IMPROMPTU:

AUDIO APPLICATIONS FOR MOBILE IP

Kwan Hong Lee

B.S. and M.Eng., Computer Science
Cornell University, 1998

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences
at the Massachusetts Institute of Technology

September 2001

Author

Kwan Hong Lee
Program in Media Arts and Sciences
September 7, 2001

Certified by

Christopher Schmandt
Principal Research Scientist
Speech Interface Group, MIT Media Lab
Thesis Supervisor

Accepted by

Dr. Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

IMPROMPTU:

AUDIO APPLICATIONS FOR MOBILE IP

By

Kwan Hong Lee

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on September 7, 2001 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Media Arts and Sciences

ABSTRACT

IMPROMPTU is an Internet Protocol (IP) based audio platform for mobile communications and networked audio applications. IMPROMPTU supports multiple audio applications on a mobile device communicating over the IP network. The audio user interface on the IMPROMPTU client provides speech interfaces to switch applications and control each application with unique vocabulary and grammar along with providing auditory feedback during the usage of the applications. The alerting model is based on the applications competing to attract user's attention. Various audio applications with different characteristics have been implemented, including radio, MP3 player, and baby monitor. Although IMPROMPTU supports conventional voice telephony services, it does so with increased negotiation between calling parties through more flexible call control. The system architecture is composed of distributed services and applications managed by the Application Manager.

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

THESIS COMMITTEE

Thesis Supervisor:

Christopher Schmandt
Principal Research Scientist
MIT Media Laboratory

Thesis Reader:

Mark Ackerman
Associate Professor
School of Information and Department of Electrical Engineering and
Computer Science, College of Engineering
University of Michigan

Thesis Reader:

Brian Smith
Assistant Professor
MIT Media Laboratory

ACKNOWLEDGMENTS

I have been truly humbled by the experience at the MIT Media Lab. I have learned the importance of discipline in life and what it takes to pursue the truth in this world. I cannot help but thank:

God for providing me with the power and perseverance as I struggled through the last two years to balance out my life. He has constantly kept me in watch and nurtured me spiritually throughout my endeavor at MIT to be focused on what is really important in life.

My parents, brother and sister and my Cornell buddies for their love, support and encouragement.

Chris Schmandt, my advisor, for introducing me to audio interfaces, actively supporting me in this research endeavor, bearing with my slow performance and advising me to go further with my attempts to prove that something really works.

Mark Ackerman for giving me advice and encouragement in times of uncertainty and for reading my thesis.

Brian Smith for his sense of humor and his help in writing a better thesis and providing support during the last mile of the thesis writing.

Speech Grads, Natalia Marmasse, Stefan Marti (for Perl Script), Sean Wheeler (audio for Audio Book), and Nitin Sawhney (ex Speech grad) for constantly encouraging me and being there whenever I needed help and advice. Thank you also for proof reading my thesis.

Rm 350 buddies, Thomaz Edison, Shyam Krishnamoorthy, Floyd Muller, and Jim McBride, who have been great friends to share office with. They kept me sane as there was always someone to turn to when I was in that room and made the Media Lab experience really enjoyable.

UROPs Jang Kim, Elina Kamenetskaya, David Kim and Joshua Baratz for giving me the opportunity to work in a team and also attack hard problems together. Especially Jang Kim for his diligence.

Tom Gardos for supporting me with computational resources for my research.

Brothers and sisters of First Korean Church of Cambridge who prayed for me as I was struggling to finish my thesis and gave me continual spiritual support.

Roz Picard who challenged me how excellence was not to be traded off with faith, but how they can coexist.

Linda Peterson for her patience and her help during my stay at the Media Lab.

Pat Turner for taking care of ordering different resources I needed.

Aisling Kelliher, John Kim and Wona Park for proof reading my thesis.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1. INTRODUCTION

"We're moving to a world where you're constantly connected. That's a good thing. If people need to reach you, they can. If you need to reach somebody else, you can. Applications will presume that you're already connected to the Internet. So they can always grab or send the data they need." [1]

With increasing mobility, mobile telephones and handheld computers are becoming commodities in our lives. Recent growth in mobile phone usage has shown the value of voice communications for people on the move[2]. Although mobile messaging and Internet access through mobile phones are already popular in Europe and Japan, they are just beginning to gain acceptance in the United States. With increasing number of data oriented applications, more and more people are using the wireless bandwidth to access different services beyond telephony. The development of Wireless Application Protocol (WAP)[3] raised hopes of revolutionizing wireless data access from the mobile phones, and VoiceXML[4] has been standardized by the World Wide Web Consortium to provide commercial voice based interactive services to the phone users. Emergence of these technologies and their acceptance indicate that people want the phones to have multiple functions and to be connected to different services providing various applications.

Nonetheless, there are fundamental problems with the current mobile telephone systems that limit their functionality. The major downfall of a mobile device is that traditional graphical user interfaces do not scale down to the mobile device's small screens. The display area is small and full sized keyboard and mouse are not available for data input. As a result, the main input method is through button and pen-based inputs. With such a wide spread acceptance of mobile phones, the importance of audio based interfaces increases as one's eyes and hands are needed for other tasks while moving around.

In addition, the wireless network connections provided to these devices are based on traditional circuit-switched phone networks, which require a caller to call a number and wait for the recipient to respond to an alert before a connection is setup. The circuit switched network is not designed to handle multiple applications or multiple, connected communication channels. For example, when you are browsing using WAP and you get an incoming voice call, you will need to hang up on the WAP connection, attend to the voice call and then redial and reconnect to the WAP service to continue browsing. In contrast, the success of i-mode service from NTT DoCoMo indicates the importance of "always on" nature, which removes significant barriers to accessing applications and services using the mobile phone[5]. i-mode uses a packet switched network for its services in contrast to WAP, which uses the circuit switched connection on mobile phones.

