which provide peripheral awareness of other users. Instant messaging users are typically stationary, communicating with each other from their desktop computers. However, cell phones today that support these instant text messaging capabilities are quickly gaining popularity, especially in Europe.

These popular methods of communication involve different media (audio, text) over different networks (Internet, telephone, cell phones). The problem is that traditional telephony provides the rich medium that is desirable in conversational communication, but it is restricted to a simple calling protocol for point-to-point communication that forces a caller to intrude the receiver. This limitation in telephony is largely due to the characteristics of its underlying network. Chat systems, on the other hand, offer greater versatility in modes of communication but use text, which is not as expressive as voice. This flexibility in chat systems results from the capabilities of the Internet as a communications network.

Audio spaces represent another method of communication that involves a rich medium over a network capable of connecting groups of people concurrently. An audio space is an environment that allows a group of users to share a common acoustic space although they are physically separated. This acoustic environment combines the benefits of traditional telephony and online chat systems to enhance communication. But the downside is that audio spaces are usually implemented within stationary environments using a dedicated network to connect specific users to each other.

For mobile users, audio communication is valuable because the eyes and hands are sometimes necessary for other tasks. Therefore, an audio space for a group of mobile users can satisfy the high demand for human connectivity as well as provide a group chat environment similar to online chats and instant messaging, but with a richer medium. TattleTrail is an archiving audio chat application that attempts to address these current limitations found in mobile communication by exploring novel communication modalities using voice as a medium over the flexible IP network. More specifically, TattleTrail acts as an application server within a mobile audio framework to provide a mobile audio space to users, support both synchronous and asynchronous modes of communication, archiving of chat messages, and new methods of interactively browsing speech.

1.1 Mobile Framework

TattleTrail has been built using an existing mobile development framework called Impromptu [13]. Impromptu is a mobile audio framework that supports multiple distributed applications over IP. It enables a mobile device to support multiple audio applications, and gives the user speech and tactile interfaces to browse through and interact with applications.

Some audio applications that Impromptu supports are: live radio, news headlines, MP3 player, telephone, and baby monitor. The user can actively use one application at a time because the mobile device's speaker and microphone are shared resources among all the applications. However, Impromptu provides an alerting mechanism that allows inactive applications to interrupt the current active application for certain events. For example, a user could be listening to music from the MP3 player, but then get interrupted by an alert from the phone application about an incoming call.

Impromptu was designed to be extensible, allowing audio application development for mobile devices to easily integrate into the framework. Therefore, TattleTrail has been successfully integrated into Impromptu, which provides mobile audio communication with a wireless device.

The next section describes an example of how TattleTrail, implemented with the Impromptu framework, could be used.

1.2 TattleTrail Example

TattleTrail is an audio chat system for mobile users over IP. With a wireless device, a TattleTrail user can have instant access to a communication channel for a group of users for an audio chat session. All users within the group are able to hear each other and speak to each other in real-time, thus providing a shared audio space to users in disparate physical locations.

All messages spoken in the chat session are recorded and archived so that users can browse through past conversations. TattleTrail uses audio time-scaling techniques to provide users with the ability to skim through previous recordings at high speeds without altering the audio pitch. When a user leaves the chat and returns later, the user first listens to an "audio history" of the chat messages missed while he/she was not present.

TattleTrail users can also interact passively. A user can leave the chat session, but choose to receive asynchronous alerts whenever there is activity within the actual chat. In this way, passive users wish to remain within the mobile audio space and listen to semi-intrusive alerts in the background.

A possible scenario is described below, illustrating how TattleTrail could be used to facilitate a mobile work force.

1.2.1 Mobilizing and Connecting a Work Force

Alice, Bob, Cindy, and David all work for the local cable company. Bob, Cindy, and David are cable technicians, and drive around the city to perform residential cable installations and repairs. Alice works at the company office, and is in charge of dispatching new installation and repair requests received during the day.

Previously, the technicians each had a CB radio inside the van that was used to communicate with each other and with Alice. If schedule changes were made, Alice would have to contact the technicians by dispatching the changes over radio. The problem was that these mobile technicians could only respond if they were inside their vans at the exact time of the dispatch. This was inefficient, and required Alice to constantly bark into the radio over and over for several new requests. Walkie-talkies were considered, but their range was not large enough to span residential installations in the city and the surrounding suburbs.

Alice realized the communication problem, and the cable company equipped a few teams with mobile devices using the TattleTrail system. One morning, Bob, Cindy, and David were all busy performing new cable installations across the city. Bob was having some trouble setting up the cable in a suburb residence that required different cable settings and channels for five separate televisions. So, while working, he spoke into his mobile device to ask anyone else listening if they knew how to set up the installation. Cindy, who was surveying an apartment residence in the city, responded immediately with a few brief step-by-step directions because she had performed a similar installation the week before. Bob thanked her, and began the installation. Because the messages are archived, Bob was able to listen to an audio history of Cindy's directions as many times as he needed to ensure a correct installation. The time-scaling was helpful so that he could skim quickly through parts of the directions he understood well, and slow down to listen to other steps that were more complicated.

Meanwhile, Alice received a repair request for a residential customer that could only stay home for an hour longer. She broadcast a message to the chat session asking if anyone was available to pick up the new repair request. David was finishing up a new installation at a residence across town. He had originally intended on going straight to his lunch break after he finished, but instead, he replied to Alice and told her he could squeeze that repair request in before lunch.

David serviced the repair request, switched to the background alerting mode in TattleTrail, and went on his lunch break. This mode allows the user to hear asynchronous alerts of the activity in the chat session. During lunch, he heard some activity in the chat, but it sounded like a few technicians were discussing some problems and solutions. A few minutes later, he heard some more activity, but this time it was Alice. He didn't want to devote much attention to the chat, but thought it was nice to have some peripheral awareness of messages broadcast. Once David finished his break, he entered the chat again, and listened to the audio history of what he missed. He skimmed through the technicians talking, and reached Alice's message. She needed someone to perform a new installation in the late afternoon, but all the other technicians said they were booked the rest of the day. David had some free time at the end of the day, so he responded and said he could service the new request.

Later that day, David went to perform the new installation. This residential installation was similar to Bob's earlier that morning. David remembered some of the steps Cindy had explained, but needed to refresh his memory. He quickly listened to the audio history of Cindy's directions and was able to complete the new installation without a problem.

Chapter 2

Related Work

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

2.1 Audio Spaces

> An *audio space* is an audio communication system for a group, the members of which are in disparate physical locations; the audio space creates the auditory illusion for each member that its users share a common acoustic space. [23]

An audio space is a subtype of the traditional media space, which typically provides audio and video streams [3].

Much research work has been devoted to the traditional media space, as shown in [3, 7, 14], but little work has focused purely on connecting separated users in a common audio space. Most previous studies involved stationary spaces in the workplace or school environment where audio and video equipment could be used to monitor a physical space. The mobile nature of this thesis requires wireless transmission of data over IP, which has bandwidth limitations. Streaming audio and video to a mobile device over IP in real-time would cause severe delays because of the rich data medium. It is also important to note that video might not be appropriate at times to a mobile user simply because the eyes might be needed for other tasks. Therefore, audio-only media spaces are examined below.

2.1.1 Thunderwire

The Thunderwire system provided an audio space that connected a group of employees in a work environment [10]. This system hardwired user stations together to provide continuous, high-quality audio to each user. All users had desktop microphones, headphones, and a control box to switch the system to three settings: *on*, *off*, or *listen-only*. When *on*, the user was able to hear all audio recorded by everyone's microphone, and everyone could also hear what was recorded by the user's microphone. *Listen-only* mode shut off the user's microphone, allowing the user to hear everything without being heard by anyone. *Off* was a totally disconnected state. Any user that signed off would miss the conversations within the audio space totally, unless someone that was present relayed the information back at a later time.

Participants could not tell who was presently listening, which they found to be bothersome. This resulted in group norms within the audio space of announcing signing on and off to the system. The synchronous, always-on audio space also led to norms of inattention and withdrawal. More specifically, expressing disinterest in conversation through an audio-only medium required users to slowly utter "fill words" or to simply pause the conversation entirely for several minutes [10]. The participants liked the system, but wished that they could know who was present, and that they could set up private, two-way conversations. Despite some of these problems, Hindus et al found that audio was sufficient for a media space, and that audio spaces could lead to social spaces that were governed by different norms.

2.1.2 Somewire

The Somewire system was a later audio space study that also provided Thunderwire functionality, with different versions offering different graphical user interfaces (GUI) and audio controls [23].

The setup was basically the same as Thunderwire, except some GUI's were created to provide users with additional information about the audio space and the other participants. Some versions of Somewire attempted to present social representations to users, while others tried to show presence.

Another new feature in Somewire was the ability to control the audio output of specific users within the audio space (e.g. volume level, left and right balance). In this way, users had full control of their specific acoustic space; users were able to control who they heard, and who was able to hear them. This gave the participants some privacy within the global audio space.

Somewire provided much more functionality and control over the audio space than Thunderwire, but the results proved that simplicity was highly favored. Specifically, the research results showed that GUI's were poor interfaces to audio spaces, and that user-level control over audio localization (i.e. customizing one's own acoustic space) and other audio attributes was unnecessary. Once again, peripheral awareness was highly desirable.

2.1.3 ConcertTalk

ConcertTalk was a mobile audio space that was used for two days by groups of Lollapalooza concert attendees [25]. The study involved the use of two-way radios that transmitted low quality audio using push-to-talk. This was more of an experimental nature than the previous studies conducted for the Thunderwire and Somewire systems. The important distinction, however, was the motivation behind the ConcertTalk study:

> Rather than look at a media space as a work tool that functioned as a "medium for social activity," the current study found a social activity and inserted an audio space to see what behaviors would emerge. [25]

ConcertTalk results were positive, in that the participants enjoyed the experience and made good use of the system. A notable difference in this study from traditional media space studies was that the audio space was not continuously recording because of the push-to-talk transmissions. Therefore, Strub concluded that the intentionality behind each utterance broadcast gave the participants privacy. The pertinent ideas to be drawn from this study are that media spaces are not necessarily confined to stationary work environments, and that mobile audio spaces in purely social environments can be used to collaborate as well as to socialize.

2.2 Technology and the Mobile Workspace

Technology and the mobile workspace, until recently, has not been a heavily researched area. Traditionally, the majority of studies involving the effect of technology in the workspace have dealt with stationary environments. But, the surge of the Internet and mobile communications popularity has opened new doors to incorporating technology with the mobile workforce.

For mobile workers, the ability to communicate with others in the mobile and stationary workspaces has been found to be a key factor in everyday work [8, 16, 17, 18]. This does not come as a surprise because of the necessity for communication in the work environment and the nature of mobility. Because a central goal of TattleTrail is to provide a communication channel (or audio space) for a group of mobile users, the mobile workspace is an ideal application which must be reviewed.

2.2.1 Field Service Work

Field service work requires technicians from various types of companies to travel to customer locations (business and residential) to perform installations, repairs, etc. These technicians will often work and travel alone, or in pairs, by driving company-owned vehicles equipped with the necessary tools for their specific tasks.

Communication

The Denver Project was a study performed on groups of rural and urban technicians working for a large photocopier manufacturing company [17]. These technicians were given radios that essentially provided them with a mobile audio space for communication.

