Mediated Voice Communication via Mobile IP

Chris Schmandt, Jang Kim, Kwan Lee, Gerardo Vallejo, Mark Ackerman
Speech Interface Group
MIT Media Laboratory
20 Ames Street, Room E15-327
Cambridge, MA 02139
+1-617-253-5156
{geek, jangkim, kwan, gvallejo, ack}@media.mit.edu

ABSTRACT

Impromptu is a mobile audio device which uses wireless Internet Protocol (IP) to access novel computer-mediated voice communication channels. These channels show the richness of IP-based communication as compared to conventional mobile telephony, adding audio processing and storage in the network, and flexible, user-centered call control protocols. These channels may be synchronous, asynchronous, or event-triggered, or even change modes as a function of other user activity. The demands of these modes plus the need to navigate with an entirely non-visual user interface are met with a number of audio-oriented user interaction techniques.

KEYWORDS: Computer-mediated communication, ubiquitous computing, audio user interfaces, speech user interfaces

INTRODUCTION

Computer mediated textual communication provides a rich mix of delivery mechanisms: email, IRC chat, Zephyr, instant messaging. These vary by attributes such as message length, whether delivery is synchronous (relatively speaking) or asynchronous, and whether targeted to an individual or a group. In at least some countries, the popularity of these mechanisms is enough to threaten conventional paper-based postal systems. At the same time, short text message (SMS) traffic over mobile telephone networks has also soared, along with pager or PDA-based two-way text messaging in North America; these methods take text messages mobile. An important consequence of this variety of messaging protocols is that users usually have more options and hence can make more and or better use of the messaging channel.

Despite the efficiency and richness of voice communication, messaging options in this modality are more limited. Although fixed and mobile telephones provide fair to excellent quality synchronous conversations, it is usually impossible to leave a voice message without ringing the recipient's phone.

Family band radios are the current generation of no-license walkie-talkies; they are half duplex, push-to-talk, one-to-many, with a range of up to several miles, and are popular among outdoor enthusiasts. Nextel provides a push-to-talk walkie-talkie mode in its Direct Connect service, which is overlaid on an otherwise conventional cellular telephone network; it is popular in the building trades. Although some PDAs now include telephones, the phones still act as single channel voice communication devices.

Impromptu is a mobile IP-based communicator which bridges these worlds (voice/text, telephone/computer) by providing the flexibility of multiple modes of voice communication in a single, multi-function mobile computing environment. It goes beyond the telephony model by supporting multiple levels of synchrony, presenting a variety of call management strategies, and allowing multiple applications to run simultaneously. Impromptu extends the window management functions of traditional screen-based computers into the mobile world with an auditory user interface and eyes-free operation. Further, the addition of audio storage and processing in the network enhance the user experience of conventional channels such as telephony.

Impromptu has supported three classes of voice channels:

- **synchronous full duplex**, like a telephone
- **synchronous half duplex**, using floor control
- **asynchronous**, record and send, like text chats
- **event-triggered**, in which a channel is set up in response to an external stimulus.

An important contribution of the Impromptu voice channels is the use of audio processing, in particular at transitions, both into the application supporting each channel type, as well as state changes within the channel. Visual user interfaces are largely static, idle until the user makes a choice or initiates an action. Because audio is continually streaming from Impromptu and the user interface is auditory as well, particular care must be taken as to how these two forms of sounds interact. Some of these aspects are subtle and not noticed except in their absence, such as cross-fading at sound boundaries to minimize abrupt transitions.

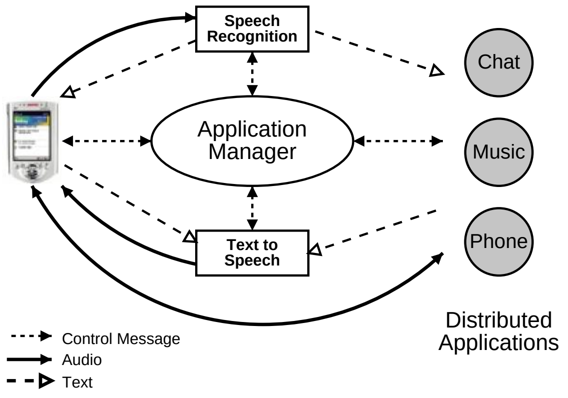

Figure 1: The Impromptu Architecture.

IMPROMPTU

Impromptu runs on iPaqs running Linux, using 802.11b for IP-based networking. Motivated by our concern for mobility, Impromptu does not use the screen for display or input; speech recognition and synthesis are provided as services in the network. Applications run simultaneously but the user generally listens to only one at a time. A number of applications have been programmed, with concern for those appropriate for the packet delivery characteristics of an IP network (i.e. delay, jitter, and packet loss):

- MP3 music player
- books on tape
- headline news using speech synthesis
- radio
- personal recorder/to-do list
- telephone
- chat
- baby monitor
- burglar alarm

An application manager (Figure 1) registers each application as it makes contact through the network, gets an **audio icon** from the application so it can be identified to the user when going active, accepts a speech recognition vocabulary for the application, and assigns a port for the application to connect to for bi-directional audio streaming. Note that we do not route audio through the application manager as this would incur additional delays; well behaved clients transmit only when allowed, while ill-behaved clients are ignored by Impromptu but waste network bandwidth. The application manager can also present an auditory **alert** on behalf of an inactive application which wishes to become active based on some external event, and informs the user via sound when applications die or reconnect during a session.