Future generations of wireless networks will provide greater bandwidth to mobile devices, and these devices will be equipped with enough computing power to provide nomadic users with a diverse set of applications. Although a general purpose appliance such as the personal desktop computer is considered undesirable for some situations and computing in the future[6], the popularity of Palm devices and handheld computers shows how multiple applications can be useful for mobile users. Currently, people have to carry many different devices such as Palms, pagers, phones, MP3 players and walkie-talkies for different types of communication. However, I believe mobile computing is experiencing the same paradigm shift that personal computers have experienced -- from supporting single applications at a time to supporting multiple tasks and broader communication capabilities (with increasing computational power and greater availability of network bandwidth). This transition will require a new type of user interface that is more suitable for mobility, just like the multi-tasking environment on the desktop has led to novel graphical interfaces using window managers.

1.1. The Challenges

Existing mobile devices that provide audio applications for mobile users are mainly built to support a single application on a single device. MP3 players, CD players, Mini Disc players, and walkmans only allow playback of music; voice recorders allow recording and playback; radios only support broadcast radio; walkie-talkies support half duplex wireless voice communication; mobile phones support full duplex communication, but are bound to a built-in alerting protocol before a communication channel can be setup. On the other hand, personal computers are a multipurpose platform for a wide range of different applications. Combined with the Internet, the PC has been a general-purpose platform for different types of communication applications ranging from instant messaging, e-mail, news groups, IRC chats and even Internet telephony. However, these applications have been designed with the assumption that users are sitting, stationary, and focused on the application. The advent of mobile IP suggests that there may be a role for a single, audio based appliance that is mobile and multi-functional that replaces all personal mobile devices currently available while supporting multiple applications for nomadic users.

In order to make such an appliance work well, this thesis presents:

1. An appropriate architecture with necessary services on the network to manage and support multiple audio applications with various characteristics.

2. A powerful but simple communication protocol to communicate with the applications and services.

3. An auditory user interface to manage and interact with these applications with minimal visual attention.

As computational power converges into smaller devices and as higher wireless bandwidth connectivity becomes more commonplace, the mobile devices will be able to support a diverse set of networked audio applications. However, these applications will be competing for different resources including the microphone and the speaker on the client, the user's attention, and the speech resources (speech recognition and text to speech). This will require appropriate management of resources and applications with adequate load sharing between the client and the network services that support the client.

In addition to the architecture, a standard communication protocol is needed for the client to communicate with the services on the network and the applications. The client needs to switch between the applications and request necessary resources when certain applications are activated. The applications also need to know when to send and receive data and when to use resources such as the speech recognizer.

Finally, a novel design of the audio interface is needed in order to interact with various applications from the mobile device. A speech interface would be a requirement for such a device since mobile users will often have their eyes and hands focused on other tasks. Users will need the capability to access and control the applications using voice.

1.2. Contributions

IMPROMPTU is an IP based mobile platform, which consists of a wireless client device and distributed services that support multiple audio applications. IMPROMPTU addresses the above principles and provides an implementation of a multifunction mobile audio appliance.

A distributed architecture was implemented with various distributed components and applications. Distributed services and applications make it suitable for efficiently load balancing and optimizing the usage of computational resources on the network. In the current architecture, a thin client communicates with the distributed services over the network in order to interact with distributed audio applications. An Application Manager was developed to manage these applications and services for the client. In order to support a personalized management of the applications, the Application Manager communicates with the user's personal services, which include the Presence, Profiler and Speech Services.

The communication protocol is based on the standard Internet Protocols. We use the TCP/IP protocol to transmit control messages among different components of IMPROMPTU. However, many components of IMPROMPTU are implemented in Java, allowing Remote Method Invocation (RMI) to be utilized to communicate among the networked components. A text messaging protocol has been implemented in order to communicate with non-Java entities such as the client and some applications.

Different audio applications that would be useful for mobile users have been implemented to demonstrate the capabilities of IMPROMPTU architecture and investigate the user interface for audio applications. Baby Monitor, Radio, News Headlines, Audio Book, Recorder, Chat, Super Phone, and MP3 Player applications have been implemented. Development, testing and coordination of distributed applications were very challenging during implementation. Speech interface was a necessity for mobile applications and most applications support speech recognition by sharing the Speech Service and dynamically loading their vocabularies when they require customized speech interactions.

A push-to-talk button interface was implemented to provide on demand speech interaction. Initially, the push to talk button was a toggle button with visual feedback of the recognized text. Such an interface confused users about the state of the speech recognition. When it was redesigned as a push and hold button, the users had a more positive response since it gave tactile feedback on the state of the speech recognizer and an auditory feedback when there was a successful recognition. Audio cues were a very important element in informing the users about the state of the applications since there was minimal visual feedback. For example, when there were no audio cues in the Recorder application to indicate the recording state, the record button would be pressed by mistake and either it failed to record anything or record without the user knowing about it. When audio cues were used to notify the start of the recording and the end of the recording, there were less false recordings.

Applications also have unique alerts that are used to signal users for attention. These alerts were initially very random and of varying lengths. Iterative testing led to the conclusion that the alerts should be around one second or less and should give an indication about the content of the application. Selecting such appropriate alerts was a challenge, and in order to facilitate this, we provide a web interface for users to set their alerts.

The design and implementation of IMPROMPTU overcomes the limitations of current mobile audio communication systems by providing an extensible architecture based on the IP network and providing a new design of audio interfaces to interact with multiple audio applications from a single device. It takes advantage of the computational power and wireless bandwidth that is becoming more available on mobile devices in order to provide the user with a combination of auditory and tactile interfaces for accessing digital audio applications. Audio interfaces have been explored for several years as a means to provide easy access to information and communication on mobile devices. However, IMPROMPTU provides an architecture that can support several of these audio applications on a single device by utilizing the availability of the wireless IP network through 802.11b wireless Ethernet (which transmits data in the unlicensed spectrum at 2.5GHz supporting data rates up to 11 Mbps) and the programming and computing capabilities provided by a Compaq iPaq-like hand held device. This will help us manage busy social lives in a networked mobile environment by providing an elegant user interface to access many mobile audio applications on a single device and improve how we communicate with people and access digital audio entertainment and information on the move.