After six months, the results from the technicians' perspective were overwhelmingly positive. The technicians normally worked alone, but the radios provided a communication channel that joined them into a common workgroup. This provided them with a means to actively cooperate and collaborate, which increased their efficiency. The "accumulated expertise of the entire workgroup" provided support to each individual technician, giving workers and customers greater confidence [17]. Story-telling was found to be an important method of learning for the technicians, which also improved efficiency. When asked if the use of text messages could suffice for communication, the technicians responded that speaking to each other was preferred. One problem that the corporation had with the experiment was that although the radios were helpful, all the lessons about learning that were broadcast over radio could not be saved.

Knowledge Management

Fagrell et al performed a knowledge management study on employees working in the electrical utilities industry [8]. The mobile technicians worked in pairs, and were responsible for installing and maintaining energy equipment. Planners were responsible for scheduling assignments by allocating resources (e.g. personnel, raw materials, vehicles), and administrators worked in the central office managing accounting and communication with customers and planners. The technicians and planners were equipped with cell phones and pagers.

The results revealed that the sharing and passing of knowledge among workers was highly evident in the mobile workspace. For planners, "constant communication is vital because of the need continually to change priorities" [8]. Story-telling, again, was found to play an important role in transferring knowledge within the group. One interesting conclusion made was that diagnosing problems was found to be a "truly collaborative effort" among mobile technicians and planners, usually stemming from immediate problems arising from the mobile context [8].

Using Mobile Devices

Kristoffersen et al explored some of the common problems found in using mobile computers in the mobile workspace for technicians in telecommunications and maritime consulting [12]. In the study, several important observations about the mobile workspace were made:

- The most important tasks for mobile workers are external to operating a mobile computer.
- Hands are often used to operate physical objects, as opposed to the office environment of keyboards, etc.
- Many tasks require high visual attention, as opposed to the office environment where eyes are commonly directed at a computer monitor.
- Mobile workers are often moving around while performing tasks, as opposed to the office environment where workers are typically stationary.

Many of the technicians in this study were given PDA's and small laptops, which were found to be cumbersome at times because they required direct manipulation and visual attention. These requirements are not ideal within many mobile contexts because of the hassle and possible danger of distraction. Sometimes, to perform tasks, the mobile technicians had to "make space" for the mobile devices so that they could be placed down somewhere suitable for direct manipulation [12]. Requiring two hands for input to the mobile device was also a problem. For mobile computing to be effective in a mobile workspace, Kristoffersen et al stated:

> Our claim is that users should not normally have to be engaged in [direct manipulation]... They should not have to "make place" for the device in the mobile situation, but just use it instantly in the situation at hand: it should just "take place."

2.2.2 Mobile Professionals

The work of mobile professionals strongly differs from that of field service workers, yet the necessity for communication remains paramount. Although mobile professionals do not encounter the complications of technical service or repair within unpredictable physical environments, they are displaced from their normal information resources and co-workers when working away from the office. According to Perry et al, one of the motivations of mobile technologies is to bridge this gap between the many resources available in a normal office space and those available while working in an unfamiliar mobile environment [18].

A characteristic problem with the mobile environment is there is "less predictability and more restricted access to information" [18]. E-mail affords a communication channel to mobile workers, but its asynchronous nature is sometimes unsuitable for situations that call for immediate attention or direct conversation. Because the mobile phone provides verbal communication capabilities and nearly ubiquitous access, it has become the most prominent technology used by mobile professionals [16].

Mobile phones provide instant access to colleagues when explicit information retrieval is necessary. This usage of mobile phones was very frequent with mobile professionals, and usually required note-taking during the actual phone call [16]. This simple observation exposes a practical problem for mobile workers that need to retain the information heard over the phone, yet are busy driving or have limited resources for note-taking (one particular participant in the study had to scribble notes on a newspaper to capture main ideas during the phone call). O'Hara et al noted that enhanced mobile phone capabilities allowing the recording of audio snippets from phone calls could facilitate mobile professionals with information retrieval.

Besides explicit information retrieval, mobile phones were also found to be useful for providing a mobile worker with a level of remote awareness of developments at the office and to maintain social connections with co-workers. In particular, many mobile professionals would phone the office at convenient times just to remain up to date with office proceedings and stay informed about general issues that might be pertinent to their work [18].

2.3 Instant Messaging

Instant messaging has emerged has a popular communication modality that allows users to chat in real-time over IP, as well as organize "buddy lists" of close associates to maintain a peripheral awareness of their chat activity. Most instant messaging use occurs in stationary environments (e.g. home, school, or office). This type of communication is only similar to TattleTrail in that users can chat with each other or with a larger group over IP asynchronously. However, a mobile instant messenger (IM) called Hubbub has recently been developed which has more in common with TattleTrail, and is described below.

2.3.1 Hubbub

Hubbub is an instant messaging application that runs on PDA's and attempts to use sound to support awareness and opportunistic interactions [11]. Hubbub provides the standard instant messaging functionality, but also uses sound to send *Sound Instant Messages* (SIM's), which are earcons with associated meanings such as "Hi" or "Thanks." The major differences between Hubbub and other IM's is the presence of an activity meter for each user in the buddy list, the type of device each buddy is using, and possibly a short snippet from the buddy describing his/her location. Users are also given the ability to log on at multiple places or devices, so the buddy list could keep track of where the buddy was located. In this way, users' idle states were calculated, and peripheral awareness of their activity could be obtained. SIM's were used to indicate buddy activity states, as well as for sending instant messages.

Hubbub shares some traits with TattleTrail in that audio was used in a mobile environment to enhance the user interface, and present some sort of audio space. Many users of the Hubbub system enjoyed the auditory cues which helped provide a sense of awareness. Some participants commented that the sounds gave them a closer connection to others located on the opposite coast, while others said the audio alerts gave them a sense of feeling for the group.

2.4 Skimming Audio Documents

TattleTrail saves every message that is broadcast to the chat in order to allow users to browse through the message archives. This functionality is necessary to provide users with a means to listen to messages they may have missed in order to "catch-up" with the current chat context before actually joining the live chat.

If browsing through chat messages was simply performed at normal playback speed, listening to numerous messages would be time consuming, especially if particular messages were not of importance to the browsing party. To enhance the user interface, TattleTrail performs speech processing techniques for skimming recorded chat messages. Therefore, techniques for skimming audio documents are examined below.

2.4.1 SpeechSkimmer

SpeechSkimmer was a user interface for interactively browsing through speech recordings at different speeds [2]. A user could rapidly skim recorded speech through an interactive touchpad supporting skimming both forward and backward at different levels.

The system used the *synchronized overlap-add algorithm* for time-scale modification to compress speech at varying rates, while applying techniques to partition recordings into salient audio segments. The heuristic used to segment the audio relied upon the premise that long, silent pauses in speech usually indicate a new topic or a new speaker. Once these pauses were detected, the SpeechSkimmer would jump to segments (backward and forward) and play only segments of audio following a long pause. This special pause jumping heuristic was applied only at the fastest level of compression. SpeechSkimmer performed normal time-scale compression with pause removal techniques at slower speeds.

The next section describes audio indexing techniques used to skim audio documents in more detail.

2.4.2 Audio Indexing Techniques

The previous section explored rapid interactive skimming of speech, with some heuristic for segmenting recordings to aid in browsing. This section focuses upon audio indexing techniques for structuring audio in order to facilitate browsing.

NewsComm

NewsComm was a system that delivered structured audio of personalized news and other program material to mobile users [20]. Because the system handled a large database of audio files containing speech, techniques were used to provide rapid browsing of the structured audio.

Like SpeechSkimmer, NewsComm detected long pauses as points of possible interest. Other audio structuring techniques were used to map "jump locations" within the recording. Speaker differentiation algorithms were used to allow users to jump to points at which speaker changes occurred. By combining multiple algorithms to analyze audio, NewsComm was able to deliver highly structured audio to the mobile device, which provided the user interface for navigating through the recordings. It is important to note that NewsComm required much of the audio processing to be performed on a server before the user could download the structured audio to the mobile device.

Dynomite

Dynomite was a portable electronic notebook that was used for taking handwritten and audio notes, which used computational resources to organize and search the notes for easy retrieval [27]. Audio could be recorded synchronously using Dynomite, which would manage the audio data. Possible user interface interactions were explored to see how audio indexing by speaker identification could facilitate the audio note-taking process.

The speaker identification algorithm explored was previously developed by Wilcox et al [26]. When speakers are known *a priori*, real-time audio indexing could be performed for recordings of multiple speakers using hidden Markov model networks to represent individual speakers and their interconnections. However, without knowledge of speakers beforehand, real-time segmentation could not be performed.

Audio Notebook

Audio Notebook was another device used to facilitate pen and paper note-taking by recording and structuring audio based upon audio structuring techniques coupled with the user's activity while taking notes [24]. The user was not required to explicitly mark important points during the recording to aid the segmentation of audio; natural note-taking activity including page turns were used as indices into the recording. This approach differs from the previous audio structuring works because of the additional information retrieved from the user in real-time.

Stifelman also used pause and pitch detection to further segment audio recordings at points of interest. In addition, Audio Notebook provided algorithms to find phrase boundaries within the recorded audio as another heuristic for segmentation. Users were then able to interactively browse through recordings and jump to locations of possible importance or interest when reviewing their "audio" notes.

2.5 Impromptu Chat

The initial prototype that led to the creation of TattleTrail was implemented within the Impromptu framework as a simple audio chat application. This application was totally asynchronous; no messages were ever heard in real-time. When a user joined the chat application, all other users currently in the chat were notified by a door opening sound, followed by the user's distinct audio icon (usually a recording of the user's name).

Any chat message sent was recorded to the chat server, which is similar to TattleTrail's model. However, there was no floor control protocol involved because messages were played individually to each user asynchronously. Once a user finished recording a message, the chat application would then play that message to every other user in the chat. This eventually led to some synchronization problems, because people could record messages at the same time, effectively speaking concurrently without knowing it. Therefore, many messages recorded were unnecessary, or out of place due to the recordings sent by other users at around the same time.

Browsing of messages was also supported in this chat prototype. The archive was maintained in a similar list, but no special browsing techniques such as time-scale modification were provided. To listen to a message, the user had to press the *up* or *down* button to start playing each archived message.

Another difference was that the prototype made no distinction between modes of interaction. TattleTrail provides browsing modes, chatting modes, and alerting modes. The original prototype just had one mode of interaction, and therefore had to deal with problems such as listening to a previously recorded message while another user just finished recording a new message. In this case, an alert was played to notify the user of a new message, and then the user immediately heard the new message once listening to the previously recorded message was finished.

The poor user interface for chatting and browsing audio, the interleaving of messages, redundant messages, and weak alerting mechanisms led to disappointment with the chat application. These weaknesses in the original prototype provided motivation to rethink and redesign the mobile audio chat application into the current version, TattleTrail.

2.6 Summary

The studies reviewed in this chapter indicate that audio spaces are suitable media spaces, and that audio spaces provide a good communication channel for a group of physically separated users. Not much work has been done within the mobile audio space, but the studies examined give promising results, namely the high activity in ConcertTalk and the sense of closeness afforded by sounds in Hubbub. Instant messaging applications such as Hubbub attempt to provide a rich user experience in a mobiles setting, but are still limited by text. Chalfonte et al found text as a medium to fall short of the expressive richness of voice, which was preferred when commenting about higher level issues [4].

