Impromptu: Managing Networked Audio Applications for Mobile Users

Chris Schmandt, Kwan Hong Lee, Jang Kim[*] , Mark Ackerman [†]
Speech Interface Group
MIT Media Laboratory
20 Ames Street, Room E15-327
Cambridge, MA 02139
{geek, kwan, jangkim, ack}@media.mit.edu

ABSTRACT

This paper discusses the software architecture of Impromptu, a mobile IP-based audio computing platform, with an associated set of network-based applications and services. Impromptu merges the communication properties and universal mobility of the telephone with the multi-tasking and open protocol world of the handheld PC. Its supporting architecture handles multiple streaming audio applications, provides speech services for consistent audio user interfaces across applications, and enables user management of these varied applications running simultaneously.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Distributed Systems; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Audio input/output*

General Terms

Human Factors, Management, Performance, Design

Keywords

Multi tasking, audio interface, mobility, Voice over IP, WiFi, audio applications, telephony, architecture, speech interface

1. INTRODUCTION

Impromptu is a mobile, IP-based audio device, with an associated set of network based services and applications. It supports multiple voice services, such as radio, news (text), music (MP3), telephony (synchronous), chat (asynchronous) and baby monitor (event triggered). Currently implemented on an iPaq with an 802.11b wireless LAN card, Impromptu is designed for highly mobile use. Its user interface does not use the display; user input is by either speech recognition or button presses. Both the Impromptu user interface as well as its application architecture are designed to support, in the audio domain, many interaction techniques of conventional window systems.

Impromptu merges the communication properties and universal mobility of the telephone with the multi-tasking and open protocol world of the handheld PC. The surge of mobile telephone use has clearly shown that communication is what counts for mobile users, and voice is where the action is. Conventional telephony has the advantages of being highly mobile, extremely connected, and usable in the midst of other activities. However, the standard telephony model is also extremely limited, since it is stuck in a single-tasking model: only one application (generally, a telephone call) can exist at a time. PC-style computers obviously are multitasking and support more open networking protocols, but lack the same level of mobility, ubiquity, and ease of use. Existing “hybrid” products basically glue a mobile phone onto a PDA. Impromptu draws on the features of both, and here we examine the benefits, and limitations, of an audio-oriented multi-application model based on mobile IP.

We hoped this would result in two advantages:

- This platform could allow one to run multiple audio applications because of its use of IP and the standard capabilities of digital environments. This would change the basic model of telephony that has existed for the last 80 years, allowing one to simultaneously run a variety of more sophisticated applications and to create a new kind of audio environment.
- This platform would also allow us to examine the use of audio in a mobile environment. Our work here extends into even more hybrid environments that would allow visual and audio interfaces simultaneously. However, because of our interest in understanding how audio might be of advantage in a mobile environment, we pushed the audio interface in the work considered here.

This paper, then, discusses Impromptu, an audio-only platform for ubiquitous computing. The paper describes Impromptu's hardware platform, sample applications, and software architecture and services. We describe the sample applications in order to show the power of creating a mobile digital audio environment, and we describe the software architecture to indicate the architectural changes required to

[*] Jang Kim is currently affiliated with Oracle Corporation

[†] Mark Ackerman is currently affiliated with University of Michigan

MobiSYS'04, June 6–9, 2004, Boston, Massachusetts, USA.
Copyright 2004 ACM 1-58113-793-1/04/0006 ...$5.00.

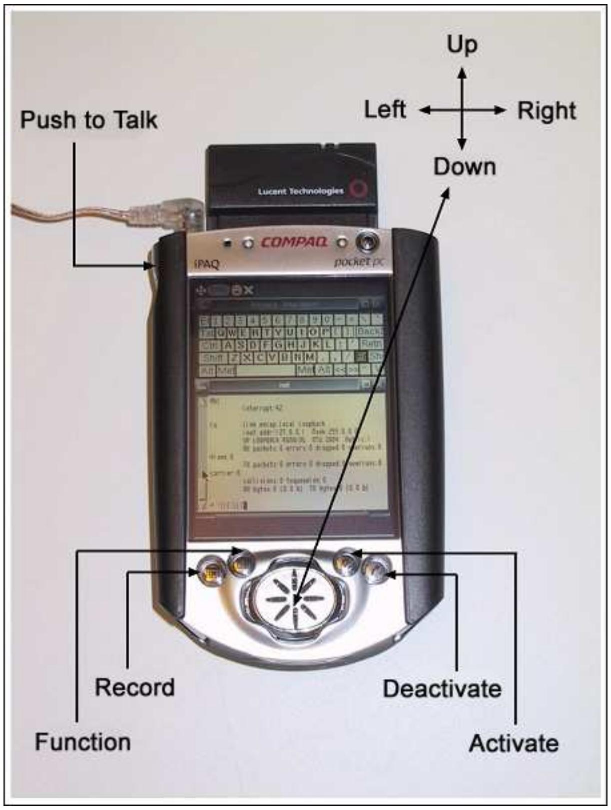

Figure 1: The iPaq Button Mapping.

support integrated audio and telephony. This architecture must also support speech and audio in the user interface, and, much like a desktop window manager, manage input and output for multiple simultaneous audio applications.

2. IMPROMPTU

Impromptu is implemented on an iPaq PDA running Linux, with an attached 802.11b wireless LAN card. Impromptu consists of a number of audio-oriented applications, and it does not use the display.[1] User input is via speech recognition, using a network-based server running IBM's Via Voice recognizer, and the various buttons on the iPaq, as shown in Figure 1.

Impromptu applications play sounds or text through speech synthesis. Currently, Impromptu is an audio-only system; applications do not have access to the display. Because of speech's constraints, only one application is active at a time. However, a central theme of Impromptu is that users will have access to many audio applications on a single device, that some of these applications may run asynchronously in the background, e.g. awaiting a phone call, and that the display is not used for input or output. Therefore, an inactive application can trigger an alert sound, such as when an incoming phone call alerts over the radio or MP3 music player. When an application alerts, the user must take action to activate it if desired.