CHAPTER 2. EXTENDED EXAMPLE

IMPROMPTU is an architecture that enables access to multiple audio applications with different characteristics from a single mobile device. As part of the thesis, I have implemented eight applications for IMPROMPTU that people would want to use in real life. In this chapter, I present several usage scenarios to illustrate how these applications would be used by people in their daily lives in different situations and how the interface features of IMPROMPTU facilitate interactions that look very different than those associated with the desktop graphical interfaces or the current mobile phones. IMPROMPTU assumes that high bandwidth wireless connectivity will be available everywhere in the future. In our implementation we used the 802.11b wireless LAN to simulate such a connected environment and as a result, the applications are usable only inside the Media Lab. In order to illustrate the full potential of IMPROMPTU, three different real life scenarios are used to introduce and examine the functionality of the applications. The first one illustrates how IMPROMPTU can be used by parents with young children. The second scenario presents a more public environment involving communication between co-workers. A final example presents how IMPROMPTU can facilitate the communication between geographically distributed family members.

2.1. A Normal Parent's Daily Scenario

2.1.1. Baby Monitor

Joseph and Mary are a married couple in their thirties who have two children, Jesus and James. Joseph and Mary are both working professionals who hire a baby sitter to take care of the babies when they are away from home. Their babies are monitored with baby monitors that are attached to the children and connected to the wireless LAN connection at home. The parents can monitor their babies through these monitors regardless of where they are through their IMPROMPTU client. The baby monitor is an application of IMPROMPTU that is activated when any of the babies start crying, or some loud noise occurs in the environment. When triggered, the baby monitors signal to alert Joseph and Mary. Each monitor can even have different alerts to indicate whether it is Jesus or James that is crying. The mother is alerted primarily and the father is alerted when the mother does not respond within ten seconds after being alerted. This feature can be customized, if desired, so that both of the parents would be alerted simultaneously, or the father could be alerted before the mother.

2.1.2. Multiple Applications

While commuting, Mary likes to listen to audio books related to children's stories. When Mary holds down the push to talk button and says "Audio Book" she hears a chime indicating that her voice command was recognized and hears a book flipping sound that notifies that the Audio Book application has come to the foreground. She has been listening to the stories of Winnie the Pooh and the application starts streaming that audio book when it comes to the foreground. The application always bookmarks where Mary leaves off during her previous interaction with the book and it starts streaming from twenty seconds of audio that precedes from where she left off last time so that she may recall which part of the story she left off.

Joseph likes to listen to his music collection while he works and he has also setup a profile for news headlines so that he can receive important news updates while he is listening to his music. He has his own MP3 music collection that he hosts at home and is able to access through his IMPROMPTU client. While he is listening to Beethoven's Symphony number 3, he receives an audio alert from Yahoo! News. He "jumps" to the application by pressing the "activate" button. The client plays an alert that the News application has come to the foreground and the News application is immediately activated. Joseph hears the headline regarding the current updates on the conflict between Israel and Palestine. He says "more info" in order to get more detailed information about the news headline. While he has the news application active, he also browses through other news by using up and down buttons.

While browsing through the news, he learns of a film festival taking place in Boston and decides to reach Mary to discuss on a date for that night. So, he browses through his applications until he reaches the phone application. He presses the push to talk button and says "Phone". He hears a short chime indicating that his commands were recognized and he hears a ringing sound that indicates the phone application is activated. He browses through the list of users using the "down" button, but Mary is the first one that comes up so he says "Call" which establishes a connection with Mary.

2.1.3. Enhanced Telephony

Once Joseph activates the connection to Mary, he is authorized to listen to her garbled audio. Garbled audio allows Joseph to listen in to the audio environment of Mary through a garbled channel, allowing him to infer her availability with minimal intrusion. This is equivalent to him approaching her in a physical encounter and deciding whether he should interrupt or not. Mary hears a soft alert from her IMPROMPTU client that Joseph is trying to reach her. Mary is a little busy at that moment so she waits to see what Joseph is calling about. Joseph realizes that she is not accepting the call so he waits for a few minutes with a garbled connection to see if she becomes more available. Mary does not respond, so he takes the next step by saying "approach" and decides to leave her a voicemail message. At that point, Mary is notified that Joseph is trying to leave a message and decides to screen the call and listen to the message. Joseph leaves a message about the film festival and whether she is willing to go out that night. Mary has either the option of accepting the call and talking to him, or simply ignoring it. In this case, she decides that a quick answer will suffice, so she decides to connect to him by pressing the "up" button. Real time connection is established and Mary tells Joseph that the film festival would be great and that she would pick him up at the T stop.

However, as Mary is driving to pick up Joseph, the baby monitor suddenly alerts her. She immediately jumps to the application by pressing the "activate" button and starts listening. James is crying for some reason and she tries to reach the baby sitter, but she does not get any response. She switches to the phone application and calls Joseph and tells him that she is heading home because of the children. Joseph instantly activates his baby monitor by saying "Baby Monitor". The baby monitor plays a unique alert indicating that it has come to the foreground. Immediately he hears James crying and Jesus saying "Do not be afraid," to calm him down. Joseph is also able to speak into the monitor to calm him down since the monitor supports full duplex audio.

2.1.4. Summary

This scenario presented several applications of IMPROMPTU and how they interact among themselves and with the users. Baby Monitor, Audio Book, News, Music and Phone applications were presented. Users have the option to interact with these applications using speech or using the buttons on the IMPROMPTU client. Only one application can be in the foreground although all applications are practically active. When the applications need focus, it signals with an alert. There are speech commands to activate the applications and speech commands used to browse the content inside the applications as in the News application. The speech recognition is done over the network by streaming the audio captured from the client to the speech recognizer and getting back the recognized text. The Application Manager keeps track of which applications are online, which applications are active and which applications may interrupt the user or not. It relays all client inputs and requests to the appropriate applications.