Research results show that communication is key within a mobile workspace, so a mobile audio chat system has promising application within this field. As reported by Kristoffersen et al [12], most mobile work situations studied could have been enhanced if the user had audio feedback from their mobile computers. Many situations in the mobile environment are not conducive to full visual attention, so audio-only applications like TattleTrail which provide instant communication and passive alerting could be helpful.

According to Fagrell et al [8], "knowledge management in mobile settings is *social* and *dynamic*." This further supports the applicability of a mobile audio chat system to facilitate knowledge management, due to the highly social and dynamic characteristics of chat.

In the mobile professional study by Perry et al, a participant remarked that "staying in touch is important, both from a management of the workload perspective, to make sure there's not this enormous pile, and secondly to be in touch if there is anything that comes up that needs a quick response" [18]. The passive alerting channels provided by TattleTrail could again serve useful purposes for mobile workers with peripheral awareness of the events back at the office. Perry et al conclude that "work becomes explicitly collaborative as the mobile worker attempts to coordinate events in their local environment with remotely accessed resources. Talk is central to the mobile work described..." [18]

Chapter 3

User Interaction

This chapter describes the user interaction experience of TattleTrail. TattleTrail has been designed and developed to provide mobile users with novel communication modalities and audio processing techniques by using a flexible application server framework over mobile IP. Users are not limited by location, less expressive media such as text, or strict communication protocols of phone networks. Because it is integrated into an audio-only framework, much consideration was taken in defining the audio interface to TattleTrail as well.

A Compaq iPAQ is used as the mobile client device through which all user interaction occurs (Figure 3-1). The iPAQ supports full duplex audio, with a microphone in the top left corner, a headphone jack (necessary to prevent feedback from the speaker), and several buttons. Currently, the screen is not used for TattleTrail because Impromptu is an audio-only framework.

All of the buttons on the iPAQ are used for sending user input events to the necessary components in the Impromptu framework. The *push-to-talk* button is used only for sending speech commands to the recognition server. Any time this functionality is used, all audio to be played is muted. The *record* button is used to record audio for applications. This button acts in push-to-talk mode as well. The *function* button is application-specific; it performs different functions depending upon the current application being used. The wheel at the center bottom of the iPAQ provides *up*, *down*, *left*, and *right* button press events. The *up* and *down* buttons are application-specific,

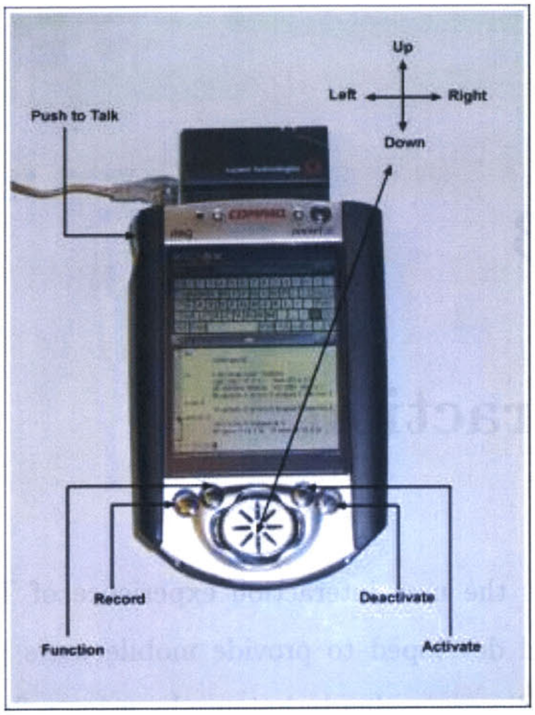


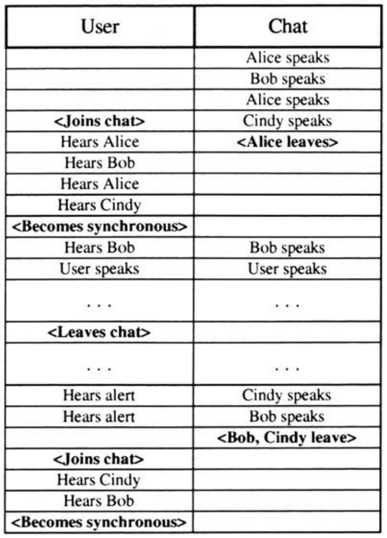
Figure 3-1: The Compaq iPAQ with 802.11b wireless LAN card.

usually used to browse through application content. The *left* and *right* buttons are global browsing buttons that allow the user to switch between applications. The *activate* and *deactivate* buttons are used to activate and deactivate applications.

When TattleTrail is activated, its audio icon is played on the client device.[1] This audio icon representing the TattleTrail application fades in musical notes mixed together with giggling children. Multiple applications are supported on the mobile client device, so the application currently being used is known as the *active* application. When TattleTrail is the active application, two modes of user interaction are available: **Catch-up Mode** and **Chat Mode**. Users enter Catch-up Mode when TattleTrail is activated; this mode allows browsing through previous or missed chat messages to "catch-up" to the current activity in the chat room.

When finished browsing messages, the user enters the synchronous Chat Mode,

[1] All Impromptu applications have a distinct audio icon which is played to alert the user of activation or an event triggered by that application.



User	Chat
	Alice speaks
	Bob speaks
	Alice speaks
<Joins chat>	Cindy speaks
Hears Alice	**<Alice leaves>**
Hears Bob	
Hears Alice	
Hears Cindy	
<Becomes synchronous>	
Hears Bob	Bob speaks
User speaks	User speaks
. . .	. . .
<Leaves chat>	
. . .	. . .
Hears alert	Cindy speaks
Hears alert	Bob speaks
	<Bob, Cindy leave>
<Joins chat>	
Hears Cindy	
Hears Bob	
<Becomes synchronous>	

Figure 3-2: Sample TattleTrail activity. Note the transitions between synchronous and asynchronous participation.

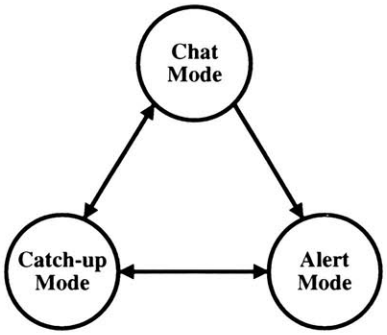


Figure 3-3: Possible mode transitions in TattleTrail.

which uses push-to-talk similar to Nextel's Direct Connect system.[2] The push-to-talk functionality is supported by a floor control protocol that allows only one user to speak at a time; all other users in Chat Mode hear the speaker in real-time over IP. Any message spoken during Chat Mode is saved on the server for all users browsing archived messages in Catch-up Mode.

A third mode of interaction is possible when the user has *deactivated* the application, called **Alert Mode**. Deactivation, within the Impromptu context, is a result of either switching to another application, or placing the mobile device in an idle state.[3] Alert mode provides the user with peripheral awareness of chat activity at different levels of awareness and participation.

Sample TattleTrail chat activity within different modes of participation is shown in Figure 3-2. The user may switch back and forth between the asynchronous Catch-up Mode and the synchronous Chat Mode at any time via the speech recognition commands "history" and "chat." To enter Alert Mode, the user must explicitly

[2] Nextel's Direct Connect gives mobile phones a virtual walkie-talkie functionality with floor control.

[3] In Impromptu, applications are never "closed." Their resources are always available while they remain online.

deactivate the application with the commands "passive" or "background." These will be explained in more detail in later in the chapter. The possible transitions between modes can be seen in Figure 3-3.

The following sections describe the modes of interaction in more detail. The first section examines Catch-up Mode and the different audio processing techniques available to use asynchronously. The second section details the synchronous Chat Mode. Finally, the user interactions supported by the Alert Mode are explained in the last section.

3.1 Catch-up Mode: Asynchronous Audio History

When entering the TattleTrail application, users are placed into the asynchronous Catch-up Mode, where archived messages are streamed to the user to present an "audio history" of the chat. If the user is activating the application for the first time, the audio history is started at the first message in the archive. The archive currently stores every message recorded, but for practical use the active archive would be limited by available hard disk space set aside for storage. If the user is returning to the application, then the audio history starts a few messages before the point when the user last exited the application.

Catch-up Mode allows the user to interactively browse through the audio recordings of the chat. Simple playback at normal speeds would provide a poor user interface to skimming the audio messages because of the time required to listen to each recording. It would be advantageous to be able to skim through voice messages just as if skimming through text messages. TattleTrail uses time-scaling techniques to provide this ability to rapidly "skim" chat messages at rates of up to six times normal speed.

The turn-taking protocol enforced by the push-to-talk floor control of Chat Mode allows TattleTrail to group recorded messages into "chat clusters." Chat clusters are determined by the pauses between subsequent chat messages. Currently, all messages recorded within 30 seconds of each other are grouped together into a single cluster. The motivation for this grouping of messages is that messages recorded closely together are most likely related, and represent an actual conversation among the chat participants. Chatting channels typically have bursty interactions separated by long periods of silence. These bursts of activity within the chat are likely messages sent in response to other recent ones.

When browsing through the audio history, the chat clusters help to give the user a perception of the traffic on the chat channel. In particular, finding long breaks in the chat recordings convey the low traffic within the chat. These long breaks in conversation are represented by a clock ticking sound that is played to the Impromptu client to indicate the passage of time. Therefore, when a user is browsing through messages and traverses to the "next" or "previous" chat cluster, the clock ticking audio is heard briefly before the new message is heard.

Upon entering Catch-up Mode, the streaming of the audio history to the user begins. Each message is played in order at normal speed. In this mode, the user can press the *up*, *down*, or *function* button to interactively control the playback speed and direction (i.e. forward or backward). Generally, the *up* button will increase speed, the *down* button will decrease speed, and the *function* button jumps back to the previous message.

When playing a message at normal speed, pressing the *up* button increases the speed by performing time-scale compression. If the button is pressed and held down, the speed is gradually increased without stopping compression. To jump to the maximum speed, the user can double-click *up*. To slow down the playback speed, the user can press the *down* button. Again, if the *down* button is pressed and held, the speed will gradually decrease.

If the user wishes to decrease speed when the playback speed is normal, TattleTrail begins to play the messages "backward." The archived recording is not played backward literally, which would not be intelligible. Instead, it plays 4 second chunks of audio at the user's desired speed, but in reverse order. Therefore, 4 second bits of audio are heard, then TattleTrail jumps back 4 seconds to play the next chunk. This technique is similar to the one implemented by SpeechSkimmer [2]. When playing backward, and the speed is decreased, the rate of compression is actually increased, which will allow the user to play backward faster. To jump to maximum speed going backward, the user can double-click *down*.

Double-clicking *up* or *down* typically jumps to the maximum speed of time-scale compression in the respective direction. However, if the user is listening to messages forward, and double-clicks *down*, playback automatically changes directions and begins playing at normal speed backward. Likewise, when listening to messages going backward, a double-click *up* changes directions and begins playing forward at normal speed.