A **push-to-talk** button activates speech recognition. Each application may be invoked by voice, and whenever an application activates its additional vocabulary is loaded into the recognizer. Speech is sent to the recognizer, and the results are routed to the application if the command was application-specific, or back to the application manager (*"where am I?"*, *"what are the applications?"*).

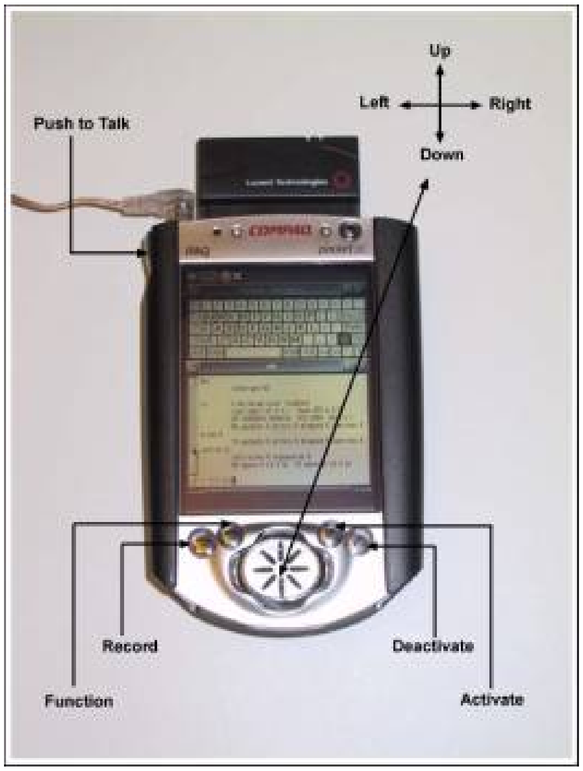

Figure 2: The iPaq Button Mapping.

Impromptu uses a number of other iPaq buttons in addition to speech recognition (Figure 2); most but not all of these duplicate speech input functionality. The **wheel** at the center bottom cycles through the applications (pressing left or right) and is also used in an application-specific manner (pressing up or down). Up and down usually have the syntax of moving through a list of entities germane to the application, such as songs, news items, or entries in the audio to-do list. The **record** button is always active and saves to a file whatever is being heard while the button is pressed. The **activate** button is reserved for responding to alerts from one of the non-active applications. It was stated above that the display was not used for input or output, but in fact it is used for entering numbers for placing conventional calls to a number not in a user's voice controlled address book, and also displays a few appropriately obtuse Unix error messages.

In shifting user focus between multiple applications without visual feedback, Impromptu relies on an auditory icon associated with each application. This distinctive sound (all audio feedback is user-configurable) identifies each application and is played when the user activates it. This sound is also used entirely or in part when an inactive application plays an alert to gain the user's attention. A more complex interaction results when multiple applications run simultaneously; the user may be listening to the radio, but audio chat messages are merged with the radio stream to allow the user to multi-task.

Because speech recognition is prone to error, voice user interfaces require special care as to feedback, a good example of some of the factors involved with the Impromptu interface. Because we expect that systems such as Impromptu will be used in noisy environments, the recognition is push-to-talk enabled. When the user pushes the "talk" button, Impromptu immediately ceases audio output; this both provides feedback that the system is listening, but also increases the probability of successful recognition. If the user lets up the button or recognition times out with no word recognized, a noisy "bonk" sound provides negative feedback. A "ching" sound to indicate that speech was recognized is needed *only* if the invoked action would otherwise not generate its own feedback. For example, saying "music" invokes the MP3 player application, playing its characteristic short guitar riff on entry, so no other feedback is necessary. However, if the user says "Call Jang", some time may elapse during call setup before Impromptu will emit any audio, so the positive feedback sound is required. Excessive audio feedback is annoying primarily because it takes a second or two to play each of the feedback sounds, slowing down interaction.

CHANNEL CHARACTERISTICS

This paper specifically focuses on the Impromptu applications supporting person-to-person communication, rather than the overall architecture and user interface, which is covered in detail in [6]. We seek to demonstrate the suitability of a wide range of voice messaging in mobile IP. In particular, such an environment enhances communication by providing:

- voice channels with different transmission characteristics for different tasks
- negotiated call setup and flexible alerting to enable foreground and background activities
- storage so a channel can support catch up when it is reentered
- audio processing, such as time compression, event detection, and mixing, to improve usability

A number of distinctions may be made to differentiate various communication channels. One is *medium*; Impromptu currently supports only voice for communication between people. Another is *synchronous* (as in a live telephone call) vs. *asynchronous* (as in a message on an answering machine), but as we will see below, the same channel may be used in a way which alternates between the two. Another distinction is whether a channel is *one-to-one*, *one-to-many*, or *many-to-many*. Channels are often characterized by whether participants are in the *same place* or *different places*, but Impromptu is designed only for the latter case.

Digital protocols underlying communication channels afford a new class of distinctions between channels, which has to do with the difference in channel behavior when *active* vs. *inactive*. A conventional telephone is either one or the other, except for the brief phase when the recipient's phone is ringing but has not yet been answered.

But computer-mediated channels can be designed to support either a more graceful *negotiation* between parties during the setup or activation phase, or use variable bandwidth and implement a *background awareness*. For example, a video space such as Portholes [3] might update the view into a neighbor's office only once every few seconds or minutes, but if both parties notice the other they might switch to a full frame rate link. Hubbub [5] is an audio-based messaging and group awareness system which automatically notifies others when a group member becomes active, by playing a sound specific to that person. These examples illustrate the blurring between active and idle through transition or awareness stages. Smith and Hudson [11] use the audio energy of a person's current conversation to modulate a stored acoustic profile of that person to create a minimally disturbing background audio presence.