2.2. A Lab Environment Scenario

David works at the Media Lab and Esther and Peter are good friends and coworkers at the lab. However, they reside physically in different locations in the lab, and they decide to use IMPROMPTU in order to stay connected with each other. They like to go out together once in a while and sometimes they also have to collaborate on projects and IMPROMPTU provides them with a convenient awareness and communications channel.

2.2.1. Connectedness

Thunderwire[7] was a shared media system that allowed several members in a work group to be always connected and share an audio space. The study showed how telepresence could be used to collaborate while forming a social space among coworkers. Ambient workspace sounds were conveyed through the Thunderwire audio channels to allow different members using the system to remain aware about each other's presence, when they were not using it to converse.

IMPROMPTU can serve a similar purpose to David, Esther and Peter as they work on a project trying to meet a deadline. The audio channel is not always open, but any of them can reach any other person through IMPROMPTU because they have given each other permission to interrupt during their work hours. In the morning when Esther comes in, she approaches David through the IMPROMPTU client to check if he is in the office. David hears an alert indicating that Esther wants to reach him. He is on his way to work and he is listening to music and ignores the alert not wanting to be disturbed. Esther just hears some garbled audio and it sounds like he is near some busy street and assumes that he is on the way to work. This "garbled" connection enables users to let others know about one's availability by publishing audio or other data to one's peers as in the AROMA[8] mutual awareness system. AROMA system used abstract representations, which required a high learning curve. However we use garbled audio to keep things simple, while preserving some privacy. The audio is garbled on the client so that it cannot be spoofed from the network.

Since David seemed like he did not want to be disturbed, Esther decides to leaves a voice mail by approaching David further (by pressing the "up" button or saying "approach") and tells him to let her know when he comes in, in order to discuss the status of a bug and also asks if he can do lunch today. David hears an audio cue saying that the caller is leaving a voice mail. This is to give the receiver an option to decide to connect the call while the caller is leaving a voice message if the message seems to be important. When the caller is also done with their voice message the receiver hears an audio cue (currently "beep beep") that indicate the end of the voice message. The voice mail is recorded by the Recorder application in IMPROMPTU. The Recorder application is an example of an application that becomes active in the background when it needs to record audio from another application. The Recorder application can be explicitly activated by the user to record user's personal voice memos or it can be activated when another application is already active, in order to record an audio content of another application.

2.2.2. Alerting and Priority

While Peter is preparing to go to work, he listens to the weather forecast on his IMPROMPTU client to see if he should take his umbrella. While he is checking the weather, he suddenly gets an alert from the Phone application. Since he can come back to the weather, he decides to immediately connect to her by pressing the "activate" button and jumping to the Phone application. He hears an alert that indicates that the call is from Esther so he presses the "approach" (up) button twice to get connected. She asks to have a lunch meeting and he agrees on it. After he is done with the call, he returns to the weather forecast application by saying "Weather" and is happy to know that he does not need to take an umbrella.

As he steps out from his home and gets into his car, he suddenly gets a call from his father in Utah. It is six in the morning there and he is kind of worried because his mother has not been feeling well lately. His intuition is correct and his father tells him that she has been transported to the emergency room. He immediately decides to forego work for the day and heads instead to the airport. He also calls back Esther and from her garbled channel he infers that she is in a meeting, so he just leaves a voice message indicating that he has a family emergency and cannot make it to work today. Esther however, was getting a little bored from the meeting so she screened the voice message and upon hearing Peter's distressed message, she connects to him immediately and sends a word of encouragement.

2.2.3. Chat Application

When Peter arrives in Utah, his mother had gone into the operation room. While he is waiting outside in the hospital he decides to reach Esther and David to see if they can chat to discuss any bugs that are occurring and keep track of each other's status. However, David, Esther and another developer were already in a chat session and discussing several issues. Peter joins the chat session and quickly skims through the chat recordings by pressing the "down" button in order to catch up on the status of the current chat. The others notice that Peter joined the chat as they hear Peter's alert. After several minutes of catching up, he decides it would be a good time to give an update. He first tells them that his mother seems to be doing ok and gives a brief status update on his modules indicating some pending bugs that he has been trying to fix lately.

2.2.4. Summary

In this section, the capabilities of the Phone application have been elaborated, and the Chat application was introduced. The garbled audio and the call screening features in the Phone application enable a more flexible call control and call setup process between the calling parties. The call control protocol alters the call state in a call session. The call state includes the different states in a call and information about callers and receivers. The call setup process establishes the call states in the network and the end points so they may start communicating.[9] Due to its capabilities beyond current telephony, the Phone application will be referred to as the Super Phone application. It supports traditional telephony so that users may dial out to and receive calls from conventional phones. However, by utilizing the IP network, it provides options to monitor one another and screen calls before a two way real time communication channel is setup so that certain trusted group of people such as family members, friends and coworkers can use the enhanced call setup functionality.

This section also illustrated how audio applications may interact with each other as the Recorder application did with the Super Phone application to provide voice mail functionality. The architecture of IMPROMPTU facilitates this kind of interaction between applications. The Chat application presented in this section is a multi-user application of IMPROMPTU that enables a group of people to have a session setup to communicate freely and post voice messages that they would like to share asynchronously. It also keeps a history like the text chat rooms so that users may browse through the contents of the chat to catch up with current conversation.

2.3. Across the Globe

IMPROMPTU will help connect geographically separated family members. Sarah is in US studying in college. Her parents travel between the US and Korea every six months. When her parents are in the US, they stay with her, but when they are in Korea, they are in Seoul, at their relative's place. Her brother John is in the military in Dae-koo, which is about 4 hours away from Seoul by car. Connecting global families in a distributed world where there are different time zones and completely different life styles is something that current telephony does not do well. IMPROMPTU will help such family members stay connected through the enhanced call setup process it provides.