If the user changes the direction of playback at any time, an audio cue (sounding like a computer blip) is played to notify the user of the change. Originally, no sound was streamed to the user, which sometimes caused confusion as to exactly when playback switched directions. If the user is listening to the first message stored in the archive, and attempts to move backward, a short "boing" springing sound is played to alert the user that he/she is at the beginning. This sound represents "bouncing off the beginning of the history" because normal forward playback is resumed automatically.

The *function* button can be used to jump to the previous message and begin playing forward at normal speed. This is useful to stop and listen to messages when browsing forward or backward at high speeds. If the user double-clicks *function*, then TattleTrail jumps to the first message of the cluster and begins playing forward at normal speed. If the user is currently at the beginning of the cluster, a jump is made to the first message of the previous cluster.

The time-scale compression techniques to this point have been strictly linear. The actual rate of compression stays constant once the user increases or decreases the speed. Having the ability to browse voice messages at faster speeds is especially valuable when skimming conversations for content is desired. However, once the time-scaling speed is increased too much, all intelligibility of the message content may be lost. The next section describes how TattleTrail attempts to solve this problem involving high speed browsing.

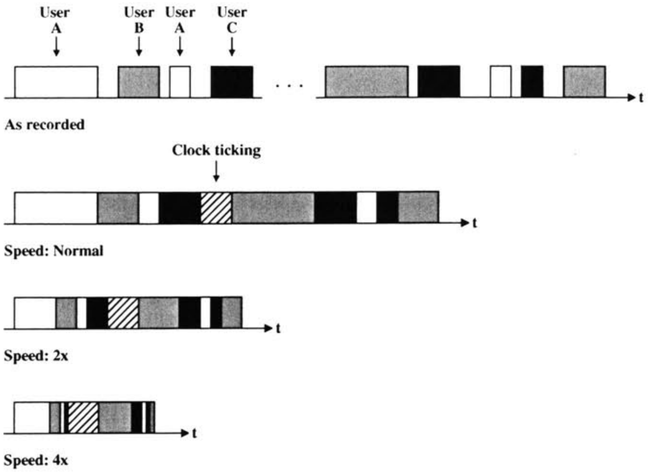


Figure 3-4: High speed browsing. Two different clusters are shown above. When browsing, clusters are separated by a clock ticking sound. Note the non-linear scaling at speed 4x.

3.1.1 High Speed Browsing

TattleTrail uses non-linear time-scaling when users are browsing forward through chat messages at rates greater than twice normal speed. Once speeds surpass approximately twice normal speed, time-scale compressed speech begins to lose intelligibility rapidly. Simply skimming through messages at a static high speed would not convey the content of messages appropriately. Therefore, the non-linear time-scaling used in TattleTrail attempts to provide users with the ability to rapidly browse chat messages while gaining some understanding of the content of the messages.

The non-linear technique compresses messages at varying speeds depending upon its particular position in the chat cluster. As seen in Figure 3-4, the beginning of the cluster is played slowly, and the end of the cluster is played extremely quickly. This technique relies upon the assumption that the beginning of the cluster usually sets the topic for the conversation.

The playback starts slowly at the beginning of each chat cluster at 1.5 times normal speed. After a few seconds, the speed is gradually increased until it reaches a maximum speed of 6 times normal. This non-linear scaling of audio provides the user with an effective speed of compression that is approximately equal to the user's desired speed. The maximum desired speed for users is 4 times normal, so this effective speed can be reached by playing unusually slow to start, and playing unusually fast to end.

3.2 Chat Mode: Synchronous Chatting

Once users are finished with Catch-up Mode, they enter the synchronous Chat Mode. Chat Mode is the simplest mode of TattleTrail, in terms of usage and implementation. In this mode, users speak to each other in real-time via IP. A floor control protocol is used to allow only one user to speak at a time, or have "control of the floor." This protocol is useful for allowing multiple networked users to share the same audio channel in a constructive manner [6].

When a user wishes to speak, he/she simply presses and holds the *record* button, which is push-to-talk. If control of the floor is granted, a short beep is heard, and the user can begin speaking into the iPAQ microphone. At this point, all other users in Chat Mode will be able to hear the speaker synchronously. The recording stops once the users releases the *record* button, and the message gets archived.

If the floor control request was rejected, the user hears a sharp chord that slowly fades away. If the user holds down the *record* button, the floor will be granted when available, in the order the floor request was received by the TattleTrail server. When granted, the same short beep is heard, which signifies control of the floor. Any message broadcast to the chat during this "holding" period will still be heard. The user may release the button at any time, but in doing so, will not be considered for the floor when it becomes available.

3.3 Alert Mode: Peripheral Awareness

The Alert Mode for TattleTrail, contrary to the other modes of interaction, is only entered upon explicit deactivation of the application. This mode is for passive users who do not wish to devote all of their attention or audio focus to the application, yet still wish to remain within the TattleTrail audio space by receiving alerts. These alerts are of recently recorded messages from the users in Chat Mode, and are sent to users in Alert Mode asynchronously. Users also have the option, of course, to simply deactivate TattleTrail without explicitly setting a state of alert. This would follow normal Impromptu deactivation, which results in the user not receiving any type of alert from TattleTrail.

For those users in Alert Mode, the delivery of TattleTrail alerts strays from the Impromptu alerting paradigm. Normally, an alert in Impromptu is overly intrusive; all current audio being played or recorded on the client device is muted while the alert is played. Once alerted, the user can specifically activate the alerting application within 10 seconds by pressing the *activate* button.[4] TattleTrail, however, attempts to improve the alerting mechanism by delivering less intrusive alerts catered to the user's desired attention level, and allowing passive interaction with the application although in a deactivated state.

The Alert Mode supports two different levels of attention, Passive and Background, which have different levels of intrusion. Passive Alert Mode is discussed in the next section.

3.3.1 Passive Alert Mode

The Passive Alert Mode has a higher level of attention and intrusion than the Background Alert Mode. While in the passive state, the user is allowed to freely activate and use other applications in Impromptu. Whenever a new message is finished recording in Chat Mode, it is broadcast to all passive users.

When the alert is received by the client device, the audio for the current application is faded down. The first half of the TattleTrail audio icon is faded up briefly to indicate to the user that the following alert is from the TattleTrail application. The audio icon fades out and is immediately followed by the newly recorded chat message. Once the alert is played, the muted application audio fades back up to normal audio levels.

This level of alert is semi-intrusive in that the active application's audio is muted

[4] This use of the *activate* button only applies when specified by the alerting application.

briefly, but it is not sharply interrupted as normal Impromptu alerts are. The semi-intrusive behavior is indicative of the user's desired state of attention. In this case, the user wishes to be able to use other Impromptu applications such as the Radio or Music application, but still has a particular interest in hearing TattleTrail chat activity.

Passive users are also allowed to actively participate in the chat, although they are not in Chat Mode. This is another significant change to the Impromptu paradigm of application interaction, which typically allows the user to only interact with the active application. In the passive case, the user may press the *record* button to attempt to gain floor control just as if in Chat Mode. This will mute any audio being played by the active application to signify to the user that recording is possible. All interactions in this state when the *record* button is used follow the Chat Mode behavior. If floor control is granted to a passive user, the user's audio is broadcast to others in the chat synchronously, and is recorded as well. When the passive user releases the *record* button, the active application audio begins playing again. Passive users do not receive asynchronous alerts of their own messages.

The Passive Alert Mode, therefore, provides a hybrid communication modality through a channel that is both synchronous and asynchronous. A subtle line is drawn between the two; passive users receive asynchronous alerts of recently recorded messages, but can speak to others in Chat Mode synchronously. This subtlety helps define the passive state in TattleTrail, where a passive user gains peripheral awareness of chat activity through semi-intrusive alerts, yet can send "intrusive" messages to all Chat Mode users. All Chat Mode users are, of course, actively engaged and desire this intrusive level of interaction.

Some TattleTrail users may not wish to have such a high level of attention to chat activity as provided by the Passive Alert Mode. For those users, the Background Alert Mode is more appropriate, which is described in the following section.

3.3.2 Background Alert Mode

The Background Alert Mode also allows the user to freely navigate and activate other Impromptu applications. Similar to the passive state, alerts of recently recorded chat messages are sent asynchronously to the background user, with the same format of a leading TattleTrail audio icon followed by the chat message itself.

When the alert is received, however, it is treated differently than in the passive state. The background state is less intrusive in nature, so alerts received are mixed in with the current application's audio at an attenuated level. Therefore, the background user's application audio is not interrupted or faded out at any time; the TattleTrail alert is simply mixed in together at a low audio level. This level of attention differs from both the passive state and the Impromptu alerting paradigm as well.

The Background Alert Mode does not support active participation in the TattleTrail chat. Users only lurk in the background, which reflects their desire to gain peripheral awareness of the chat activity without being intruded upon in a destructive manner.

The following section explores the mode transitions in TattleTrail.

3.4 Switching Between Modes

The previous sections of this chapter gave detailed descriptions of each mode of interaction. TattleTrail supports synchronous, asynchronous, and hybrid channels of communication between mobile users. This final section examines how they fit together through user interactions.

Asynchronous Catch-up Mode is first entered upon activation of TattleTrail. When a user transitions to Chat Mode, a stereo chime consisting of two tones that fade from left to right is played. This sound is reflective of the transition being made from an asynchronous channel to a synchronous channel. A user can make this switch after reaching the end of the audio history, or by speaking the command "chat."

When in Chat Mode, the user can switch back to Catch-up mode by speaking the command "history." A different chime, also consisting of two tones, is played to

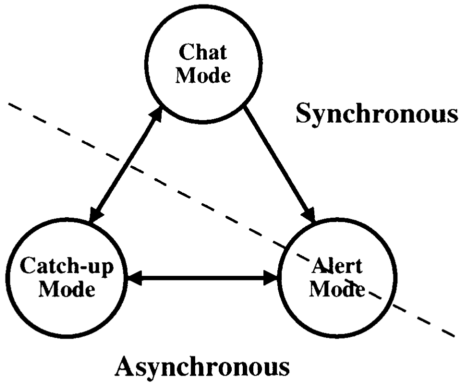


Figure 3-5: The synchronous and asynchronous boundary between different modes of interaction in TattleTrail.

represent this transition from synchronous to asynchronous modes. The first recording played in the audio history will be the first message of the chat cluster that precedes the most recent cluster in the archive. This is based on the assumption that if a Chat Mode user wishes to listen to the audio history, he/she most likely desires to start at a message at least a cluster before the current message.

At any time that TattleTrail is the active application, the user may switch to one of the two Alert Modes by speaking "passive" or "background." This explicitly deactivates TattleTrail and sets the user's alert state. No audio cue is played for this transition because Impromptu has a strict activation/deactivation audio protocol. An application being activated has its audio icon played, while direct deactivation puts the Impromptu client in an idle state signified by an audio icon of a man yawning. Therefore, playing an audio cue directly before the yawning sound would lead to some confusion, as excessive audio input can become destructive. Instead, the yawning snippet can be used as feedback to the user that his/her "passive" or "background" command was correctly recognized.

Once the user has slipped into Alert Mode for the first time, the specific state (passive or background) is saved by TattleTrail as a preference. Thereafter, when the user deactivates TattleTrail, he/she will automatically be placed into that same state of alert. This is to simplify user interaction when deactivating the application by not requiring an explicit "passive" or "background" command each time, assuming that the user wishes to remain in that state of alert whenever TattleTrail is not the active application. If the user wishes to change this, an explicit command must be used the next time TattleTrail is deactivated. To avoid the Alert Mode altogether, the user may speak the command "kill" to deactivate the application and receive no alerts.[5]

A diagram of the possible transitions within TattleTrail is again shown in Figure 3-5, but with the synchronous and asynchronous boundary displayed.