An audio channel with *storage* allows conversations or events to be saved for future listening. Especially in a multi-tasking environment, made worse by use in the distracting real world, storage allows a user to catch up on a channel when free to attend to it. Additionally, *audio processing* techniques allow more rapid catch up through silence removal, time compression, and clustering. Although quite subtle once implemented, mixing and fade in/out techniques reduce intrusiveness of interruptions or alerts.

The remainder of this paper will present three sets of voice communication channels implemented for Impromptu. Not only do each of them represent different points in the above described set of channel distinctions, some of these channels can be used or configured to exhibit different behaviors. We will show the flexibility and range of user interface options available by IP.

TELEPHONY – SYNCHRONOUS CONNECTIONS

The first class of Impromptu channel is *telephony*, as in an ordinary telephone call, i.e., a synchronous full duplex (bidirectional) audio connection. In reality delays of 10s of milliseconds may occur in these connections, but when delays exceed 150 or 200 milliseconds, ordinary conversation begins to break down due to the inability to interrupt the other party. There are a number of commercial "voice over IP" (VoIP) products, ranging from telephone sets which plug into LANs and run IP to computer-to-computer "free phone" products, of which NetMeeting is a high end example. If the audio is to leave the IP network it must go through a PSTN (Public Switched Telephone Network) gateway, which speaks protocols and has electrical characteristics of the telephone network.

To date, providing telephony functions over IP has simply replicated the existing binary call setup procedure; the network alerts and the remote party either answers or not. The caller has no control, cannot choose just to send a voice message, and does not know how busy the called party is. Quiet Calls [8] allows some negotiation during which the called party can hear the caller but transmit only limited prerecorded messages, and perhaps transition to a full duplex audio connection. Impromptu instead uses the audio channel for negotiation, and gives the caller some control in addition to the recipient.

Impromptu's telephone application, **Garblephone**, calls to or from the PSTN (i.e., the "address" of the call is a telephone number), or peer-to-peer to other Impromptu clients. Only peer-to-peer calls have the advantage of a sophisticated call setup protocol. Garblephone's audio icon is a single telephone ring.

Conventional telephony is the least interesting Garblephone channel, as it simply acts like a mobile phone. To place a call, the user enters a number on the iPaq screen (the only use of the screen for any Impromptu application), a PC with a telephone interface peripheral ("computerfone" from Suncoast Systems) dials a call, and audio from the phone line is digitized and streamed to/from the iPaq.

Since the user will most likely be listening to another application when an incoming call arrives, Garblephone must request the application manager to sound an alert, the single phone ring. Then normal playback resumes, and the user has ten seconds in which to press the **activate** button or the call goes to voice mail. With the exception that the computerfone box detects the end of the call, allowing Garblephone to play a cue (slamming door) if the other party terminates first, Garblephone behaves pretty much like any mobile phone.

Garblephone becomes more interesting, revealing the impact of new call setup protocols on the telephone user interface, for direct Impromptu-to-Impromptu calls. The caller connects by voice command (*"Call Jang"*) to the application process, which runs on the PC with the analog telephone hardware. Garblephone alerts the called party with an audio cue consisting of a voice recording of the caller stating his or her name; the goal of alerting is to convey both the existence of the call and the identity of the caller. We could use text-to-speech synthesis for this alert, but for such a short alert, which also usually comes in the midst of whatever audio (music, radio, etc) the user is hearing, there is strong added value to hearing the natural timbre and intonation of the caller.[1]

If the called party ignores the alert, the caller is dropped into

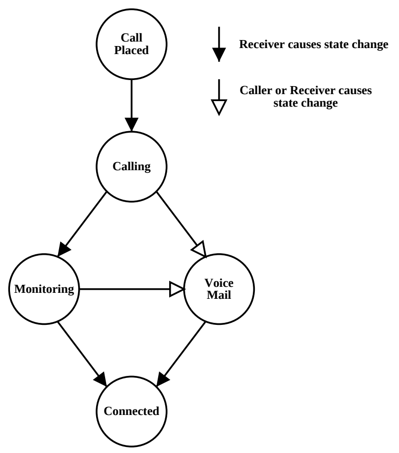
Call Placed
Receiver causes state change
Caller or Receiver causes state change
Calling
Monitoring
Voice Mail
Connected

Figure 3: State changes in Garblephone.

voice mail. If the called party responds by pressing the **activate** button, call setup proceeds through a series of mutually negotiated states of connectedness, as shown in Figure 3. At each state, either party can either move closer to a full duplex connection, or abort the call into voice mail. It is this negotiation which reveals the strengths of flexible protocols.

In the first state after the called party activates Garblephone, *calling*, the caller is allowed to eavesdrop (Figure 4). The audio is processed, however, with a goal of allowing the eavesdropper to understand the conversational state of the called party (e.g. alone, on the phone, giving a talk, joking, listening to music) and possibly the identity of other participants if they are well known to the caller and even a few words, but rendering the overall conversation unintelligible. Although using a quite different algorithm, this is related in effect to the Smith and Hudson approach, with less concern about intelligibility of individual words. Additionally, no audio is transmitted without at least minimal action (activating the phone application) on the part of the called party.