Users select an application by speaking its name, or by using the "wheel" control in the center bottom of the iPaq to scroll through the applications by pressing the left or right buttons. As each application is selected in turn, it plays a distinctive audio icon, a sound meant to evoke the concept of that application. Although Impromptu includes a default set of application sounds, users would most likely wish to configure their own, so the sounds are specified by URL.

Several iPaq buttons have global meaning. Speech recognition is initiated by pressing the *push-to-talk* button (which immediately stops any application audio output). The *activate* button allows for activation when a background application alerts. The *record* button causes whatever audio that is currently being played by Impromptu to be saved in a file for the recorder application, without needing to bring the recorder explicitly to the foreground.

Each application may have its own speech recognition vocabulary, which is enabled while it is active (Figure 2), and may use many of the iPaq buttons for functions of its own choice. In general, each application uses "up" and "down" on the wheel to mean "next" or "previous", in a list of the application-specific content it is providing.

The next section describes the current applications in order to ground the later discussion of architecture and user services.

3. APPLICATIONS

Central to the motivation behind Impromptu is that a single mobile audio device can provide a number of different applications. Each application has its own user interface and functionality, but they share many characteristics. Impromptu applications can be divided into audio content access, personal information management, and communication channels. Exposing the reader to the range of these applications sets the stage for the architecture and user interface management issues which form the core of this paper.

There are three types of applications, each discussed in turn. The first group of applications provide different types of audio content.

3.1 Audio Content

The **music** application is an MP3 music player. It supports sequential and random play, and the user can always skip to the next or previous song in the user's play list. There is some support for requesting a particular genre or artist, but not a full scale speech jukebox interface. The user can ask about the current song; the artist and title spoken using speech synthesis. The audio icon for the music player is a short guitar riff.

Radio is exactly that. Currently a single channel is picked up on a tuner and digitized to be streamed to Impromptu users. There are a number of sources of streaming IP-based radio but we have not incorporated them due to audio transcoding issues. There is no memory in the radio application, although previously the first author built a system, Radio with a Memory, which allowed a user to jump between two stations. This system would store the unattended channel and when the user switched back to it, play with rapid time compression until the buffered audio was played out, then slow down to real time. Its audio icon is the sound of stations fading in and out as a tuner knob is turned.

AudioBook is a books-on-tape application. It accesses digital audio from a variety of sources; a previous system

[1] The display is actually used for input only when entering telephone numbers for conventional phone calls. For this exception only, the input is interpreted by Impromptu components as client button presses.

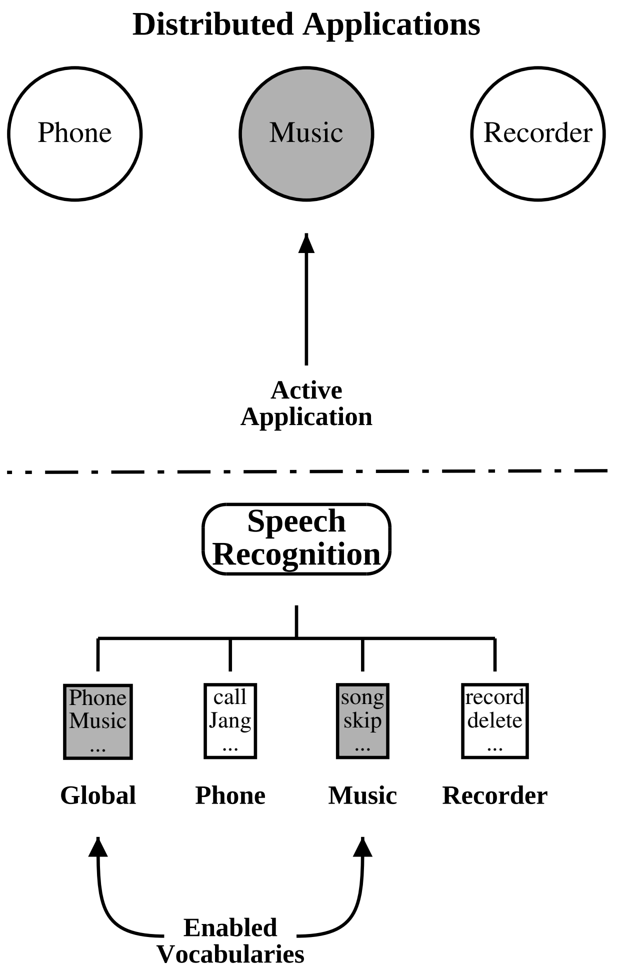

Figure 2: Recognition vocabularies. The recognition engine stores each application's vocabulary, and dynamically enables them when they are active. The Music application vocabulary is active in this case. The global vocabulary is always enabled.

also digitized and saved certain public radio programs. AudioBook keeps "bookmarks" so when a user returns to a recording, it restarts just before where the user last left off. Its icon is the sound of pages turning.

News reads headline news stories scraped from the Yahoo news web pages. Using speech synthesis, it reads headlines. Using either voice or buttons, the user can skip to the next headline, or ask to hear the entire story. Although we do not think users will want to hear a large amount of news in this manner, it does effectively demonstrate the speech synthesis capability of the speech service. Its icon is a recording of a person shouting "yahoo!", to credit the source.

3.2 Personal Information Management

The second group of applications lets one manage one's own audio data. The **recorder** is a personal audio to-do list, with a simple user interface and an additional function. In normal use, one activates the recorder, and pushes the record button to add an item to a single list. The list is played using the "up" button on the iPaq wheel control, by saying *"play", "next"*, etc. We left this application simple as our group previously demonstrated a much more sophisticated handheld audio note taker [12] which used speech recognition to create and navigate between lists and audio time compression to play lists rapidly in browse mode.

In addition to this normal activation mode, the recorder can be accessed at any time by pressing the leftmost of the iPaq buttons, the *record* button, just as if recording while the application is active. In this mode, any audio which is being played by Impromptu is also forwarded to the recorder, so it can save pieces of phone conversations, radio, etc. This mode of operation might seem to violate our paradigm of a single active application, but actually the recorder application is actively recording audio received from the client in the background, which reveals the utility of having multiple applications running simultaneously. The audio icon for the recorder is a sound like a snicker, somewhat in reference to the privacy concerns about any digital audio recording.