[5] Before a user activates TattleTrail for the first time, he/she is in an inactive alert state by default.

Chapter 4

Design and Implementation

This chapter describes the software architecture of TattleTrail. The chat system essentially operates as an application server that provides audio content and services for multiple remote clients. All TattleTrail components are implemented in C++ and run on Linux.

TattleTrail performs substantial audio processing and management, streams audio to and from clients over IP, and handles client control messages (i.e. user input via speech recognition and button press events). Therefore, performance and scalability were heavily considered during the design and implementation.

The rest of the chapter is divided into four sections. The first section explains the Impromptu framework in more detail, and how TattleTrail fits in. The second section discusses the general TattleTrail design. The third section describes the catch-up mode implementation. The fourth section details the design of the synchronous chat functionality. The fifth section describes the passive awareness and alerting mechanism.

4.1 Impromptu Framework

Impromptu is a mobile, IP-based framework for an audio device, providing support for multiple audio applications and speech processing services such as speech recognition and speech synthesis. It is an audio-only platform composed of distributed

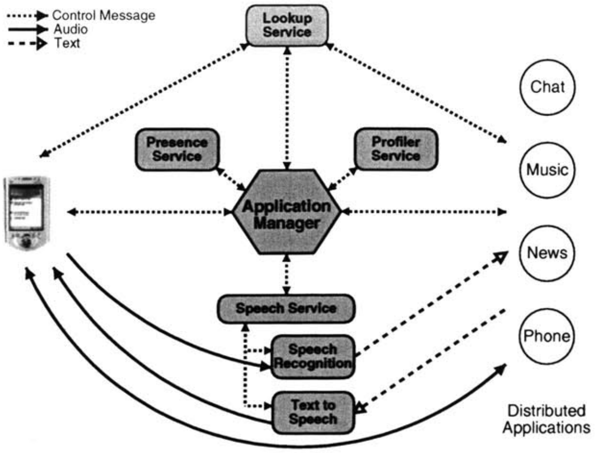


Figure 4-1: The Impromptu architecture. Note that different types of connections are made for audio, text, and control channels.

components over the IP network that interact together to service mobile clients. The distributed components allow the mobile device to offload processor-intensive applications and services, giving the user the ability to manage and operate several applications in a mobile environment.

As seen in the architecture diagram in Figure 4-1, TattleTrail (labeled "Chat") acts as a distributed application component within the Impromptu framework. TattleTrail maintains audio socket connections to each mobile client device, control socket connections to the Application Manager, and text socket connections to receive speech recognition output for each client. A direct connection to the client device is necessary for audio content in order to minimize network delay. Control messages from the client, however, are channeled through the Impromptu Application Manager. Speech recognition is used heavily by Impromptu clients, so some of TattleTrail's functionality can be managed through voice commands that are processed by the speech recognition component and output to the application. More information about the Impromptu architecture and design decisions can be found in [13].

Impromptu currently supports different applications that work effectively within an 802.11b wireless network. In addition, the author joined the Impromptu project when development efforts first began. Therefore, implementing TattleTrail within this framework has given further proof of the architecture, as well as provided a mobile audio development platform for rapid implementation and integration.

The following section describes the Impromptu application development platform.

4.1.1 Application Servers

Applications within the Impromptu framework act as application servers that provide a wide range of audio services for a mobile client device. But, each application must follow Impromptu connection and control protocols in order to integrate into the architecture and behave properly. Therefore, much of the necessary initialization and protocol logic was carefully embedded into generic super classes and utility classes in order to allow application developers to concentrate on implementing application-specific functionality. This application development platform facilitated rapid prototyping and implementation of new applications, as well as stand-alone applications that have been transported into the Impromptu framework.

TattleTrail has been implemented using the Impromptu application development platform. Without additional work, this gave TattleTrail logic to support multiple audio connection management, user control processing, speech recognition capabilities, and existing Impromptu alerting mechanisms.

A brief overview of the Impromptu client, and its interactions with applications, is discussed in the next section.

4.1.2 Mobile Client

Impromptu clients are run on Compaq iPAQ's running Linux (Familiar v0.5.1 distribution) with an Intel StrongARM processor and a standard 802.11b wireless network card. The client software manages multiple audio connections to different applications and speech services, and sends all control messages to the Application Manager, a central communication hub (Figure 4-1). The software accepts user input through button presses and voice commands. Button press events are sent to the Application Manager to be routed to the intended distributed component, while voice commands are streamed to a speech recognition server to be processed and then output to the appropriate component as well. Buttons and voice commands are used to switch between applications, and to perform application-specific functions (e.g. jumping to a random song in the Music application's play list).

In addition to detecting user events, the Impromptu client has an existing alerting mechanism for receiving events triggered by applications or other Impromptu clients. For example, a user could be listening to MP3's through the Music application, and then suddenly receive an Impromptu phone call from another user. The Music application, known as the *active* application in this context because it currently has control of the device's audio focus (i.e. microphone and speaker), is then interrupted so that the user can be notified of the phone call event. The MP3 playing on the device temporarily stops so that an audio alert specific to the Phone application can be played.

This alerting mechanism always forces the active application's audio to be completely stopped until the alert is finished playing. The new, additional alerting mechanism introduced by TattleTrail will be discussed in a later section. The next section examines the TattleTrail application design, which was built upon the Impromptu application development platform.

4.2 The TattleTrail Application

TattleTrail is implemented as a multi-threaded application server that is integrated into the Impromptu framework. Parts of the `TattleTrail` application implementation are tailored to follow certain Impromptu protocols, such as control messages, establishing socket connections, registering application information, etc., and are described in more detail in [13].

When the `TattleTrail` application starts up, several threads are spawned. The `RecordThread` and `AlertThread` are created, which handle the recording and alerting of chat messages. In-depth explanations of these processes follow in later sections of this chapter.

After the application has registered its status within the Impromptu framework, the ControlThread is created. This thread handles all control messages that are sent to the application, including user input and general Impromptu-specific commands or data. The ControlThread manages multiple Transmission Control Protocol (TCP) sockets for receiving control messages, including one socket for each Impromptu user subscribed to the TattleTrail application. The ControlThread receives user input, such as button press events, and processes them accordingly.

The next section describes the TattleTrailUser class, which represents a single user subscribed to the application.

4.2.1 The TattleTrailUser Class

The TattleTrail server receives user information for each subscribed Impromptu user that joins the Impromptu network. From this information, a TattleTrailUser object is created when the user first activates the TattleTrail application.

The TattleTrailUser object stores user-specific data. This consists of the user's Impromptu identification, network address, and speech information. Other information is stored, which will be explained later in the chapter. TattleTrailUser also contains thread synchronization routines in order to control each user's SpeechThread.

The TattleTrailUser object establishes a socket connection with the speech recognition server running within the Impromptu framework upon a user's first activation of the application. A new SpeechThread is then spawned, one for each user. The SpeechThread simply listens for any commands recognized by the speech recognition server for the particular user that are sent to the application. Any speech command received in this thread is processed, which is similar to the ControlThread processing of button press events. TattleTrailUser only allows the user's SpeechThread to run when the user is active by waking and sleeping the thread.

The next section describes the Catch-up Mode implementation that provides an audio history with novel browsing techniques for chat users.

4.3 Asynchronous Audio History

The audio history functionality is the most processor-intensive and complicated component within TattleTrail. The audio history essentially provides a mobile TattleTrail user with rapid, interactive audio browsing capabilities of previously recorded chat messages in real-time. This includes time-scale modification of speech on-the-fly to allow for users to speed up or slow down, non-linear time-scaling for high speed browsing, and the ability to "play backward."

When the TattleTrail application gets activated, the user immediately enters Catch-up Mode, which presents the audio history of the chat messages. When a user enters, the `TattleTrail` application object spawns a new `HistoryThread`, which handles all audio history processing for the user. The thread is destroyed once the user exits Catch-up Mode, which frees server resources.

Once the `HistoryThread` starts, a new TCP socket connection is negotiated with the Impromptu client device as a channel to stream audio history content. Although TCP sockets can introduce unwanted network delay, ensuring the delivery of all audio packets proved to be more important because the intelligibility of time-compressed speech sharply dropped when a User Datagram Protocol (UDP) socket connection was used.[1] In addition, Catch-up Mode is asynchronous, so minimal network delay was not an issue. This is the only instance in TattleTrail where TCP sockets are used as an audio channel.

Once the TCP connection is established, the `HistoryThread` instantiates a new `AudioHistory` object. After `AudioHistory` has initialized, the `HistoryThread` begins listening to user input via the `TattleTrailUser` object maintained by the `TattleTrail` application object. As previously described, the `TattleTrailUser` class stores all user state information. For the audio history, `TattleTrailUser` stores user state information concerning which message to begin with, desired time-scale modification speeds, and direction of playback. The user state may change whenever user input is received.

[1] UDP does not guarantee packet arrival, so packets may be lost in the network.

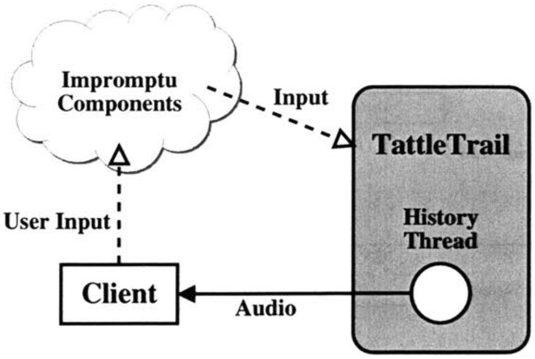


Figure 4-2: Architecture diagram of Catch-up Mode.

Depending upon the input received, AudioHistory will speed up time-scale modification of chat recordings, slow it down, or play backward. Some of the previous works reviewed used audio indexing techniques such as speaker identification to segment audio recordings of speech in order to facilitate browsing later [20, 26]. This, however, is not necessary in Catch-up Mode because the chat recordings are already segmented into separate messages recorded by each speaker. Another audio indexing technique explored in previous works was using long pause detection to indicate possibly interesting points or changes in topic of discussion [2, 20, 24]. Pause detection was not used in Catch-up Mode because chat messages tend to be short in duration, making the beginning of each message a logical point of interest. Figure 4-2 shows a high level architecture diagram of Catch-up Mode.[2]

The following sections describe the different modules used to implement the audio history in more detail. The next section describes how time-scale modification was implemented in TattleTrail by the Sola class.

4.3.1 The Sola Class

Background

Time-scale modification in TattleTrail is performed using a variation of the *synchronized overlap-add* algorithm (SOLA) presented in [9]. The SOLA algorithm was first

[2] Note that user input is received by the global ControlThread and the user's dedicated SpeechThread. These are not shown in the architecture diagrams for clarity.