Audio is garbled (at the transmitter) by a block shuffling scheme with cross-fading at block boundaries (Figure 5). Blocks consist of 50 to 100 milliseconds of audio. At each block time, one of the previous six blocks is selected at random for transmission, the least recently recorded block is deleted, and the just recorded block is added to the candidates for future transmission. Cross-fading mixes the audio from each block at block boundaries, to minimize the acoustic effect of the regularly repeated block boundaries (which sounds like an echo or reverb, depending on block size).[2] The block size is chosen to convey about a syllable's worth of speech (typically two to four phonemes). Larger block sizes increase the likelihood of intelligibility, while smaller sizes make the sound less speech-like and interfere with percep-

[1] Note that we could use similar techniques for an incoming conventional call, based on caller ID. Our home brew voice mail system does exactly this, doing a reverse directory lookup on the user's personal address book and then on a campus-wide online database.

[2] In order to cross-fade, at the time we transmit the Nth block, we actually select the N+1st block, as some of its data will be required for the fade

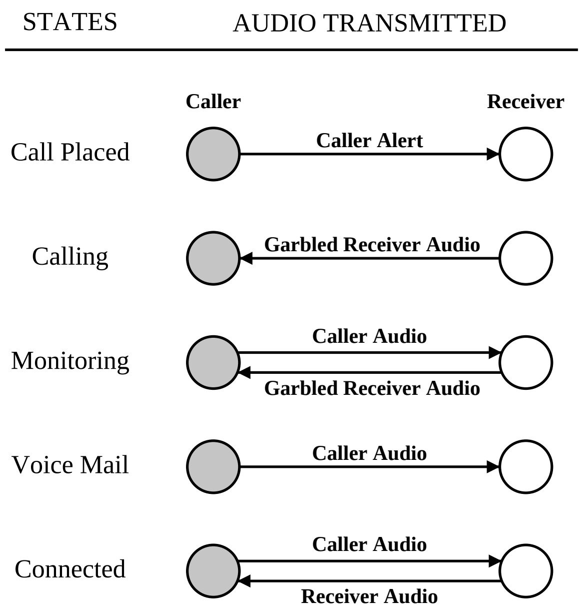

Figure 4: Audio transmitted in Garblephone states.

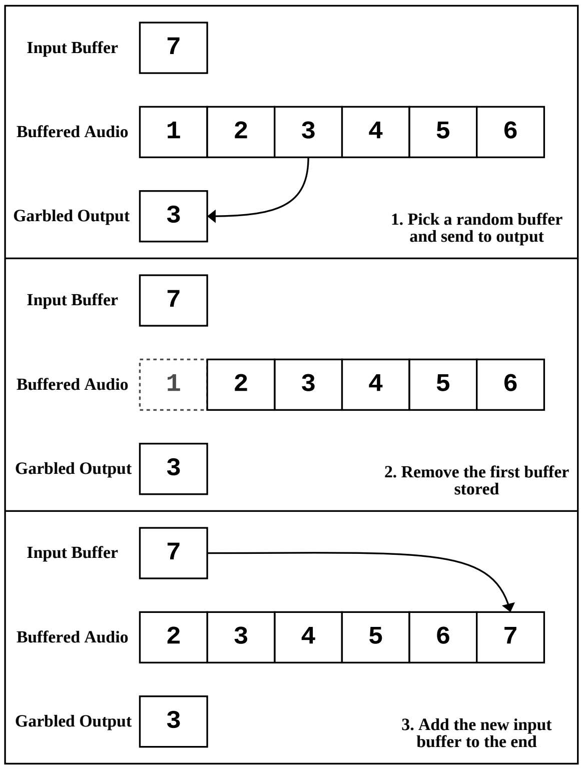

Figure 5: Garbling algorithm in Garblephone.

tion of the intonational patterns of the eavesdropped speech. The goal of this state is to allow the caller to gauge the activity level of the remote party; if he or she is busy and the call is not urgent, the caller can just drop into voice mail.

The called party may proceed to the next, *monitoring*, state by pressing up on the wheel button ("up" means "move closer"); the caller hears a tone indicating this transition has occurred. At this point, audio becomes full duplex, with the caller still hearing garbled audio but the called party now able to hear the caller clearly (Figure 4). This creates a first stage of call screening, a deliberately one-way clear channel, so the caller might say *"Answer, it's urgent!"*

If either party directs the call to *voice mail* (by pressing down, or "away"), the called party can also screen while the message is being recorded. The caller may only leave a message, but the recipient may still retrieve the call and connect.

Alternatively, the call may proceed from the monitoring state to the *connected* state if the recipient presses the up button. At this point a clear full duplex audio channel streams between the two Impromptu devices.

Impromptu's telephone channel is enhanced by audio processing in the network, the ability to extend existing call control protocols with new messages, and the ability to smoothly switch between full and half duplex audio.

CHAT – SEMI-SYNCHRONOUS CONNECTIONS

The second type of Impromptu channel involves some form of push-to-talk and is therefore not fully synchronous. Since users of a ubiquitous mobile device might wish to communicate by voice while busy, we attempted to build an audio equivalent to lower bandwidth text-based computer "chat" systems. After building an initial simple chat application, we redesigned it with more concern for providing efficient browsing of stored chat audio as well as a more effective background activity mode.

Text-based chat systems allow pairs or groups of users to exchange short text messages. Some systems (such as the old Unix "talk") show input letter-by-letter, while more recent software such as IRC chat allows editing in a private buffer before transmission. The programs just mentioned run in their own window or pseudo-terminal, while systems such as Instant Message or Zephyr pop up a window when a message arrives. Some of these systems also maintain group awareness, i.e., they indicate when participants enter or leave the chat program. There is a thin boundary between chat and email, where threads in a mailing list may generate traffic which is chat-like but generally with longer postings and somewhat more time between messages, i.e., less expectation of "same time / different place" of a chat.