Another application which may be considered "personal" information management is **BabyMon**, a digital baby monitor with some intelligence. BabyMon is designed to replace the analog radio baby monitors which are ubiquitous in households of any affluence with a newborn. BabyMon is software which runs on the normal Impromptu hardware configuration; i.e. it is just another program which can run on the platform. BabyMon detects baby cries, which are identified 400 to 2000 milliseconds of loud sound punctuated by 200 to 1000 milliseconds of relative quiet (while the baby inhales for the next cry). The background noise threshold for cry detection is allowed to slowly vary, to compensate for background noise levels.

BabyMon is the first application described which runs independently in the background and can decide it has reason to go active. When BabyMon has detected three cries, it alerts with its audio icon, a sound of a baby crying. If the user activates the application, a full duplex audio channel is established between Impromptu and the baby monitor. Note that we always use the recorded cry as the alert sound, because if the baby monitor falsely detected a cry when there was none, or if the cry was soft, it might be confusing to hear this unknown sound in the midst of whatever audio was being played.

WatchDog is a burglar alarm which operates in a manner similar to BabyMon. When a loud noise (above an adapting moving window threshold) is detected, WatchDog alerts with (surprise!) the sound of a barking dog. When the user activates the application, it first plays the stored audio which triggered the alert, as a sudden brief sound could have ended by the time the user activates, and then goes into full duplex mode.

This could also be a function of a home intercom system, such as described in [5], where the sudden noise could be associated with someone falling or dropping a dish, and one would wish to ask *"Is everything OK?"*. Again, this could be a useful application for remotely monitoring home from the office or car.

3.3 Communication Channels

From some points of view BabyMon and WatchDog can be thought of as event-triggered communication channels, because they alert when their programmed event occurs and then can become full duplex channels. Impromptu also supports normal and enhanced telephony, and an audio "chat" application with a conversational log. The third group then consists of these two rich applications. We left many details omitted here; full discussion may be found in [8].

Garblephone is the Impromptu telephony application; it operates as both conventional and enhanced telephone. As a conventional telephone, the user enters a number (or receives an incoming call on a pre-assigned number) and the Garblephone application, running on a remote PC, connects to the public switched telephone network via an analog telephone line interface unit (Computerfone, from Suncoast Systems). Once the call is dialed, full duplex audio is streamed across the IP network, converted to analog, and routed onto the phone line. An incoming call works in a similar manner, with the server generating an alert (the sound of a ringing phone) and answering the call and connecting audio if the Impromptu user activates.

Calls placed between Impromptu users demonstrate the novel applications which can be developed when applications are released from the limitations of conventional telephony. Garblephone allows increased negotiation for both parties while a call is set up. A known caller is announced by name with the alert, and if on a list of trusted callers, the caller can eavesdrop on the called party. The audio is garbled, however, by an algorithm which randomizes the order of recently recorded 100 millisecond blocks of sound. This algorithm allows more intelligibility than that described in [11] because we desire to reveal enough information about the state of the called party to help the caller decide whether to interrupt or go to voice mail; this includes the general character of the conversation (serious, light, joking, etc.), and possibly the identity of other conversants. The chosen block sizes preserve enough syllables to allow some speaker identification and to convey intonation.

The calling party may abort the call to voice mail, or the called party may "move closer" (using the "up" button) to hear the caller in the clear (Figure 3). This is to allow the caller to state his or her reason for calling only if the called party desires to hear it. If the called party does not send the call to voice mail, the next stage of closeness is a full duplex "telephone" connection, Impromptu-to-Impromptu.

A synchronous audio connection is the most demanding for the network and is also very much a foreground activity for the conversants. An alternative is an audio equivalent of computer-mediated text "chat". Our initial chat prototype modeled those applications closely; the user recorded a contribution and then submitted it to the chat for others to hear. A much more satisfying interaction uses a "walkie-talkie" approach, a push-to-talk mode in which all parties in the chat hear one person while he or she talks. This became the core of the current chat application, **TattleTrail**.

Like its predecessor, a TattleTrail user joins a chat in "catch up" mode, which allows for a quick scan of previous chat content. Time compression using the SOLA algorithm [6] allows the browsing user to interactively speed up and slow down previous messages. All chat contributions are clustered into "bursts" of back-to-back talk, for ease in navigation. Once the user has caught up, he or she jumps into synchronous mode, hearing others speak in real time. To talk, the user presses a button, and hears either a beep or a "no" chord (similar to the equivalent default sound in Windows) which indicates whether the floor has been granted; only one user may have the floor, i.e., transmit, at a time.

As the user enters and leaves the TattleTrail application, he returns to catch up mode and then becomes synchronous each time. Additionally, while TattleTrail is inactive, but if it has been activated in the past, it uses as an alert a portion of its own audio icon (a chime followed by the babble of children's voices) appended with the audio just sent by a chat participant. These two modes are designed to allow the Impromptu user to participate in the chat in the background or from time-to-time. Figure 4 depicts a possible TattleTrail usage scenario.

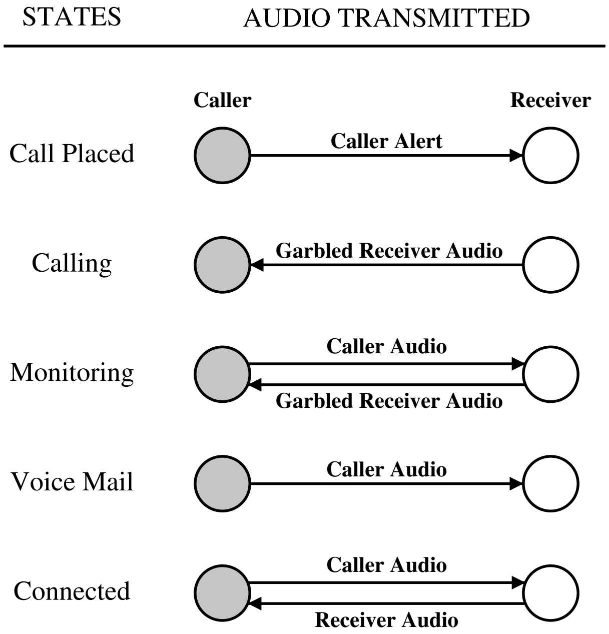

Figure 3: Audio transmitted in Garblephone calling states.