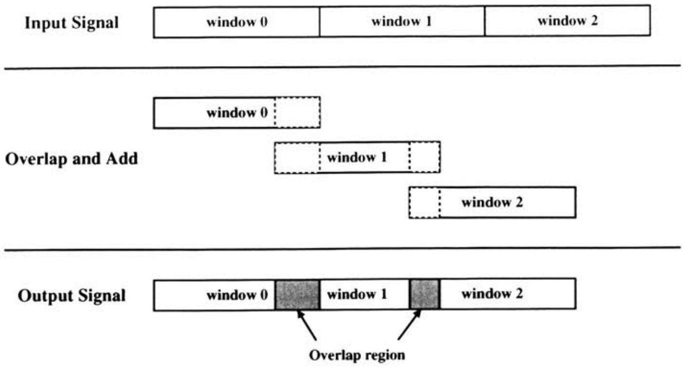


Figure 4-3: Time-scale modification using the SOLA algorithm.

Window Length (*winlen*)	32 msec
Analysis Shift (S_a)	32 msec
Shift Search Interval (K_{max})	12.5 msec

Table 4.1: Optimal parameter values for time-scale compression using SOLA.

described by Roucos and Wilgus [19], and has become a common method of time-scale modification of speech in real-time. It is computationally efficient because no complicated mathematical operations such as frequency-domain calculations, pitch extraction, and phase unwrapping are required for processing [15]. This computational efficiency makes SOLA suitable for real-time applications [1].

In the SOLA algorithm, a speech signal is separated into windows for analysis and processing. For time-scale compression, each window is shifted backward over the end of the preceding window, which shortens the entire speech signal (Figure 4-3). An optimal region of overlap between the windows is found by determining the highest point of cross-correlation. In other words, the current window is overlapped with the preceding window at a point where the intersection has a maximum similarity of signals. This region of overlap is then added and averaged together. SOLA can be viewed as a greedy algorithm, which always selects a locally optimal choice between subsequent windows, ultimately preserving the pitch, magnitude, and phase of the signal [1]. The overlap and adding of windows effectively removes redundant pitch periods in speech in the time-domain, which provides efficient time-scale modification outside of the frequency domain.

Implementation

The Sola class, which implements the SOLA algorithm, only supports time-scale compression because only rapid browsing of audio was desired.[3] Therefore, the optimal values of three parameters for time-scale compression reported by Hejna in [9] were used for the implementation (Table 4.1).

The Sola class is used to retrieve a block of audio from a .wav file, analyze it, and apply the SOLA algorithm to output a smaller block of time-scale compressed audio. Once Sola opens a .wav file and loads the audio data, the method getNextBlock() grabs a block of audio that is four windows long (128 msec). Each four-window block is analyzed, shifted with respect to a given speed scaling factor, and then output to the calling process by the method compressBlock(). In this way, an entire speech signal can be time-scale compressed on-the-fly with variable speeds by processing only small blocks of the signal at each iteration. Therefore, this implementation could support an entire speech recording to be time-scaled by a different speed factor for every four-window block it grabs and processes.

Processing small blocks of audio at a time forced the TattleTrail implementation of SOLA to perform extra computations. The SOLA algorithm initializes an entire speech signal, and processes it from beginning to end in a continuous manner. But because a TattleTrail user has the ability to interactively control the speed and direction of time-scale modification of speech in real-time, an entire speech signal must be initialized and processed in small blocks. Because the overlap and add step involves analyzing the current and preceding window, a pointer to the last window, the last windowing function $w(n)$, and buffer offsets from the preceding window are stored to allow correct processing of the next window. This was necessary when processing a new block of audio that needed the preceding window from the previous block processed. Therefore, each call to compressBlock() initializes parameters from the last block processed before analyzing the current block in order to perform correct time-scale compression.

[3] The SOLA algorithm can be used for time-scale expansion as well.

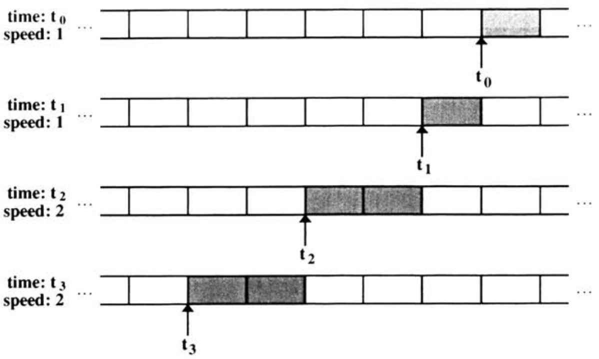


Figure 4-4: Playing backward. The audio is divided into segments of 4 seconds. Segments are played at each iteration (the gray boxes), then the pointer is set back to the previous segment for the next iteration. Note at times t_2 and t_3, the playback speed is twice as fast, so the algorithm jumps back 2 segments in order to play 4 seconds of time-scale compressed audio.

An optimization to the SOLA algorithm helped counter the extra computations necessary for block-by-block processing. The original SOLA algorithm uses a normalized cross-correlation function to find the point of maximum similarity between two windows. This function, however, accounts for more than two-thirds of all computations performed in the algorithm. This is due to division by a computationally expensive square root operation. Hejna suggested to use a *least difference* estimation instead of the normalized cross-correlation function as a possible optimization, which was found to produce time-scale compressed speech that was indistinguishable from the original function [9]. Therefore, the `Sola` class uses the least difference approach, which uses no square roots or multiplication, to increase performance.

The `Sola` class implementation also supports scaling audio "backward" for users wishing to jump backward during the audio history processing in Catch-up Mode. Samples of audio are not scaled and played in reverse order; this would result in unintelligible speech. Instead, the SpeechSkimmer technique of playing backward is used [2]. The speech signal to be processed is broken into segments of 4 seconds, then each segment is time-scaled normally (i.e. forward). Once an entire segment is processed, the following call to `getNextBlock()` returns the first block of audio in the previous segment for processing, as shown in Figure 4-4.

The `ChatCluster` class is examined in the next section.

4.3.2 The ChatCluster Class

Chat messages in TattleTrail are grouped into "clusters" of messages depending on the time each message was recorded. All chat messages are saved as timestamped files of the approximate start and end times of the recording.

The `ChatCluster` class represents an individual chat cluster. It stores a list of the filenames of the chat messages in the cluster, as well as methods to navigate through the list. `ChatCluster` also stores the relative duration of the entire cluster (i.e. total duration of all messages) in number of 128 msec blocks.[4] An internal pointer exists to represent a block position within this cluster time frame. When processing messages in a `ChatCluster`, these block structures give a relative "position" with respect to the entire cluster at any given time. Therefore, when `Sola` is used to time-scale a chat message in a `ChatCluster`, the controlling process can keep track of its progress in terms of position within the cluster of messages.

The relevance of determining position within a cluster of messages is described in more detail in the following section, which discusses the `AudioHistory` class and how the `ChatCluster` and `Sola` classes are used to help present a highly interactive and flexible audio history.

4.3.3 The AudioHistory Class

In TattleTrail, the `AudioHistory` class produces an audio history for the user by performing variable time-scale modification in real-time. `AudioHistory` manages which recorded messages to read from disk, at what speed to perform time-scale compression, and in which direction to play back audio.

When `AudioHistory` is instantiated, it examines the list of chat messages that are archived, and groups them into `ChatCluster` objects. `AudioHistory` then proceeds to process the first chat message using `Sola` to open the file and time-scale it according to the user's desired speed. If the user was previously participating in the synchronous Chat Mode, and just returned to TattleTrail, `AudioHistory` determines

[4] Note that this is the same size as blocks processed by the `Sola` class.

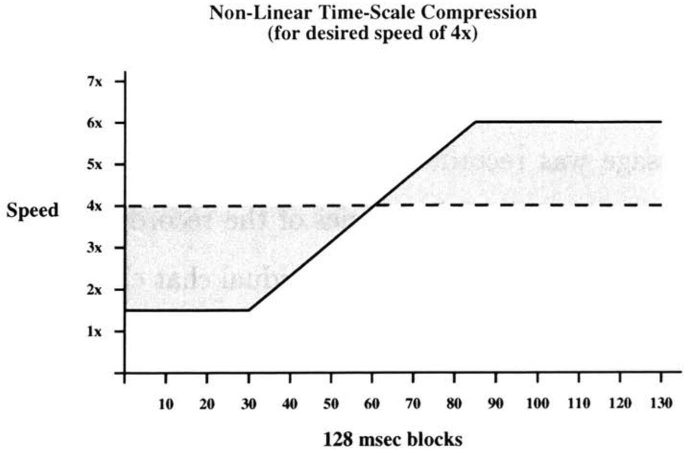


Figure 4-5: Non-linear time-scale compression of a chat cluster. The first 30 blocks (3.84 seconds) of the cluster are scaled at a speed of 1.5x. Thereafter, speeds are gradually increased until a breakpoint is reached at the max speed of 6x. The user's desired speed of 4x is effectively reached, as the shaded regions are of approximately equal area.

the last message, if any, that the user heard within the chat. It then begins processing the beginning of the `ChatCluster` preceding the cluster of the last message heard. This functionality allows the user to skim one cluster before reviewing the messages missed while away.

High Speed Browsing

TattleTrail performs non-linear time-scaling for users browsing chat messages at rates greater than twice normal speed. This non-linear time-scaling is structured around chat clusters, relying on the assumption that the first message of the cluster (or conversation) usually sets the topic for the rest of the messages in the cluster. This browsing technique differs from previous works that used structured audio to find meaningful points to emphasize [2, 20, 24, 26].

Therefore, the `AudioHistory` class processes the beginning of each `ChatCluster` object at a slower speed, and rapidly speeds up time-scale compression of subsequent messages in the `ChatCluster`. This non-linear time-scaling delivers all the messages in a single cluster at an effective high speed desired by the user while providing some intelligibility of the chat content.

The non-linear rate of compression is determined by the relative position within the ChatCluster before processing each block. Once a user's desired speed breaks the threshold of twice normal speed, AudioHistory begins applying the non-linear technique to the time-scaling of the cluster. The beginning of the cluster is compressed by a static speed of 1.5 times normal for an established duration of 3.84 seconds (30 blocks of 128 msec each). Then the speed is gradually increased linearly until the cluster position reaches a breakpoint calculated from the ChatCluster (Figure 4-5). Thereafter, the speed stays constant until the end of the cluster is reached. This non-linear compression technique allows the user to skim the entire cluster at an effective rate, while actually varying the speed during browsing. This is accomplished by finding the correct slope and breakpoint of the cluster depending upon its total duration. Note that if the cluster is not long enough, the actual effective rate may not be reached due to the initial 30 block compression stage and the maximum speed of 6 times normal.

Audio History Termination

After the user reaches the last chat message of the audio history, or explicitly leaves, the HistoryThread begins termination procedures by closing the TCP socket connection to the Impromptu client device. If the user left the TattleTrail application entirely, the HistoryThread simply terminates and the application deactivates the user. Otherwise, the user proceeds into Chat Mode.

The next section describes Chat Mode and its implementation.

4.4 Synchronous Chat

Chat Mode allows users to speak to each other synchronously over IP. More specifically, when a user speaks, all others currently in Chat Mode hear the user synchronously (i.e. real-time, but with slight network delay). To minimize delay, all audio sent over IP during the synchronous chat are sent using UDP sockets. Continuously streaming audio over a TCP socket connection would gradually increase delays due to packet loss. All packets received by the TattleTrail server are saved during the chat so that all messages can be archived for browsing. When a user leaves the Chat Mode, a pointer to the last message heard in real-time is stored in the `TattleTrailUser` object to provide an audio history "bookmark."