Because typing is slow, participating in a text chat may sometimes be in the background, or use only part of one's attention. This is helped if the screen can hold multiple messages; since we read much faster than we type, a glance from time to time can keep up with a conversation. When activity picks up and there are postings from many participants, more attention is needed. Postings are likely to be even less formal than email, and hence require less editing, further supporting short, rapid interchanges interspersed with time spent waiting or attending to something else. An audio chat may provide similar functionality.

We attempted to design for several styles of voice chat. One style might be most suitable for a work group, where there may be bursty traffic which should be heard by all, such as the groups at NASA Mission Control described in [13]; our chat applications have not yet supported multiple simultaneous groups. Working at a university, however, we experience bursts of interactions among group members as they come and go, and occasional messaging spanning most of the 24 hour day. Support for a temporally distributed work group requires a "catch up" mode so a new comer can get up to date. Another style of chat is a decidedly background activity; one might catch up with family and friends while killing time awaiting a bus, for example. Audio is particularly suitable for real life situations when one's hands and eyes are otherwise busy.

Our initial audio chat was simple. Each participant could record messages using a push-to-record button, and after recording, the message would be sent to all others. New messages played automatically. When joining a chat, one could optionally hear in sequence all the audio recorded to date, but had to press "down" after each recording in order to hear the next. While in this catch up mode, new messages were not played, but sounded an alert. If chat was not the active application, an iconic alert would sound to indicate that a new message had arrived; to hear it the user had to return to chat and go back into catch up mode.

This method may be satisfactory if chatting is infrequent, but if participants are talking back and forth it is not responsive enough. After one person finishes recording, he or she must wait for the other party to hear what was just transmitted and finish recording a response before hearing anything. Because nothing is transmitted until recording is finished, maximum channel utilization is 50%. Channel utilization exceeds real time, as recorded messages are sent upon completion and require less time in transit in the network than for recording; a participant might hear many messages in a row without pause. Another advantage of this method is that no floor control was necessary, though messages could arrive out of sequence, and overlapping chat contributions may be repetitive.

Despite its more efficient use of network capacity, the lack of responsiveness of the original chat was frustrating and led to a new chat application, **TattleTrail**, which more gracefully merges catch up and synchronous messaging modes. The main goal was to switch from completely asynchronous messaging (record then send) to synchronous multicast mediated by greedy floor control (no involuntary interruption).

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 6: Sample TattleTrail chat activity displaying different modes of user attention.

It also facilitates running the chat as a background activity with more informative alerts. TattleTrail's audio icon consists of musical tones mixed with a jumble of laughing children's voices.

Modes

TattleTrail has three modes of user attention. When a participant first joins an active chat she or he is put into asynchronous **catch up mode**, described more fully below. Once caught up, the application automatically enters synchronous **push-to-talk mode**, similar to Nextel's Direct Connect (a virtual walkie-talkie with floor control). The user pushes the talk button, which responds with a short beep if the floor has been granted. If so, one's speech is peer-to-peer multicast to all other chat participants, and also stored as a timestamped file in the chat server. A chord with a sharp attack and slow decay indicates that the floor was denied, but if the user holds the button down, control will be granted when available.

Each user may independently enter and leave TattleTrail. Upon returning, participants are put into catch up mode to hear those messages they may have missed while attending to other applications, and then put back into push-to-talk mode (Figure 6). A user leaving TattleTrail transits to **background mode**, in which awareness of the chat is limited to alerts, as described below.

Browsing

When one first joins a chat, or rejoins it after spending time in another application or perhaps being off the network, TattleTrail's catch up mode provides for rapidly scanning the recorded chat. Browsing makes heavy use of the SOLA method of time compressing speech for playback without increasing its pitch [9]. SOLA uses the autocorrelation function to determine the periodicity of a windowed sample of speech and, assuming it is voiced, removes whole pitch periods using an "overlap and add" method which is essentially a cross fade. But drawing in part on the perspective offered by Arons' SpeechSkimmer [2], our user interface attempts to control not just the raw speed of playback, but rather the rate at which salient information is heard. In this section we present a browsing method highly tuned to the audio chat genre.

Beyond approximately twice normal speed, time compressed speech loses intelligibility rapidly, and other methods must be employed.[3] SpeechSkimmer included a strategy of pause removal, but more importantly used pauses to delineate the introduction of "new" or "interesting" portions of a lecture; at high playback rates it would play some seconds of speech at the beginning of one portion and then jump to the next portion. Audio Notebook [12] took this idea further, also using the lecture genre. Stifelman used both pauses and pitch to segment recordings based on discourse theoretic models of speech phrases; listeners could jump to the next segment associated with a new topic, and the phrase boundaries gave a better indication of how much speech to play. NewsComm [10] used speaker changes to segment audio from a different genre, radio newscasts, and the user could jump to the next speaker; a speaker-differentiation algorithm first processed the recording to find speaker boundaries. Hindus' xcapture [4] recorded selected portions of telephone conversations based on turn-taking and use of an extra local microphone at one end to determine which party was speaking.