2.3.1. Telephony Reinvented

When Sarah's IMPROMPTU client is idle, her family members can easily check up on her, since she has given them trusted permissions. They would not get clear or real time audio when they check on each other, but they would be able to infer from the garbled audio whether any of their family members are busy or idle. During John's lunch hour, he "approaches" her to see what is up with her. He listens to the garbled audio and there seems to be some activity. Although it is one AM in US, he realizes that she is not asleep. Sarah is alerted that her brother is trying to connect to her. She was writing some e-mails while listening to some music with her IMPROMPTU client, so she decides to accept the call and start a short chat with her brother.

In another instance, Sarah tries to call John or "approach" him. She hears some garbled audio for 30 seconds and infers that he seems to be in a military training. While she is listening to garbled audio, John approaches her and transitions to monitoring state to see what she is calling about. This establishes a one-way clear channel where John can listen to what his sister is interrupting him about. As Sarah asks him whether their parents have arrived safely in Korea, he switches the state by "approaching" her further and setting up a full duplex connection. He tells her that he was directing the training and he has a few moments during the break. He also tells her that their parents have arrived safely in Seoul and he talked to them last night. Sarah asks John to join the chat session later when he is done with the training.

2.3.2. Another Application: Radio

Sarah's parents have been listening to the Korean radio by accessing the Radio application from their IMPROMPTU clients. They were on their way to a sauna since that is the first thing Koreans usually do when they return to Korea. Her father suddenly hears an alert from the Super Phone and he decides to switch the application by pressing the "activate" button while her mother continues to listen to the radio. As he switches to the Super Phone, he asks his wife to save the radio content using her recorder so that he may share it after he is done with the call. When the "record" button is pressed, his wife hears a beep that indicates the start of the recording. Sarah's father cannot really record the radio while he is attending to the call since current IMPROMPTU Recorder application can only record the currently active application.

When he hears Sarah's unique alert he just presses the "up" button twice to skip the monitoring state and get connected directly. Sarah asks how they are doing and he tells her about what is happening in Korea and that they are going to a sauna. Suddenly Sarah's mother wants to chat with her also so they all browse and activate the Chat application. Although currently Chat only supports asynchronous messaging, the Chat application can provide multiple people to also chat together in real time. As they were chatting for about half an hour, they hear John joining the chat. He quickly browses through previous recordings of the chat and greets the other family members. At this time, John's father decides to continue to listen to Radio and let his children chat with their mother. He uses the browse buttons (left/right buttons on the IMPROMPTU client) to switch to the Radio application, since the speech recognizer does not cope too well with his English pronunciation.

2.3.3. Summary

This section presented a scenario where IMPROMPTU can facilitate communication between geographically separated family members by providing them an awareness channel and a private chat room that is accessible from the same client. Also the final application that has been implemented for IMPROMPTU, the Radio application is introduced. This application is actually a pretty simple application that streams audio from a tuner. However, this demonstrates how IMPROMPTU integrates different kinds of audio applications that we currently use and access using different devices. Finally the interface on IMPROMPTU to access applications provide some main functionality such as browsing different applications using buttons to complement the speech interface that might not be always adequate in certain situations.

CHAPTER 3. THEORY/RATIONALE

In this chapter I describe some previous and related work that has motivated the development of IMPROMPTU. The first section describes audio server research that dealt with some similar research issues in architecture and user interfaces for making audio easily accessible from the desktop computers. In the second section, developments in IP telephony is presented along with how IMPROMPTU differs from traditional IP telephony. The third section describes works in mobile phones that have been developed over the last few years and the resulting commercial services that are becoming available. In the fourth section, I present some related infrastructure and architecture work that tries to provide users with access to ubiquitous computing. The fifth section details previous work on audio interfaces that have motivated the development of IMPROMPTU. Finally, in the last section, previous work on awareness channels is described to support IMPROMPTU's new approach to setting up calls.

3.1. Audio Servers

Audio servers have been an active research area in the 80's and early 90's to make digital audio more accessible from the desktop computers. Integration of computer and telephony led to applications that could seamlessly access voice mail recordings from the workstations. The audio servers were used to provide audio services such as digital signal processing, telephony, text to speech and speech recognition, to various applications on the desktop. Desktop audio research presented in [10] investigated "a unified view encompassing the technologies, applications, and user interfaces to support audio processing at the workstation." The research on audio servers and architecture for desktop audio were driven by the requirements of the applications that would be more useful with the use of digital audio and their user interfaces on the desktop workstations. By making audio as a basic data type that is as easily manipulable as text on the desktop computers, the users could more easily interact with audio from their workstations. The architecture for achieving desktop audio resembled the client server architecture of the X windows system by allowing multiple applications to access digital audio resources simultaneously without interfering with each other. The technology provided users with easy access to telephony, to their voice mail, to voice annotations in their calendars and to text to speech and speech recognizers. IMPROMPTU is a natural evolution from the previous research on audio servers to investigate the requirements of the user interface, software architecture and audio applications for users on the move in the mobile arena. Audio applications become more useful to mobile users and speech interfaces becomes more of a necessity due to lack of visual interfaces and sophisticated inputs such as the keyboard and the mouse on mobile devices. The audio server research promoted a server based approach to processing audio and presenting it on the desktop computers for different applications, and IMPROMPTU takes the idea a step further to allow distributed audio applications to be easily accessible from mobile devices through mostly auditory interfaces.

3.2. Internet (IP) Telephony

Internet telephony protocols have been developed in the last few years to send voice over the IP (Internet Protocol) network[11]. The two main protocols that have been standardized are the H.323[12] from the International Telecommunication Union (ITU) and the Session Initiation Protocol (SIP)[13] from the Internet Engineering Task Force (IETF). H.323 is a very complex[14] protocol that encompasses all aspects of multimedia communications over packet-based networks. The protocol even includes standards for audio and video codecs. The call control protocol it uses is H.225, which is a subset of Q.931 protocol that is used in ISDN digital telephony. The call setup protocol which is the negotiation protocol for setting up the connection between two entities that are trying to communicate, follows the simple call setup process as in the traditional telephony involving setup, alerting, connect and release messages. Recent development efforts in SIP also include support for presence related protocols to provide awareness between callers[15]. "Presence is a means for finding, retrieving, and subscribing to changes in the presence information (e.g. online or offline) of other users."[16] These protocols specify how audio and video can be transported over the Internet and what kind of signaling protocol is necessary to communicate between terminals and different entities on the network.