The next section describes how recording is implemented.

4.4.1 The Record Thread

When the `TattleTrail` application object is started, the `RecordThread` is spawned. Only a single thread is necessary per chat group for recording purposes because only a single user can speak at a time.[5]

The `RecordThread` operates simply and efficiently because all messages spoken during Chat Mode are sent as IP multicasts, allowing audio packets to be sent peer-to-peer rather than being channeled from one client to all other clients through the server. This offloads sending and receiving processes equally among all Chat Mode users and the TattleTrail server.

When a user speaks, his/her audio is broadcast to a specific IP multicast address. All multicast traffic is handled at the transport layer via UDP, forwarding multicast packets to all routers for a certain number of hops designated by a time-to-live [5]. All IP multicast Level 2 network hosts receive and ignore multicast traffic unless the kernel is told specifically to "watch" for packets sent to a certain multicast address [5]. When a host "joins" a multicast group address, it is actually informing the kernel to pick up any packets with that specific multicast address instead of ignoring them. Therefore, a chat group in TattleTrail is actually represented by all those users that have "joined" a specific IP multicast group address.

The `RecordThread` opens a UDP socket and joins the multicast group for the chat. Any user in Chat Mode will have joined this multicast group as well. All spoken messages are sent to this group, so the `RecordThread` simply reads all IP multicast traffic and saves it to a file. The push-to-talk mode allows only one speaker at a time, which

[5] The current implementation only supports one chat group, but support for multiple chat groups is a practical extension.

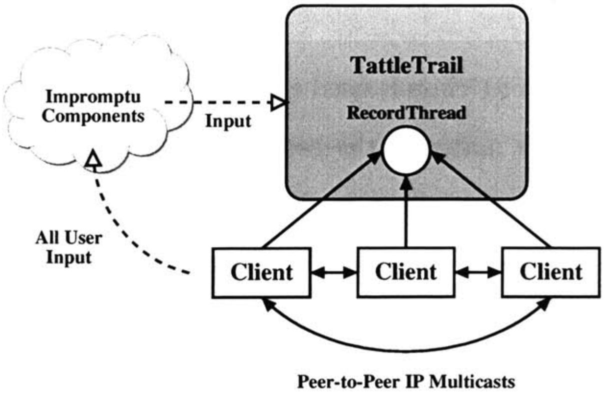


Figure 4-6: Architecture diagram of Chat Mode.

conveniently divides all bursts of multicast traffic to belong to a single user, allowing for easy message archiving. When no user is speaking, the RecordThread is suspended to save server resources. This is done through thread synchronization controlled by the AudioFloorControl class. When a message is to be recorded because a user wishes to speak, AudioFloorControl wakes the RecordThread. Likewise, the thread is put to sleep when a user is finished recording. Figure 4-6 shows the high level architecture diagram of Chat Mode.

The next section describes how the floor control protocol used to enforce push-to-talk mode is implemented.

4.4.2 The AudioFloorControl Class

TattleTrail provides a first come, first served floor control protocol for users wishing to speak during Chat Mode. This protocol is suitable for groups of users sharing networked audio resources because conversations naturally follow turn-taking behavior [6, 22]. It also eliminates possible interruptions and false starts that could result from network delay and lack of visual cues if the chat channel had an open floor [21]. Another benefit of using a floor control protocol is to avoid some of the problems found in previous audio-only media spaces where a participant's environment was constantly being recorded, which was sometimes unnecessary and slightly intrusive [10, 23]. The floor control ensures that audio is only recorded when a user wishes to do so, which maintains privacy and reduces unnecessary audio broadcasts to the rest of the group.

The AudioFloorControl class is used to manage which user receives control of the "floor," and consequently manages the recordings of the floor that are saved during the chat. Control of an "open" floor is given to any user requesting to speak. If the floor is open, the first user to request the floor receives control. A user request is determined by the TattleTrail application from a specific button press event (the Impromptu *record* button). The first-to-press detection is fair because TattleTrail listens over the network for user input messages, so the first user request message received wins floor control.[6]

Once a user gains control of the floor, AudioFloorControl stores the current time and wakes the RecordThread. Because it is push-to-talk, the user must keep the *record* button pressed while speaking. When the user releases the button, this event is relayed to TattleTrail. AudioFloorControl determines the finishing time, puts the RecordThread to sleep, and saves the file under a time-stamped name indicating the start and end time (represented in seconds elapsed since January 1, 1970):

```
1021401507-1021401513.wav
```

As mentioned earlier, the start and end times give an estimated duration of the chat message, which is pertinent to the AudioHistory class in structuring chat clusters according to pauses between subsequent messages, and for non-linear time-scaling.

If a user requests the floor, but does not receive it, then another user must have floor control. All requests made after control has been issued are queued by AudioFloorControl and dequeued in a first in, first out (FIFO) order. When the user with the floor finishes speaking, the floor is released and given to the first user in the request queue. Any user waiting in the queue must also keep the *record* button pressed in order remain in the queue. If the user releases the button while waiting, the AudioFloorControl is notified of the event and pops the user from the request queue.

[6] It is unfair, however, if network delay causes a user that pressed first in real-time to not gain control. This exception is acceptable because network delay cannot be controlled.

AudioFloorControl currently does not prevent abuses of floor control. Therefore, any user in control can hold the floor indefinitely, causing starvation of other users requesting the floor. The floor control implementation relies upon responsible Impromptu users; a more strict protocol could be implemented by setting a timer thread for a maximum length of a recorded message in order to prematurely retract the floor from a user if necessary.

The next section describes how the new alerting mechanism is implemented.

4.5 Alerting

Alert Mode delivers asynchronous alerts to passive and background users in TattleTrail. Once a message is finished recording, it is streamed via UDP. As mentioned earlier, streaming audio data using UDP sockets minimizes network delay and provides IP multicast functionality. When sending data to multiple recipients simultaneously is necessary, IP multicasting is an obvious choice. Therefore, all TattleTrail alerts are broadcast to Alert Mode users via IP multicasting.

The alerting mechanism in TattleTrail added a new alerting paradigm to the Impromptu framework, as described in the previous chapter. Therefore, some additional functionality was implemented within the Impromptu client software module to provide a separate alert channel (i.e. a UDP socket), as well as audio processing techniques such as mixing and fading two streams of audio together.

When a user requests to drop into either Passive or Background Alert Mode, TattleTrail instructs the client device to join a different IP multicast group for receiving alerts, and adds the user to hash maps representing passive or background users. The same multicast address is used for both passive and background users. This is possible because both types of users receive the alerts; they just perform different audio processing techniques locally on the client device like fading or mixing audio. When a user leaves either mode of alerting to rejoin the TattleTrail chat, the client device also leaves the multicast group so that future alert packets sent will be ignored. Users are, of course, removed from the internal TattleTrail hash maps as well.

The main module used to provide alerting is the AlertThread, which is described in the next section.

4.5.1 The Alert Thread

When the TattleTrail application object starts up, the AlertThread is spawned. The AlertThread can be compared to the RecordThread in that they provide exactly opposite functionality, yet are implemented similarly. Both rely upon IP multicasting to efficiently send or receive audio by requiring the underlying network to perform the grunt work of forwarding packets to multiple destinations. The RecordThread simply received all IP multicasts sent to a particular multicast address representing the chat. The AlertThread does the opposite; it sends audio packets to a particular multicast address representing Alert Mode users.

After the AlertThread is first created, it establishes an inter-process TCP socket connection to itself. This internal socket is then passed to the AudioFloorControl class, which manages all the creation and saving of chat messages. The socket acts as an inter-process message channel between the main TattleTrail processing loop and the AlertThread. Thread synchronization procedures used for the RecordThread were considered, but the TCP socket implementation was chosen instead because of simplicity and efficiency.

When AudioFloorControl saves a chat recording to a file, it also sends the file name over the TCP socket to the AlertThread. The AlertThread blocks on that socket until data is received, which keeps the thread asleep until a new message is recorded. This allows TattleTrail to save server resources by minimizing thread cycles. Once the filename is received, the AlertThread loads the file data, appends the first half of the TattleTrail audio icon to the message by mixing and fading the two files together, streams the processed audio to the alert IP multicast group, and begins blocking on the message socket again.

Streaming audio from files on the server to clients via IP multicasts proved to be problematic initially. Because of UDP properties, an entire file cannot be sent at a time; overloading any IP host's internal network buffer with UDP packets will result

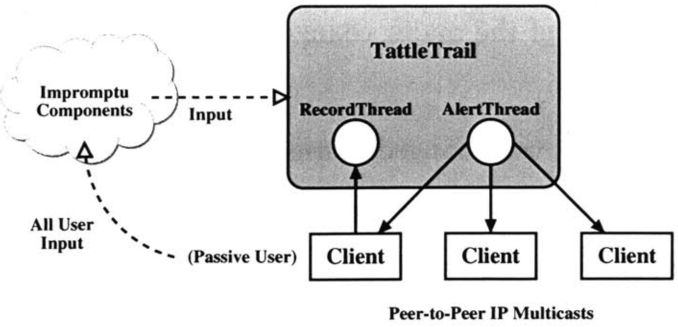


Figure 4-7: Architecture diagram of Alert Mode.

in packets getting dropped, thus never reaching the application layer. To counter this, the `AlertThread` pauses for a certain number of milliseconds between each buffer sent over the network.

In addition, UDP does not guarantee packets will be received, nor the order in which they will be received. Therefore, a client device receiving UDP alert packets cannot know if a packet received is from the beginning of the alert message, the middle, or the end. Client devices must know the beginning and end of each alert message received in order to correctly mix, scale, or fade the audio as described in the previous chapter. Therefore, before starting to stream the alerts, the `AlertThread` sends appropriate messages to passive and background users stored in the hash maps via the Impromptu message channel. When the last alert UDP packet is sent, the `AlertThread` similarly sends all passive and background users messages signifying the end of the alert UDP multicast. In this way, both passive and background users can be listening to the same IP multicast address, but receive different control messages from TattleTrail about how to present the audio to the user. Figure 4-7 shows a high level diagram of the architecture of Alert Mode.

4.5.2 Passive User Interaction

Passive users in TattleTrail are permitted to record messages to the chat synchronously, although they are not actively using the application. The architecture of TattleTrail allowed support for this functionality without any changes, besides allowing the `AudioFloorControl` class to accept a request from a `TattleTrailUser` that was in

Passive Alert Mode. All of the major changes were implemented in the Impromptu client software module.

The Impromptu client was changed to send *record* button press and release events to the TattleTrail application whenever the user is in Passive Alert Mode. When pressed, the client device temporarily joins the Chat Mode IP multicast group and sends all recorded audio until the button is released. This is assuming that floor control is granted; nonetheless, all interactions while the *record* button is held down exactly mimic those interactions of active users in Chat Mode. TattleTrail treats the user as if he/she were actively in chat due to the button press and release events received. When the button is released, the client device leaves the IP multicast group and ignores the ensuing asynchronous alert multicast sent by the `AlertThread`. This prevents a passive user from hearing his/her own recorded message.

4.6 Summary

This chapter has presented the software architecture of TattleTrail. The chat system has been designed to support multiple mobile clients within the Impromptu framework with synchronous and asynchronous communication channels offering wide ranges of audio processing techniques. An overall architecture diagram of TattleTrail with all modes of interaction is displayed in Figure 4-8.