4. ARCHITECTURE AND NETWORKED SERVICES

Impromptu is designed to consist of a thin mobile client which communicates with network services, most importantly an application manager, speech recognition and synthesis services, and a number of distributed applications (Figure 5). Because the applications all involve either fetching audio or text content from the network or networked storage, or communicating with other mobile clients, they can reside only in the network. This forces the architecture to manage streams of audio in multiple directions and between multiple software components, in real time. Because audio has unique qualities, this also forced certain considerations on the Impromptu architecture and capabilities.

The most important Impromptu component is the **application manager**, which intervenes between client and applications. Although the application manager could reside on the mobile client, its marshalling role argues strongly for its implementation as at least a separate process. Being able to run elsewhere offers many advantages when the client becomes disconnected, changes networks, or must respond to differing network quality of service, e.g. by substituting in codecs of higher compression ratios. As Litiu et al point out in [3], such an application manager can "park" connections

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

to keep them alive but only slightly active by controlling them on behalf of a temporarily disconnected client.

The application manager (one per user) plays a crucial role in management of Impromptu resources, and merits a detailed look. The application manager functions in part as the equivalent of a window manager, in terms of managing input/output resources and changing focus between applications. When the application manager starts up, it currently contacts a simple **lookup service** and registers its available status. The lookup service stores all registration information, and is the initial point of contact for Impromptu components. When an **application** starts up (generally *not* one per user), it consults the lookup service to find the addresses of all running application managers (i.e., those which are currently engaged with Impromptu clients), and then registers its online status with each application manager. During registration (Figure 6) the application specifies the location (URLs) of its distinctive audio icon (to be played when it goes active) and its specific speech recognition vocabulary to the application manager. The application manager uses the **speech service** to load the application's vocabulary into the speech recognition engine, and notifies the client about the new application so that socket connections between the two can be established for streaming audio.

The speech service handles all speech recognition and text-to-speech services for the client over the network. These services could be implemented on the iPaq, and in fact the current iPaq product supports both, but at the time this project began speech recognition was not available. Because it is important to separate the concern of details of speech recognition from each application, the speech services should be separate processes wherever they reside. Since mobile processing is still significantly more expensive than server based, and speech services can be shared by multiple clients, providing these services in networked servers is a cost effective solution.

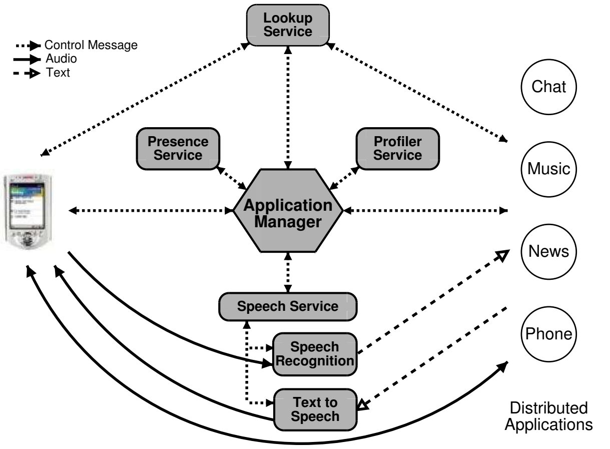

Figure 5: The Impromptu architecture. Note that different types of connections are made for audio, text, and control channels

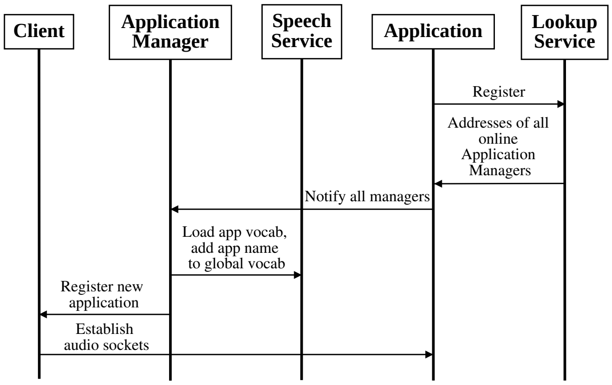

Figure 6: Application registration.

When a new Impromptu client starts up (Figure 7), it registers with the lookup service and uses it to find an available application manager process. Once a message channel is established between the client and the application manager, resource allocation begins. The application manager registers with the presence, profiler, and speech services for the client. The **presence service** keeps track of the user's subscribed "buddies", while the **profiler service** keeps track of the subscribed applications and profiles the user's activities by monitoring the application usage. Once the services have been activated, the application manager provides the client with the speech service information for establishing an audio socket to the speech service. Additionally, in a process similar to that described above, the application man-

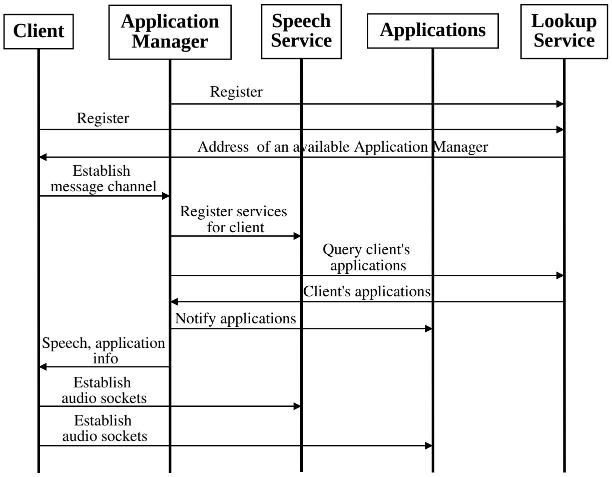

Figure 7: Client registration.

ager contacts the lookup service to obtain addresses for the registered applications to which that particular Impromptu user is subscribed.