Different aspects of speech structure can be used to enhance browsing at high speed, depending in large part on the characteristics of the recorded genre. These lead to different strategies for mapping user interface controls to audio playback strategies, especially for selecting which audio will be played when some must be discarded. The chat genre can be characterized by alternating turns of talk between two or more conversants, relatively short turns of talk, and somewhat bursty interactions. There may be long periods of inactivity on a channel, but once activity starts, there is likely to be a series of transmissions which are highly correlated. Walkie-talkie or emergency band radio conversations manifest similar behavior.

TattleTrail utilizes the staccato nature of the chat channel to cluster recordings into "bursts", which are sets of turns with less than 30 seconds of silence between adjacent turns. In the simple case, the user needs to browse a small number of bursts, and simple speed control is used; if the user has been away for longer, the segmentation into bursts becomes more apparent with high speed browsing.

When the user enters browse mode, TattleTrail plays the unheard turns, in order. Pressing the "up" button increases playback speed; as long as the button is pressed, speed gradually increases. Audio keeps playing as it is sped up; there is no "pause" control while browsing. If the user wants to skip browsing, double clicking "up" jumps to the end and switches into synchronous mode. When the user presses "down" the playback speed slows down. Although SOLA can be used to play back a recording at less than normal speed, TattleTrail does not; when playing at normal speed and the user presses "down," playback switches direction and starts playing backward. Continuing to press the "down" button makes backward playback go faster and faster; double clicking it jumps to the beginning of playback and it starts playing forward, at normal speed.

[3]This number is very rough, because it depends on the content of the recorded speech and exposure of the listener to time compressed speech.

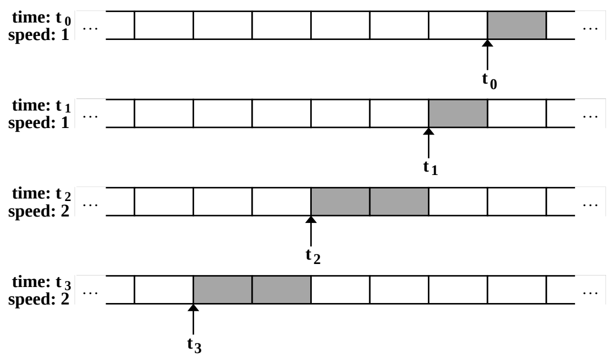

Figure 7: Playing backward. The audio is divided into segments of 4 seconds. Segments are played at each iteration (the gray boxes), then the pointer is set back to the previous segment for the next iteration. Note at times t_2 and t_3, the playback speed is twice as fast, so the algorithm jumps back two segments in order to play 4 seconds of time compressed audio.

"Playing backward" is accomplished using the SpeechSkimmer technique shown in Figure 7. If samples were merely playing in reverse order, the speech would not be intelligible. Rather, we break the recording into segments of 4 seconds, play each segment normally (possibly also time compressed), and then play the previous segment. We repeat this process until we have played the very first segment in the file; once back to the beginning, forward play resumes. During reverse playback, a single click "up" reduces the speed at which reverse playing occurs, while a double "up" click picks up forward playback at normal speed from the beginning of the current segment. This last technique, allowing the user to revert to normal playback as soon as some interesting speech is heard, was also motivated by SpeechSkimmer user studies.

If the user has been away from the chat for some time, a number of bursts will have occurred, and a higher level of skimming is needed (Figure 8). To help convey the temporal structure of the chat, after each burst of turns is played, a short "tick tick" watch sound is played, to indicate the passage of time. As the user speeds up playback, once a speed

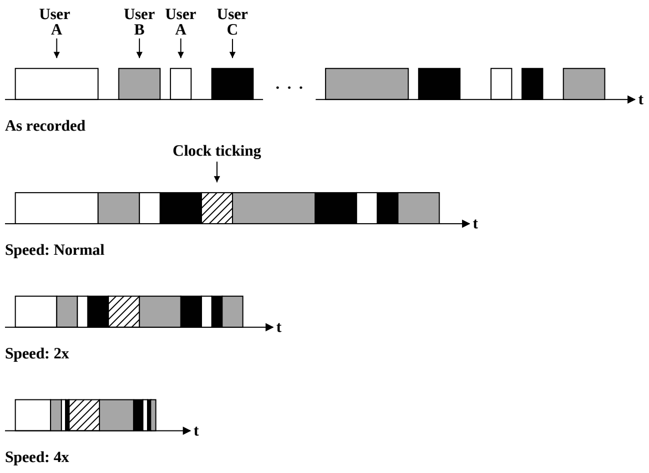

Figure 8: Browsing chat messages at different speeds. Two different bursts are shown above. When browsing, bursts are separated by a clock ticking sound. Note the non-linear scaling at speed 4x.

of 2.5 times normal is reached, we take advantage of the turn structure to present a non-linear time compression technique. For each burst, the first turn of the burst is played at twice normal speed, and the remaining turns are played much more quickly, to obtain an effective compression of 2.5 (or higher, if the user continues to select for a higher playback rate). The goal of this presentation technique is to introduce each burst with some intelligible speech which hopefully sets the topic for the burst.

Alerting

Alerting includes any audible indication of activity by inactive Impromptu applications. While the user is active in TattleTrail, he hears only chat audio, either in browse or realtime mode. Upon exiting, however, he remains in the chat and is alerted to new chat content even while listening to another application. This is similar to using two windows at once in a GUI, and is the only situation in which Impromptu mixes audio sources.

The initial chat prototype generated alert sounds when someone joined or left the chat, but gave no indication of traffic in the chat. TattleTrail does so, at two different attention levels. In dominant mode, the chat channel has infrequent activity but should become the focus of attention when activity occurs. As a background activity, alerts indicate that some activity has occurred but do so in a less intrusive manner. Currently one leaves the application and goes into primary attention alerting, and after a number of transactions in which the channel alerts but the user does not respond, it slips into background alerting.