The design and implementation made heavy use of IP multicast via the UDP layer to offload the sending of packets to multiple recipients onto the network. The peer-to-peer framework also allowed flexibility in establishing "chat rooms" and awareness groups. Whenever sending data packets from one to many is required over IP, using the built-in multicast protocol ensures simplicity and efficiency as long as packet loss can be tolerated.

The major bottleneck in the system is the Catch-up Mode implementation, which performs interactive time-scaling. Because audio history cannot be handled in a peer-to-peer fashion where processing can be equally divided, TattleTrail must establish a point-to-point connection to service a particular user. Providing audio history

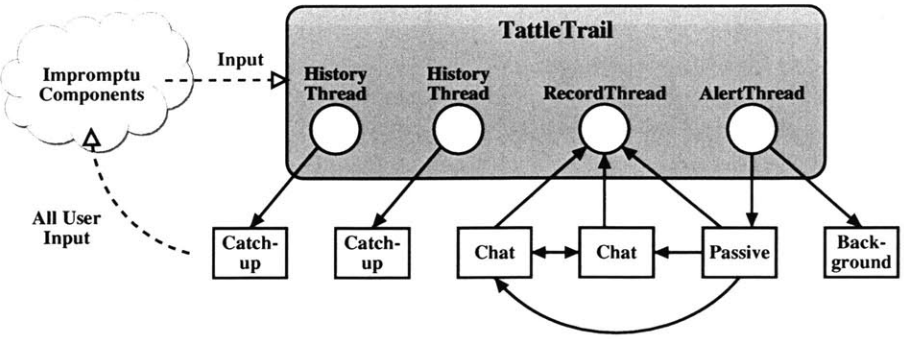


Figure 4-8: Architecture diagram of TattleTrail. The clients in Catch-up Mode have a dedicated **HistoryThread** streaming audio. Chat Mode users IP multicast audio packets to each other and the **RecordThread**. Users in Alert Mode receive asynchronous IP multicasts from the **AlertThread**. Note that the passive user can synchronously send IP multicast chat messages to those in Chat Mode.

capabilities forces the server to devote a single thread per Catch-up Mode user to process several audio files while responding to user input events. This is much more draining than using a single recording thread to support multiple users that are in Chat Mode, or using a single alert thread to provide awareness to multiple users in Alert Mode.

Chapter 5

Conclusion

TattleTrail has been designed and developed to address current limitations in mobile communication, namely resulting from a lack of flexibility of protocols and communication modalities, in order to shed light upon different approaches toward staying connected. It is no doubt that for mobile users, the ability to communicate with others is paramount. Although text based systems offer mobile users a wide variety of applications such as email and instant messaging, communication by voice remains superior.

This is evident by the soaring use of mobile phones today. Whether for work or play, people on the move have made connectivity a necessity. The glaring issue, however, is the old, restrictive communication protocols enforced by traditional telephony. TattleTrail demonstrates that when using a more flexible underlying network such as the Internet, these old barriers and confinements can be broken, and new communication modalities explored.

TattleTrail provides mobile users with the ability to instantly communicate to a group of users synchronously, anywhere and anytime Internet connectivity can be obtained. Although this is currently not ubiquitous, the assumption is that wireless networks will soon provide this type of access just as mobile phones today can obtain service signals from almost anywhere in the country. Judging from the high popularity of mobile phone use and instant messaging use, combining the two into a mobile voice chat system logically follows as a next step in mobile communication.

This ability to actively chat in real-time over IP, however, is not the only communication modality supported by TattleTrail. Through the use of passive, asynchronous alerts, TattleTrail also provides mobile users with a less intrusive mode of communication while maintaining peripheral awareness of chat activity.

Another feature drawing from online chats is the ability to catch-up to a chat conversation by asynchronously skimming through an audio history of chat messages previously recorded. This can be especially useful in a mobile environment where connectivity can be sparse, and external events can cause long periods of interruption or separation from the communication channel. In these cases, it is advantageous to provide a means to asynchronously listen to any message that may have been missed. The interactive browsing techniques of conversations using time-scale modification also raise the important issue of how to peruse recorded speech when treating voice as a data type.

The different combinations of synchronous and asynchronous communication modalities implemented within TattleTrail offer new points of direction for mobile communication, and is perhaps the most important contribution of this thesis. Of particular interest is the new alerting model which provides a hybrid communication channel that receives asynchronous alerts while sending synchronous messages. TattleTrail has demonstrated that transitions across these synchronous-asynchronous boundaries do not necessarily imply that communication among parties must be halted. In other words, communication between two parties in real-time can be continued even when one party drops, either willingly or possibly from loss of wireless connectivity, through asynchronous messaging.

TattleTrail has explored various aspects of mobile communication modalities afforded by the flexibility of IP, and has proven that new models of interaction via mobile devices will become a possibility in the near future. Hopefully, the feasibility of providing such functionality found in TattleTrail can soon become a reality in order to break the current limitations in mobile communication today.

5.1 Future Work

TattleTrail has quite a few areas where expansion and future work can be applied. Most important is the ability to support several chat groups simultaneously. The architecture of TattleTrail is conducive to this type of expansion due to the simplicity of using IP multicasting.

An additional function that could be supported in the future is the ability to open a private, full duplex audio connection with another user in the chat. This would be useful if two people became engaged in a conversation in the chat that would be more suitable to a continuous duplex connection similar to a phone call rather than using push-to-talk mode.

A popular feature of online chat systems that has not been replicated in TattleTrail is the use of buddy lists to help give a sense of awareness of a specific group of people. Because TattleTrail was integrated into an audio-only framework, a graphical user interface (GUI) was not an option, so displaying a buddy list was not possible. However, ways in which buddy list information could be relayed should be explored. One possible solution is to provide awareness of buddies using specific earcons, as implemented by the Hubbub instant messenger [11].

Bibliography

[1] B. Arons. Techniques, perception, and applications of time-compressed speech. In *Proceedings of American Voice I/O Society*, pages 169–177, 1992.

[2] B. Arons. Speechskimmer: Interactively skimming recorded speech. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pages 187–196. ACM, 1993.

[3] S. Bly, S.R. Harrison, and S. Irwin. Media spaces: Bringing people together in a video, audio and computing environment. *Communications of the ACM*, 36(1):28–47, January 1993.

[4] B. Chalfonte, R. Fish, and R. Kraut. Expressive richness: A comparison of speech and text as media for revision. In *Proceedings of Human Factors in Computing Systems (CHI'91)*, pages 21–26. ACM, 1991.

[5] J. de Goyeneche. Multicast over TCP/IP HOWTO (Linux Online!). http://www.linux.org/docs/ldp/howto/Multicast-HOWTO.html, 1998.

[6] H.-P Dommel and J.J. Garcia-Luna-Aceves. Design issues for floor control protocols. In *Proceedings of Multimedia and Networking*, pages 305–316. IS&T SPIE, 1995.

[7] P. Dourish, A. Adler, V. Bellotti, and A. Henderson. Your place or mine? Learning from long-term use of audio-video communication. *Computer Supported Co-operative Work*, 5(1):33–62, 1996.

[8] H. Fagrell, F. Ljungberg, M. Bergquist, and S. Kristoffersen. Exploring support for knowledge management in mobile work. In *Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work*, pages 259–275, 1999.

[9] D. Hejna Jr. Real-time time-scale modification of speech via the synchronized overlap-add algorithm. Master's thesis, Massachusetts Institute of Technology, February 1990.

[10] D. Hindus, M. Ackerman, S. Mainwaring, and B. Starr. Thunderwire: A field study of an audio-only media space. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, pages 238–247. ACM, 1996.

[11] E. Isaacs, A. Walendowski, and D. Ranganthan. Hubbub: A sound-enhanced mobile instant messenger that supports awareness and opportunistic interactions. In *Proceedings of Human Factors in Computing Systems (CHI'02)*. ACM, 2002.

[12] S. Kristoffersen and F. Ljungberg. "Making place" to make IT work: Empirical explorations of HCI for mobile CSCW. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP'99)*, pages 276–285. ACM, 1999.

[13] K. Lee. Impromptu: Audio applications for mobile IP. Master's thesis, Massachusetts Institute of Technology, September 2001.

[14] W. Mackay. Media spaces: Environments for informal multimedia interaction. In M. Beaudouin-Lafon, editor, *Computer-Supported Cooperative Work*, Trends in Software Series, pages 55–82. Wiley and Sons, 1999.

[15] J. Makhoul and A. El-Jaroudi. Time-scale modification in medium to low rate coding. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1705–1708. IEEE, 1986.

[16] K. O'Hara, M. Perry, A. Sellen, and B. Brown. Exploring the relationship between mobile phone and document use during business travel. In B. Brown,

N. Green, and R. Harper, editors, *Wireless World: Social and Interactional Implications of Wireless Technology.* Springer-Verlag, 2001.

[17] J. Orr. Ethnography and organizational learning: In pursuit of learning at work. In *Proceedings of the NATO Advanced Workshop on Organizational Learning and Technological Change*, pages 47–60. NATO, 1993.

[18] M. Perry, K. O'Hara, A. Sellen, B. Brown, and R. Harper. Dealing with mobility: Understanding access anytime, anywhere. In *ACM Transactions on Computer-Human Interaction.* ACM, 2002. (To appear).

[19] S. Roucos and A. Wilgus. High quality time-scale modification for speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 493–496. IEEE, 1985.

[20] D. Roy and C. Schmandt. NewsComm: A hand-held interface for interactive access to structured audio. In *Proceedings of Human Factors in Computing Systems (CHI'96)*, pages 173–180. ACM, 1996.

[21] K. Ruhleder and B. Jordan. Co-constructing non-mutual realities: Delay-generated trouble in distributed interaction. *Computer Supported Cooperative Work*, 10(1):113–138, 2001.

[22] H. Sacks, E. Schegloff, and G. Jefferson. A simplest systematics for the organization of turntaking for conversation. In A. Schenkein, editor, *Studies in the Organization of Conversational Interaction*, pages 7–55. Academic Press, New York, 1978.

[23] A. Singer, D. Hindus, L. Stifelman, and S. White. Tangible progress: Less is more in Somewire audio spaces. In *Proceedings of Human Factors in Computing Systems (CHI'99)*, pages 104–111. ACM, 1999.

[24] L. Stifelman, B. Arons, and C. Schmandt. The Audio Notebook: Paper and pen interaction with structured speech. In *Proceedings of Human Factors in Computing Systems (CHI'01)*, pages 182–189. ACM, 2001.

[25] H. Strub. ConcertTalk: A weekend with a portable audio space. In *Proceedings of the 6th IFIP Conference on Human-Computer Interaction (INTERACT'97)*, pages 381–388. IFIP, 1997.

[26] L. Wilcox, D. Kimber, and F. Chen. Audio indexing using speaker identification. In *Proceedings of SPIE Conference on Automatic Systems for the Inspection and Identification of Humans*, pages 149–157. SPIE, 1994.

[27] L. Wilcox, B. Schilit, and N. Sawhney. Dynomite: A dynamically organized ink and audio notebook. In *Proceedings of Human Factors in Computing Systems (CHI'99)*, pages 186–193. ACM, 1997.