While a user is running Impromptu, the application manager collects user input and either acts on it (for example, the user requests to change application) or forwards the user input to the active application. If the user changes applications, a *deactivate* message is sent to the current application and an *activate* message is sent to the new application. The *push-to-talk* button is special; when it is pressed the client stops playing audio and a message is sent to the speech service, to cause it to start reading audio from Impromptu (Figure 8). The speech service performs recognition and returns the result either to the client or to the application, depending on which vocabulary contains the recognized word.

Note that although the application manager (with the help of the speech service) acts as a dispatch mechanism for user input, this architecture does not interpose it in the audio paths. To do so would add delay to the audio, and possibly greater chance for lost packets; this would have a particularly negative impact on the synchronous telephony application. Instead, each application is expected to be well behaved, and not transmit audio when it is inactive. In any case, the client will read from (and play) the audio only from the appropriate socket, but if the deactivated application continued to send audio it would consume network and client resources.

5. USER INTERFACE ISSUES

The previous section described an architecture with some similarity to a window system to manage multiple applications. This includes activating applications, notification and registration when new applications come online, redirecting user input to the active application, managing audio output from the applications, and a mechanism for an inactive application to alert the user via audio. In this section we describe the Impromptu user interface, which was designed to allow the user to keep track of all this activity without a display. Note that we are concerned here with the Impromptu system more than the user interface of any particular application, much as one might speak of the “MacIntosh user interface.” The application manager really acts like a window manager, with separation of content (audio) and control (button presses and recognized speech).

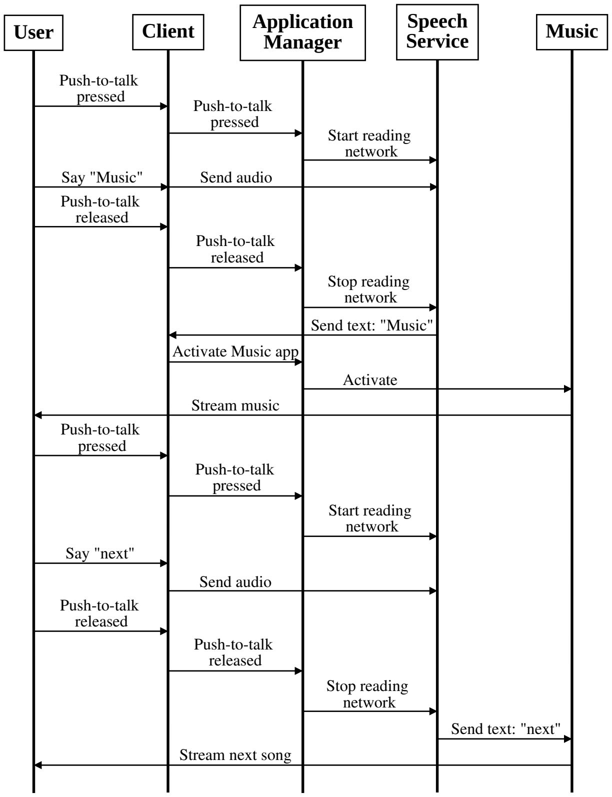

Figure 8: Using push-to-talk.

The first point of note is that we have chosen to have only a single active application, so in general only a single source of audio plays at a time (with the exception of the chat application, which tries to maintain a presence while inactive through creative alerting). Multiple applications could be active if we were to mix their output, which is technically straightforward but not very intelligible or useful. Other techniques could be used. One is to present multiple channels in different spatial locations and allow the user to selectively attend to any one of them, as was done in AudioStreamer [9]. Another is to use various audio processing techniques, including reverb and equalization, to make one audio source “stand out” over the others, as was proposed by Ludwig [4]. Neither of these techniques will work well with the built in low-quality iPaq speaker, however. Although some users tolerate headphones (especially those who prefer to listen to music) each of these techniques places added cognitive load on the user, who must manage to differentiate the sound sources while continuing to function in the real world. Therefore our current design has limited output to a single active application, except for short periods.

Because of this, there must be some alerting mechanism, which in Impromptu must be audio-oriented. We have already discussed why only one Impromptu application can be updating, or playing an audio stream, at a time. However, much as a window system sometimes decorates, or indicates graphically, which application is active, Impromptu does this by sound, or audio icon. The purpose of the audio icon is to allow the user to track which application is active or seeking attention in lieu of any visual cues. This sound plays when a user selects the application, when an application becomes active, when an application terminates (in this case, accompanied by the sound of a slamming door) or when an inactive application alerts, or requests user attention. Alerting is similar to the window system function in which an application pops up a dialog box on top of the rest of the windows to gain user attention.[2]

User input (key and mouse presses, for a conventional window manager) needs to be dispatched to the active application. Impromptu's application manager does a service similar to a window manager, using a combination of speech and button input, with almost all commands enabled by either modality. Some functions, such as selecting an application, are naturally easier with one modality. By voice, one simply speaks its name. But by button, one must use the "left" and "right" buttons to scroll through the applications until the desired one is found, much like a telephone user interface which says *"Press any key when you hear the choice you wish."* and then starts reciting a list of options. But a function such as skipping music tracks is performed rapidly and reliably by pressing "next" and "previous" buttons than by saying the same words [12].

Buttons are likely to be preferred to speech input in a noisy environment, despite their sometimes inconvenient locations, because speech recognition accuracy is likely to be the leading factor in user satisfaction [14]. The user may be more likely to remember a command word than which of four or five poorly labeled, or reused, buttons to select a function. But most important is that multimodal input is particularly suited for error management; the most successful error strategy, in general, is when an error occurs on one modality, to switch to the alternate [13]. In other words, when a speech recognizer has difficulty understanding a word the first time, it is quite likely to have the same difficulty the second time, if it is repeated.

This raises the issue of user interface feedback. When the user speaks a word, the recognizer may detect the correct word, a different word, or nothing. Our initial user interface did not respond when a word was not recognized; if an application name was recognized, a "correct recognition" sound was played, followed by the audio icon for the particular application (see below), followed by sound from the application itself. This was incorrect. The user needs feedback when the recognizer detects no word in response to input, in order to know to try something else. And it is not necessary to play a "correct recognition" sound, because the correct audio in response to the speech command will be played immediately if the command was recognized. If the speech command was to activate an application, playing the audio icon of the invoked application is unnecessary as well, because it will quickly become apparent whether the ensuing audio being played is the desired sound or not. It may, however, be useful to play the icon repeatedly to help new users learn to map the icon to the application.[3] It *is* necessary to play that sound when scrolling by button press, however, as multiple presses in quick succession are usually necessary, and it is important to hear distinctive sounds quickly to know if the desired application has been reached.