In dominant mode, each new chat message is played clearly. Whatever application's audio is being played at the time is faded down for the TattleTrail alert, consisting of the first few

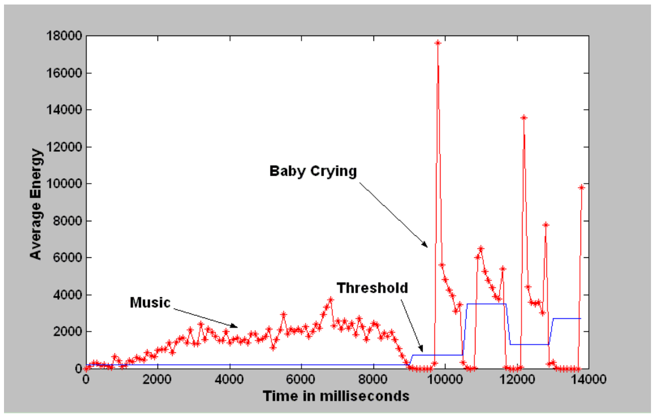

Figure 9: Baby cry detection. In response to the music, the base threshold is increased. Noise and quiet periods trigger the detector after the third cry.

notes of the TattleTrail audio icon, followed by the new chat contribution. At the end, the interrupted application audio fades back up. At this point for ten seconds the push-to-talk button becomes active, and the user can respond. While this button is depressed, the application audio is muted. If the floor is granted, the user is speaking to the chat group synchronously.

When in background attention mode, the chat alert is less obtrusive. Instead of interrupting the active application's audio, it is mixed in, beginning with the audio icon cue and followed by the chat audio, all at an attenuated level. This provides some cues as to activity on the chat channel, but it may be difficult to understand the actual content. In this mode immediate push-to-talk is not available (though further evaluation may indicate this is confusing); to speak one rejoins TattleTrail and may either catch up or transmit.

EVENT-TRIGGERED CONNECTIONS

The ability to do processing and storage in the network or associated audio device enables a third style of voice channel supported by Impromptu: one which the channel goes active not because someone decides to communicate, but because some external event occurs. In this case, the client generates an alert, an iconic sound so as to be consistent and avoid confusion. The user may then activate the application, which sets up a full duplex streaming audio connection. Alternatively, the user may activate the application at any time, which then provides a "telephone" channel.

BabyMon

Conventional baby monitors continuously transmit. They use radio spectrum all the time, and transmit needless distracting sounds such as the neighbor's lawn mower or chain saw (fortunately babies seem able to sleep through anything).

BabyMon is an application which runs on an iPaq in lieu of Impromptu; the user selects the application and leaves the mobile unit wherever the baby is sleeping. BabyMon then contacts the application manager and the service becomes available to Impromptu users. BabyMon continuously digitizes audio and analyzes it for patterns characteristic of crying babies. A cry is characterized as a fairly loud sound followed by silence or low energy sound while the infant inhales. BabyMon's detector requires at least three cry cycles, where a cry is defined as between 400 and 2000 milliseconds of sound above a threshold, punctuated with 200 to 1000 milliseconds of quiet between cries (Figure 9). The threshold, or background level, in turn is allowed to be slowly varying; i.e. the lawn mower won't trigger the monitor but the baby will have to cry more loudly to outweigh its noise.

When BabyMon detects a cry, it alerts Impromptu, which plays its audio icon, a single loud baby cry. If the user then activates the application, BabyMon switches to a full duplex audio connection so the user can hear the baby and speak to it to attempt to calm it down (not very effective with infants, but sometimes useful with two year olds). If the baby continues to cry but the application is ignored, it will continue to alert from time to time.

WatchDog

Like its namesake, **WatchDog** responds to loud noises. It also uses a slowly varying background audio level adaptation, and alerts with a bark. Unlike BabyMon, WatchDog stores the digitized audio which caused it to trigger, as that might be a short and transient sound like glass shattering. When the Impromptu user activates, WatchDog first plays the stored sound, so the user can hear the cause of the alert, and then switches to a full duplex audio connection.

Although it may seem unlikely that one would try to converse with thieves in one's house, this scheme could equally well be used in a home situation around a family intercom system such as that described in [7]. In this case the user would be more likely to ask *"Are you all right?"* or *"What was that?"*

Monitoring applications reveal another advantage of IP. Aside from firewall issues, one can easily monitor events at a remote location, such as listening in to home while at work.

CONNECTIONS AND TRANSITIONS

We have described three classes of computer mediated audio channels provided by Impromptu: telephone (synchronous), walkie-talkie (semi-synchronous) , and monitor (event triggered). Part of the contribution of Impromptu is enabling all three types, and this allows us to make two generalizations. The first is the value of a variety of channels in a single mobile device. The second is that the most intriguing novelty in these channels is their internal transitions. The value of a mobile implementation is assumed, given the worldwide popularity of mobile telephones.

Connections

We began this paper with an argument that computer- mediated voice communication should exhibit the same breadth of channel type as computer-mediated text communication. Because users take advantage of multiple text channels, we believe the channels must be serving different needs. Is the same true of voice?