However, the protocols, themselves, do not inherently change how we establish connections and communicate using voice with other people. As described above, the call setup model for these protocols still follow the traditional idle, call in progress, alerting, connected and disconnected model. The negotiation of the call setup only involves either answering or not answering an incoming call with the ring alert being the only indication for the desire to setup the call. Voice over IP mobile phones that are capable of operating on the 802.11 wireless networks are not much different from a normal telephone except that they can access the LAN and transport audio over the IP network. Except for the endpoints having IP addresses and the quality being dependent on the network traffic, the user experience was not much different from current mobile phones. Callers still have to alert, and neither of the parties on each end know what the call would be about until they are actually connected. The alerting is context insensitive. Neither the sender nor the receiver of the call can know the urgency of the call or the status of each other until the call is setup. As a result, what the users experience on the IP phones is not much different from what they experience when they use conventional telephones.

IMPROMPTU leverages the IP connection between IP enabled devices and the "always connected" property of the network to provide users with a different experience when connecting with people using voice communication. It also utilizes the universality of the IP to coordinate between distributed applications and services and to transmit digital audio over the network to the client. By making the call setup protocol more flexible, a user can sense another user's status using garbled audio and subsequently transition to a normal connection if the receiver is available for interruption. It also provides receivers with the option of screening calls in real time before establishing connections to converse. Call screening allows the receiver to listen to the caller to determine the purpose or the urgency of a call, without the caller hearing the receiver. This models real life physical interactions between people when they initiate face-to-face conversations. Before interrupting someone, one approaches close enough to notice the availability of that person. The interrupted person can decide whether to interrupt their current task or ask the person to come back later or send an e-mail. In this interaction, both parties are aware of what is happening to each other during the whole conversation setup process.

3.3. Mobile Telephony

In mobile telephony, the WAP Forum[3], NTT's i-mode[17] and the "Voice Browser" Activity[4] are the main consortiums developing standards for providing visual applications and interactive voice portals for mobile phone users. Their work will provide different services that are useful to current and future mobile phone users. However, the small screens on current mobile phones make visual interfaces very difficult to use for most applications. The speech based interactive applications provide quite useful services, but because the phone operates on the circuit switched network, the users are required to disconnect and dial a different number to access different applications unless all applications are provided by a certain provider. The limited bandwidth on the current phone network also limits the quality of audio that can be transmitted over the network. Upcoming development of third generation wireless networks for mobile phones and mobile devices that support IP network connections will become a potential deployment infrastructure for IMPROMPTU. Many different packet data oriented applications are proposed, and IMPROMPTU will provide guidelines on how applications, particularly audio applications, can be supported on these devices.

Telephones have been made for circuit-switched networks, which require a separate channel reservation for each connection. This makes multiple connections to multiple applications expensive and difficult to manage. Multiple audio applications can be supported by current telephony through service providers such as TellMe[18], which offer multiple audio applications such as stock quotes, weather, news, driving directions and yellow pages through a 1-800 number. TellMe also offers easy ways to publish "phone sites" that you can browse (with speech) using the phone. However, the services are limited to the interactive speech based audio applications that VoiceXML[4] can offer.

Among many different audio applications IMPROMPTU supports, enhanced telephony changes the call control and call setup protocol of how people communicate using mobile devices for voice communication to provide users with richer interaction over the mobile devices. The traditional telephones use circuit switched phone network, which requires the phone to always alert before a call can be setup. The Internet Protocol (IP network) changes this completely because one is virtually always connected with other people over the mobile devices. As a result, different ways of setting up calls can be realized.

Alerting can be more intelligent and provide more information about the status of a call. Before an actual real time two-way communication is setup, an awareness channel can be established in order to provide an "outeraction"[19] channel for call setup. This channel does not involve a direct interaction and communication, but aids in the process of setting up the call by providing both the caller and the receiver with information about each other without interrupting their current tasks. This awareness channel provides a flexible negotiation mechanism to both the caller and the receiver in making the call setup decision for connecting fully or not at all.

3.4. U C

Other projects that also attempt to change the Internet and telecommunications landscape, are the ICEBERG[9, 20] project at UC Berkeley and the Oxygen[21] project at MIT Laboratory of Computer Science, which are larger in scale and scope compared to IMPROMPTU. The ICEBERG project explores the lower level computing infrastructure and wireless networks for providing services to mobile devices. Oxygen project explores the more fundamental levels of computing infrastructure and communication infrastructure, which involves research in hardware chip designs to user interfaces. The Ektara[22] architecture proposed by the Wearables group at the MIT Media Lab also encompasses a larger scope in wearable and ubiquitous computing to provide computing resources and data to wearable clients. The goal of ubiquitous computing is to make computers invisible while making them widely available in the normal physical environment of people's daily lives[23]. Don Norman suggests that current personal computers are too complex and designed to be too general purpose to do all things for all people[6]. However, he also argues for information appliances that are activity specific devices that fit the way we work.

The architecture proposed for IMPROMPTU is more modest compared to the works in ubiquitous computing since its primary goal is to support distributed audio applications and enhance audio based services for users. However, the goal of shifting the focus of attention so that the users may engage in their active tasks is similar. By bringing as many audio applications as possible into an integrated environment, IMPROMPTU models more of a general purpose desktop computing paradigm, but IMPROMPTU can be considered more of an appliance for accessing audio information and audio communication ubiquitously. IMPROMPTU is an architecture that stands in the transition stage (marked by Mark Weiser as Widespread Distributed Computing stage) to ubiquitous computing vision where distributed audio applications are connected and accessible to the users at any time through an IMPROMPTU client. All the complex audio processing and applications run in the background and the distributed architecture of the system becomes transparent. Also, IMPROMPTU is here currently and it works now over the Internet protocol. However, with true ubiquitous computing becoming available, IMPROMPTU's design of user interface and audio applications will become more accessible while being less intrusive. Rather than just one device being the source of audio, the audio source may be the physical environment itself.