In order to provide consistent feedback behavior, some user interaction components are provided in the application manager and speech service. The analogy from window systems is that the applications are written using a consistent widget set, or user interface interactors. Although there is no mechanism provided for an application to override these behaviors, it can modify them. TattleTrail provides an innovative alert, by mixing its icon sound with the actual audio from the alerting piece of conversation, and then substitutes this for its normal alert. When it sends an updated alert message to the application manger, the appropriate sound is played.

These user interface capabilities are required for a multi-application audio platform. The next section discusses the advantages and liabilities involved in streaming audio over IP.

6. WHY IP?

Mobile telephones are ubiquitous in many parts of the world; why not just use this network for an Impromptu-style collection of audio applications? There are many reasons. The telephone network offers few options in terms of call control or setup; a call is either answered, left ringing, or sent to voice mail. There are no options for "partial" connections, such as Garblephone's. Because it is a circuit switched network, usage charges are by the minute, making applications such as radio, music, or books-on-tape expensive. When playing recorded speech, many minutes of audio data are ready to be sent in any sudden period of network availability, making a packet-based network more efficient under some circumstances. Additionally it takes some seconds to set up a call; when calls are minutes long this may not matter but for the rapid exchange of messages in an audio chat session this overhead affects performance.[4]

Most importantly, IP is a widely accepted transport protocol upon which it is easy to build application-specific control messages and multiple higher layer protocols that can run simultaneously. Similarly, its addressing modes allow multiple simultaneous data channels to be open without setting up a connection for each one each time it is needed. Unlike telephones, with IP it is easy to integrate with network-based services such as storage, audio processing, speech recognition, and text-to-speech synthesis. Additionally, except for firewall issues, IP is a global networking scheme; a user could equally well listen to the baby monitor or his or her personal music collection at home or in the office. Similarly we have demonstrated Impromptu at other sites, using the local wireless LAN to connect to servers and resources back at the MIT campus.

IP is not, however, ideally suited to synchronous audio, i.e., telephony. Telephones use circuit switched networks with guaranteed quality of service; in other words, they ensure some bandwidth for audio with defined limitations on

[2] These dialog boxes or pop-up windows may be modal, and grab user input so no other window is active, or they may allow the user to ignore them. Impromptu supports only the latter, which is appropriate for audio-only interfaces

[3] Any such sound mapping should be configurable by users, who may be sensitive about which sounds are used. Users who configure are likely to recall the mapping more readily than with the default sounds.

[4] The usual Voice over IP telephony protocol, H.323, also has non-negotiable call setup, which is why we chose not to use it.

latency. When the audio is broken into packets and sent on a network along with packets from many other addresses, network usage may be more efficient or flexible, but problems may result for an inability to guarantee reliability. Streaming media on IP networks face limitations in the form of latency, or delay, and jitter, variations in the delay time, and also lost packets, which never arrive at their destination (infinite latency).

In a full duplex audio connection, increased latency means that when the other party talks, the receiver does not hear that until some time after. This quickly affects one's ability to interrupt, and seriously degrades conversational quality once latency increases beyond about 250 milliseconds (as was the case when trans-oceanic telephone calls were carried by high orbit geostationary satellites) [2, 7]. Jitter affects how much buffering is required at the receiver, unless we are willing to tolerate breaks in the continuity of the audio stream (i.e., it is time to play a packet, but it has not yet arrived). More buffering effectively increases the delay before one party hears another.

Lost packets may occur because of collisions between packets on the network or other causes. A protocol such as TCP, a layer on top of IP, assigns sequence numbers to packets, assembles them in sequence at the receiver, and can request retransmission when gaps are detected; this makes TCP suitable for moving files, for example. If TCP is applied to a streaming audio connection, however, retransmission will result in momentary silences, and it is generally more acceptable to just play the next packet and keep going. For streaming media, a protocol such as UDP, in which packets are just sent once and presented in the order received, or RTP, which is UDP plus some control information [10], are usually used. If a packet is lost, it is just skipped and the next packet is played, with an audible glitch at the boundary. Some error correction protocols for speech allow for partial error recovery at gaps; the Robust Audio Tool (RAT) is an example [1]. Packet loss error can be particularly problematic for speech recognition.

This argues that applications built for Impromptu should be suitable for IP, as they tolerate some buffering. The baby, for example, is hardly bothered if a parent is listening to her cries with a 500 millisecond delay, since each cry lasts longer than a second. Listening to radio with a similar delay is an issue only if one encounters someone else listening to the same program with a different delay. Push-to-talk with floor control avoids speech interruption difficulties encountered in telephony; even successful commercial implementations of push-to-talk, such as Nextel Direct (a walkie-talkie service overlaid on a digital telephone network) exhibit delays which would interfere with telephone conversation.

It is also possible to use multiple protocols, as well as to cover delays by buffering, because of the temporal nature of an audio user interface. For example, when the user presses the push-to-talk button for speech recognition, Impromptu sends audio to the recognizer using TCP. This means recognition cannot begin until the user releases the button, but there will be no glitches when the audio is processed at the speech server, and hence better speech recognition. And whenever the user switches context and hears an audio cue, the few seconds which elapse while the cue plays is time in which application audio may be streamed over the network and buffered locally.

Impromptu runs reliably in our laboratory, including multiple clients and iPaqs running monitor applications sharing the same 802.11b cell. We have also run mobile clients at remote sites, with applications and services running in the laboratory; except for the highest quality audio applications, performance has been adequate. It remains to be seen how wireless IP will be deployed by commercial carriers. 802.11b, the de facto standard for campus or home wireless networking, is increasingly found in public places like coffee shops or hotel lobbies, and we hear anecdotes of surreptitious access to unprotected corporate networks in many urban business districts. For practical deployment, Impromptu may switch between private and public networks depending on the user's location. Because it is IP-based, this could be accomplished with minimal modification to the application manager.