A telephone channel requires significant attention; in a two-person conversation, silences speak as effectively as words. The ability to interrupt on a full duplex channel reinforces the tendency to treat the channel with much the same attentional resources as a face-to-face conversation. Note that we did not implement an always-on full duplex channel, as was done in Thunderwire [1], in large part due to privacy concerns around the combination of always-on and mobile, and in part due to the difficulty of effective microphone placement for an always-on mobile device.

A half-duplex chat-style application lends itself more easily to background or occasional use, at the price of remembering to push to talk. Push-to-talk does, however, limit extraneous environmental audio and keeps foreground activity, such as conversation with someone in the hallway, out of the channel. This extraneous sound was sometimes a distraction in Thunderwire. Eliminating it lends more value to the chat log, and makes the channel more amenable to groups distributed in time as well as space.

A monitoring channel is simply effective use of network and attentional resources. There is no need to continually transmit uninteresting sound if a decision can be made at the source as to its value. Similarly there is no need to attend to the channel, and an iconic sound (instead of streaming the triggering sound) resolves ambiguities (was that sound local or remote?) when the trigger occurs. WatchDog's ability to replay the sound triggering transmission solves the problem of missing interesting transients.

Transitions

Transition refers to change of state of audio transmission within or with respect to an application. The most simple transition, common to all these applications, is when the user scrolls through applications with the wheel and hears the auditory icon for each in turn.

Transitions also occur with alerting. The most simple alerts, e.g. the telephone ring of Garblephone or the barking dog of WatchDog, simply indicate activity in the application. More information is conveyed with alerts which vary with application content, e.g. the caller's voice announcing an incoming call from another Impromptu user, or the actual message sent to the chat application. The latter can effectively place the application automatically in the foreground (auto-activate). Or, the chat alerts can serve as quiet background awareness of the existence of activity in the channel.

In addition to its two alert modes, the possibly repeated transitions between catch up mode, listening to prior chat recordings with several speech compression techniques, and synchronous audio multicast make TattleTrail a much more exciting application for users. And the transitions through a series of garbled or clear eavesdropping states change call setup from a binary action (answer or ignore) to a negotiation between caller and recipient, with several possible outcomes.

CONCLUSION

The flexibility of IP-based networking and a software architecture which allows rapid development of distributed applications with ad hoc messaging on top of IP enables the range of channel types which have been developed to date under Impromptu. Synchronous, asynchronous, and event-triggered channels fill different communication needs and have a variety of transmission, storage, and audio processing requirements. By placing all of these in a single device, we hopefully change the focus from the device to the task, communication, and allow users to choose the appropriate channel.

At the same time, creative management of the channel audio content and sound effects between multiple channels, and appropriate transitions and audio cues between modalities within applications make for a more compelling user experience. Particularly in a mobile environment, user attentional resources are often scarce and screens may be out of sight; the auditory medium must afford communication rather than hinder it, in a manner appropriate to the application selected.

Although much tuning and convincing group user experiences are still required before a system such as Impromptu should be made widely available, we currently have a working prototype which hopefully raises useful challenges while offering attractive communication possibilities. Significant iterative design has gone into the interfaces described in this paper.

ACKNOWLEDGMENTS

Elina Kamenetskaya wrote the telephone gateway and parts of Garblephone. Mike Jacknis, Stefan Marti, and Nitin Sawhney helped implement and evaluate an early version of the garbling algorithm. Ivan Chardin assisted with the Baby Monitor application. Barry Arons developed some of the browsing techniques reused in reviewing chat history. Eric Hulteen made some small but powerful suggestions for speech recognition feedback. This work was supported by the Digital Life and Information:Organized research consortia.

REFERENCES

1. M. Ackerman, D. Hindus, S. Mainwaring, and B. Starr. Hanging on the ’Wire: A field study of an audio-only media space. *ACM Transactions on Computer-Human Interaction*, 4(1):39–66, March 1997.

2. B. Arons. SpeechSkimmer: A system for interactively skimming recorded speech. *ACM Transactions on Computer-Human Interaction*, 4(1):3–38, March 1997.

3. P. Dourish and S. Bly. Portholes: Supporting awareness in a distributed work group. In *Proceedings of Human Factors in Computing Systems (CHI’92)*, pages 541–547. ACM, 1992.

4. D. Hindus, C. Schmandt, and C. Horner. Capturing, structuring, and representing ubiquitous audio. *ACM Transactions on Information Systems*, 11(4):376–400, October 1993.

5. 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.

6. K. Lee. Impromptu: Audio applications for mobile IP. Master’s thesis, Massachusetts Institute of Technology, September 2001.

7. K. Nagel, C. Kidd, T. O’Connell, A. Dey, and G. Abowd. The Family Intercom: Developing a context-aware audio communication system. In G. Abowd, B. Brumitt, and S. Shafer, editors, *Ubicomp 2001: Ubiquitous Computing*, Lecture Notes in Computer Science Series, pages 176–183. Springer-Verlag, 2001.

8. L. Nelson, S. Bly, and T. Sokoler. Quiet calls: Talking silently on mobile phones. In *Proceedings of Human Factors in Computing Systems (CHI’01)*, pages 174–181. ACM, 2001.

9. 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.

10. 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.

11. I. Smith and S. Hudson. Low disturbance audio for awareness and privacy in media space applications. In *Proceedings of the ACM Conference on Multimedia*, pages 91–97. ACM, 1995.

12. 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.

13. J. Watts, D. Woods, J. Corban, E. Patterson, R. Kerr, and L. Hicks. Voice loops as cooperative aids in space shuttle mission control. In *Proceedings of Computer-Supported Cooperative Work*, pages 48–55. ACM, 1996.