3.5. Audio Interfaces for Mobile Users

Nomadic Radio is a wearable platform for nomadic access to unified messaging. Interface techniques were designed for simultaneous and peripheral listening, spoken feedback, and navigation via voice and tactile interaction[24]. This thesis explored wearable audio interfaces for accessing asynchronous information (voice mail, e-mail, and calendar) from a wearable device and contextual alerting for timely notification of incoming messages while minimizing interruption to the users. Nomadic Radio describes the advantages of audio interfaces that allow hands free operation for mobile users. However, Nomadic Radio only supported incoming asynchronous messaging to access voice mail and e-mails converted to speech using text to speech. IMPROMPTU provides an implementation to some of the future works described in Sawhney's thesis with regards to awareness communication channels, synchronous and asynchronous two-way communications and their interaction protocols that allow transition from one communication channel to another. In solving these problems, IMPROMPTU first provides an architecture that supports multiple audio applications with an adequate arbitration of resources through distributed services. An alerting model based on the applications competing for the resources on the client and the user's attention is provided so that the users are notified about the events that occur in the application space. Alerting of events triggered by not only incoming calls from people, but by multiple audio applications are handled by the IMPROMPTU architecture. Also, the user interface is designed to be scalable to support different applications using speech, tactile and auditory interfaces on a handheld voice terminal. To provide an awareness channel, IMPROMPTU supports garbled audio that can be used to negotiate further interruption when a call is being setup and provides capabilities to transition to a full duplex audio channel or transition to Chat application for asynchronous voice messaging.

NewsComm[25] is a handheld playback device for structured audio recordings. It allowed easy browsing of audio books and news content. However, content had to be retrieved from the audio server and needed to be manually downloaded to the handheld device by docking it, due to the lack of wireless network connection. This limited access to real time streaming content from the device. IMPROMPTU, on the other hand, makes a wireless high bandwidth connection a requirement, and it allows on-demand access to audio book contents and news. This allows new content to be available from applications immediately, requiring a more sophisticated alerting paradigm to indicate different events that dynamically happen during the use of the application. There is no intelligent structuring of the content for indexing or annotating the media stream as NewsComm does, but an application developer could potentially implement such functionality in the application.

3.6. Awareness Channels

Interaction and Outeraction: Instant Messaging (IM) in Action[19] describes how instant messaging has enabled people to "negotiate the availability of others to initiate conversation", and how this capability has facilitated the establishment of communication channels. Such a feature is not yet available in current telephony although such capabilities are being developed in the IP telephony protocols as described above. IM is also "used to maintain a sense of connection with others within an active communication zone." When IM was used in office environments, people initially would start using IM to converse, but they would switch media to a phone for more responsive interaction in the course of a single communication event.

IMPROMPTU brings together the advantages of IM by providing information on the availability of the users while improving the user interface for switching media between an awareness channel and a real time communication channel. Instead of having to switch from typing on the computer to speaking on the phone, one can use one device for both. Normal phones are tedious for unprompted informal and brief message exchange, however IMPROMPTU facilitates such exchange by providing Garble Phone[26] awareness channel, call screening functionality, and the subsequent transition of connection to real time media for direct voice connection with the communicating party from a single device.

3.7. Summary

This section has illustrated some related work that has guided the design and implementation of IMPROMPTU. Audio servers have provided an architecture for desktop applications to access various audio services on the desktop, and IMPROMPTU takes this a step further to provide audio applications of varying characteristics to be accessible from a single mobile client. The architecture provides distributed speech recognition and text to speech services that can be used by the applications to acquire user input from speech and present text using synthesized speech. IMPROMPTU overcomes traditional call setup models implemented in current IP telephony and mobile telephones to provide users with more flexible negotiation during the call setup process. It provides an awareness channel that can be used by callers to decide whether to further interrupt the receiver and a similar screening channel on the receiver side to provide more information about the caller before getting fully connected. In addition to traditional telephony and enhanced telephony, IMPROMPTU supports audio applications with different characteristics making it a ubiquitous audio client for accessing audio based applications. With a single appliance, people can access their MP3 music, listen to their favorite radio station and enjoy their audio book collections, while getting timely news updates. Based on the need for non visual interfaces for mobile users, IMPROMPTU also provides a simple elegant user interface based on the speech and tactile interfaces with auditory feedback to guide users in accessing the multiple applications it supports.

CHAPTER 4. USER INTERFACE AND APPLICATIONS

This chapter introduces IMPROMPTU client's user interface design and the applications that have been implemented. The first section highlights some key features of the user interface that are used to interact with the applications. IMPROMPTU uses a combination of button, speech and auditory interfaces to allow the user to navigate through different applications and interact with each of them. In the second section, the applications that have been implemented for IMPROMPTU are described in greater detail. IMPROMPTU supports applications with different properties and resource requirements. These factors determine how the user interacts with them and how the applications access resources on the client and the distributed services.

4.1. User Interface

This section presents the user interface design of the IMPROMPTU client. The goal of the design was to make it as simple as possible and as intuitive as possible to access different applications. The challenge is trying to design a coherent user interface for accessing multiple applications from a single client. A combination of button input, speech interface and audio cues are used. The IMPROMPTU client allows users to browse through different applications and use the alerting cues to determine which applications are active and which applications need attention. "Push to talk" functionality is essential for providing users with an effective speech interface. Once an application is activated, different applications require different user interfaces. Buttons and voice commands are used in combination to provide serial and random access to the applications. Some applications make use of the Speech Service to provide speech interfaces to browse and access the applications.