In the short term commercial carriers are deploying GPRS, but other forms of wireless IP are competitors. We are likely to find performance and bandwidth varying between urban core and suburban or rural locations. Full deployment of Impromptu would require speech compression, but note that each application could employ its own codec. We are already used to intelligible but low quality speech over mobile phones. MP3 is itself a compression scheme. Radio and books-on-tape applications probably require high bit rate codecs, but are more amenable to larger amounts of buffering.

7. PERFORMANCE AND SCALABILITY

In this section we address issues of performance, mostly in terms of the user interface, and scalability of the underlying architecture.

7.1 Performance

Our primary concern with Impromptu was to demonstrate the viability of an audio-only user interface to an array of audio applications, and show how features of a visual user interface (such as multiple windows, input focus, window decoration, and non-modal dialog boxes) could be supported. Of course any user interface needs to be responsive enough to not interfere with the user's selection of and interaction with a particular application. Hence we were concerned with performance at several levels.

7.1.1 Voice over WiFi

There are of course questions as to the viability of transporting rich media, such as audio or video, over wireless IP networks. While we built and tested Impromptu, we were able to stream audio over WiFi using IP networking quite well. Synchronous phone conversations occurred with latency on the order of 100 milliseconds (150 is acceptable according to CCITT standards), and our applications in general ran with only small amounts of buffering. As network congestion increases, we expect more lost packets, of course, and we have seen this on heavily loaded networks. However part of our initial design imagined Impromptu being used on a home WiFi network to access audio around the house (hence our inclusion of the baby monitor) including music collections, listening to the radio, and switching to the phone to take a call. In such an environment it is highly likely that adequate wireless bandwidth exists to support streaming audio.

As network congestion increases, several steps can be taken. One is to introduce more buffering, and hence latency, as this allows more time for stray packets to arrive (or for a retransmission scheme). For most of the Impromptu applications, a bit more latency is quite tolerable. Another scheme is to back off to a lower quality codec. Depending on the audio material, this may or may not be acceptable; it is not clear that one would wish to listen to the radio with the quality of a mobile phone. But there certainly is a range of audio quality that can be moved up and down.

In a seriously overloaded network, multiple media streams may not be possible without causing so much network contention that no traffic gets through satisfactorily. In this case it would be up to the application manager, in concert with another network management service, to generate busy signals and prevent some audio applications from running at all. Another possible approach is to use audio time scaling to slow down the speech output a bit; this can be done so that it is barely noticeable and allows considerable buffering to be invisibly built up, as long as the application is not full duplex and interactive.

7.1.2 User Interaction

Our concerns for performance were much more centered around the ability for such a distributed system to maintain an adequately responsive user interface. Here the primary concern was the separation of the application manager from the client hardware itself. Although initially done for the sake of rapid software development, this scheme also allowed services such as speech recognition to be performed much more effectively on faster processors found in desktop computers. But the separation also raises performance issues in that round trips are required to perform user actions. Specifically, when the user does push-to-talk, the audio for recognition is recorded, then sent using TCP, as losing blocks of audio data seriously degrades speech recognition. Recognition cannot begin until the whole utterance has been spoken and transmitted from the handheld device.

In fact, given how relatively slowly people speak (180 words per minute is typical) the lag for speech recognition was generally not a factor. When it increased, however, the user interface became difficult, as it was not clear whether a word had not been recognized, or was still in transit. The crispness of the user interface was increased noticeably when we instituted a simple protocol of stopping any audio output the moment the user presses the “talk” button; this can be done entirely locally and gives the user confidence that s/he is being heard. The second step was to locally generate a “not recognized” sound after a timeout, so the user would try again. If a recognition result is in transit, it will arrive before the user finishes speaking for the second time, and interaction can proceed normally; of course this breaks down if the delayed first recognition arrives while the user is speaking a different command.

So what happens in actual use is, the user presses the talk button, and audio output stops. If the spoken command is internal to the active application, nothing happens until that application receives the recognition result (which does not require a round trip back to the client; it is sent directly to the application by the speech service); since users are used to some delay when using applications, this is usually tolerable.

If the user selects a different application, then the audio cue (audio icon) for the new application plays before the application starts. Currently this is accomplished by sending audio for this cue from the application manager, and this usually is adequate. However, interaction could be improved slightly by caching these audio cues on the handheld (they are short, and played repeatedly).

7.2 Audio Mixing

Another issue with the thin client is audio mixing. Although our initial design rationale was that only a single application at a time would play, we changed this late in the project. Applications such as audio chat (voice IM) need to run in the background, as there is usually little activity on these channels. But if the user does not hear fresh activity, s/he does not know to attend to the channel. So with the chat running in the background, a new chat sound would be preceded by the chat audio icon, followed by the recorded audio from the remote chat user, mixed in at about 40% of full volume. This provides “background listening”.

This mixing was done on the client. The price for mixing on the client is that double the WiFi bandwidth is being used during the duration of the mix. However, if the mixing were done on the server-side, since we make no assumption that applications run on the same processor, an additional end-to-end network delay would be incurred, and we chose to avoid this.

7.3 Scalability

Scalability becomes an issue in situations in which there are a number of clients and/or services running on the same network simultaneously. Of course in the domestic use scenario much of the content would be local to the home PC, and remote content could be streamed into the home via broadband connections with relative ease (at least for audio, given the considerations above about buffering). However, even in the home there may be multiple users and multiple handheld devices in use.

7.3.1 Speech Recognition

We chose to put the recognition service with the application manager and use the client as a thin user interface to these. Centralized speech recognition allows the most effective use of processor cycles. Since recognition only occurs when a user pushes the talk button, and users listen a much greater portion of the time than they issue commands, it would in fact be very easy to deploy multiple clients around a single recognition server; this would have required minor changes to our software. Such techniques are already used in interactive telephone-based voice response systems in call centers, where DSP channels for recognition are routinely switched between multiple calls, as they are the most expensive resource.

However PDAs are now emerging which have enough processing to perform small vocabulary recognition locally, and this would be the first feature to push back into the client. It allows recognition to be performed at constant speed even in high collision networks, and with the caching of the audio icon cues, a more robust and reliable user interface could be provided.

7.3.2 Network Congestion and Disconnected Operation

As the network becomes increasingly congested, rather than offer poor service to all, it is better to block some channels; the user interface might provide an auditory “busy signal” sound. Alternatively, buffering can be increased and/or different codecs may be employed. Measurement of network congestion, and collaborative distribution of bandwidth among applications and individuals, requires coordination between a number of processes. In the architecture in which the application manager is not on the client, it is probably on the wired portion of the network, which typically has significantly higher bandwidth than the wireless network. Thus this split architecture would allow for better negotiation between entities and decisions to be made by and communicated from a network management service without added traffic on the wireless network, which could already be taxed.

Separating clients from the applications allows applications to reside on different computers, and hence could ultimately lead to efficiency by organizing content appropriately on these servers, such as multiple layers of caching using local memory and local disk, all of which is likely to be greater than on a handheld device. Some services, such as the radio service, would also benefit from the use of multicast; only the chat application was multicast in Impromptu.

Finally, placing the application manager within the wired network allows for mobility of the client across multiple wireless networks. For example, it may go WiFi at home, GPRS while driving in to work, and back to WiFi (maybe at a higher bandwidth) in the office. Rather than set up and tear down the many connections and associated state of each application to which the client is connected, all that is necessary is for the client to re-authenticate itself to the application manager, and open a new set of audio sockets to each application to carry traffic on the new network, and the user can simply pick up where s/he left off.

Impromptu does not, however, support any disconnected operation. Clearly some caching is possible, especially since audio can be transmitted much more quickly than it is consumed at ordinary listening speeds. So a book on tape, for example, could be largely downloaded in a relatively short period of time, and listened to while out of network range. For another example, storing music as large numbers of MP3 files on a portable device has become commonplace since we started working on Impromptu.

8. CONCLUSIONS

This paper has discussed the design issues of software architecture and user interface services for a mobile audio-only platform, Impromptu. Impromptu attempts to move the networked PC into the world of mobile audio applications, which have up till now been the domain of telephony or dedicated devices. On PCs we are used to running multiple applications simultaneously, and easily switching our typing and viewing between them. IP-based networks have provided low level flexible routing while allowing multiple applications to support higher level protocols.

In the open architecture world of the Internet, a wide variety of applications and communication protocols have evolved, such as email, web based discussion boards, text chats, and instant messaging, and users are free to select among them. But text based communication and visual user interfaces detract from mobility.

We seek to encompass the popularity of mobile telephony, and augment it with a variety of related applications which have to date required separate devices and RF spectrum. At the same time, we seek to address the limitations of mobile phones through emphasis on audio-only applications and a range of computer-mediated voice channels, all enabled by the flexibility of IP.

In order to motivate discussion of the Impromptu architecture and user interface services, we initially discussed Impromptu's application suite implemented to date. The architecture supports the streaming audio, storage, and processing requirements of all these applications, in the context of an IP network. It must provide a set of user interface services to allow speech recognition and synthesis in a shared manner. Finally, it must enable user management of multiple simultaneous applications through a consistent and effective user interface.

If Impromptu is compelling, it is less the number than the *variety* of audio applications it supports, and the interactions between them. The characteristics of streaming audio place added demands on a distributed ubiquitous or collaborative computing architecture. The transitory nature of sound in a non-visual user interface requires support from the software environment if the user is to manage multiple applications. Solutions to these problems are much more the domain of computing than telephony, even though the device might feel more like a jazzed up phone than a computer to the user.

9. ACKNOWLEDGMENTS

Elina Kamenetskaya wrote the POTS telephone gateway and parts of the Garblephone application. Mike Jacknis, Stefan Marti, and Nitin Sawhney helped implement and evaluate an early version of the audio garbling algorithm. Years ago, Barry Arons developed some of the browsing techniques reused in reviewing chat history; his work with the first author on audio "window systems" influenced the notion of foreground and background non-graphical applications. Eric Hulteen made some small but powerful suggestions for speech recognition feedback.

10. REFERENCES

[1] V. Hardman, M. Sasse, and I. Kouvelas. Successful multiparty audio communication over the internet. *Communications of the ACM*, 41(5):74–80, May 1998.

[2] C. Jensen, S. Farnham, S. Drucker, and P. Kollock. The effect of communication modality on cooperation in online environments. In *Proceedings of Human Factors in Computing Systems (CHI'00)*, pages 470–477. ACM, 2000.

[3] R. Litiu and A. Parakash. Developing adaptive groupware applications using a mobile component framework. In *Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work*, pages 107–116. ACM, 2000.

[4] L. Ludwig and M. Cohen. Multidimensional audio window management. *International Journal of Man-Machine Studies*, 34(3):319–336, 1991.

[5] 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.

[6] 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.

[7] 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.

[8] C. Schmandt, J. Kim, K. Lee, G. Vallejo, and M. Ackerman. Mediated voice communication via mobile ip. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST '02)*, pages 141–150. ACM, 2002.

[9] C. Schmandt and A. Mullins. Audiostreamer: Exploiting simultaneity for listening. In *Proceedings of Human Factors in Computing Systems (CHI'95)*, pages 218–219. ACM, 1995.

[10] H. Schulzrinne, S.Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. *IETF RFC 1889*, 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, C. Schmandt, and E. Hulteen. Voicenotes: A speech interface for a hand-held voice notetaker. In *Proceedings of Human Factors in Computing Systems (CHI'93)*, pages 179–186. ACM, 1993.

[13] B. Suhm, B. Myers, and A. Waibel. Model-based and empirical evaluation of multi-modal interactive error correction. In *Proceedings of Human Factors in Computing Systems (CHI'99)*, pages 584–591. ACM, 1999.

[14] M. Walker, J. Fromer, G. Di Fabbrizio, C. Mestel, and D. Hindle. What can I say?: Evaluating a spoken language interface to email. In *Proceedings of Human Factors in Computing Systems (CHI'98)*, pages 582–589. ACM, 1998.