4.1.1. Browsing Applications

Users can browse through the applications serially, by using browsing buttons or randomly, by activating them using voice. As the user browses through the applications by going LEFT or RIGHT. As the user browses through the applications, distinct audio alerts are played for each application and the application is immediately activated if the user discontinues browsing. The following diagram illustrates the browsing process.

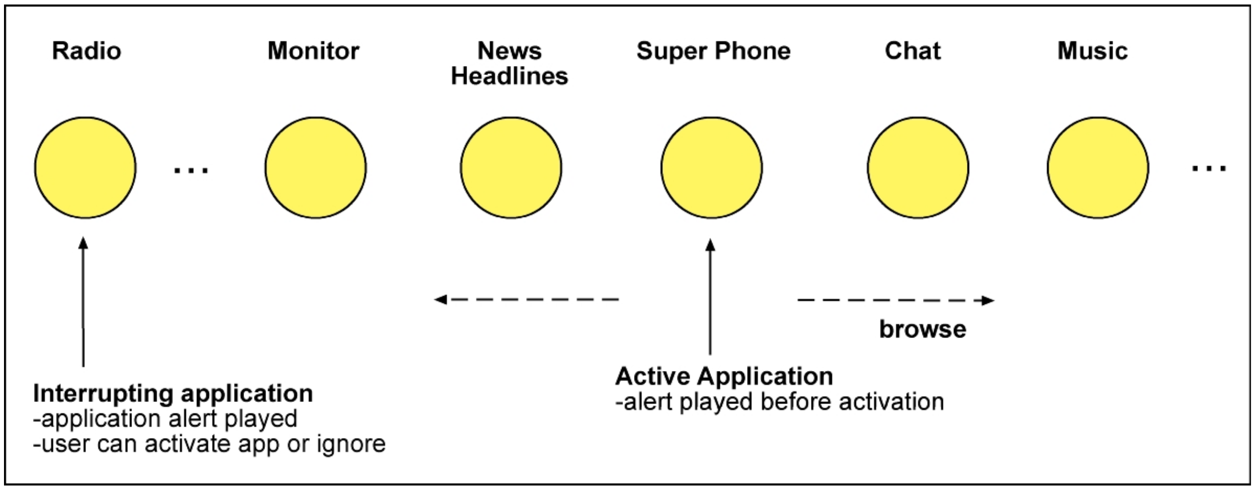


Figure 1. Browsing applications.

If the user does not want to have any applications active, he/she can either browse to the Sleep application, press the deactivate button which jumps to the Sleep application, or voice activate it by saying "sleep". The sleep application is an idle application that basically does not do anything. Once an application is activated, further user input will be processed by the application except for the browsing commands. After the user has finished using an application, the user can move to another application by browsing or voice activating the new application.

4.1.2. Buttons

The client captures key input from the user and forwards it to the Application Manager, which in turn forwards it to the applications. The hardware buttons on the iPaq are used to control the applications and send commands to the Application Manager. These keys are mapped in the following manner.

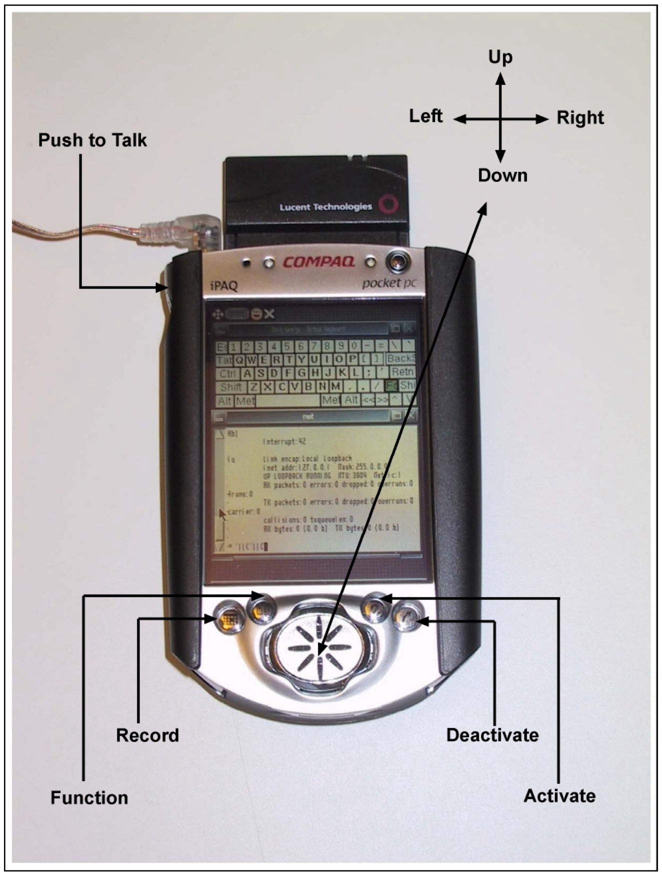


Figure 2. Mapping of buttons on the iPaq for providing tactile interface.

The activate button and the deactivate button send `ACTIVATE` and `DEACTIVATE` messages respectively to the Application Manager. Other key presses or user input send the `APP` message to the active application. Button pressing and releasing can be detected and the number value corresponding to the event is sent as part of the message. If an application is already active and the "activate" button is pressed, the client checks to see if there are any pending applications (applications that have alerted for attention) and the next application in the pending queue becomes active. The deactivate button deactivates the active application and switches to the Sleep application.

The UP/DOWN buttons are used in the applications to usually browse and navigate inside the application. This is adequate due to the serial nature of audio contents. FUNCTION button is the only other available button that is used by applications to implement specific functions. For example, in the Super Phone application, the FUNCTION button is used to hang up or make calls. Buttons are used as an alternative to the speech interface and there is always more speech commands available to the application than button commands.

4.1.3. Alerting and Interruption

VoiceCues[27], which are audio signatures that identify people and auditory icons, are used to provide users with awareness and feedback on the currently active users, active applications and the different applications that users browse. When a new application comes online or an application goes offline, the user is also notified with distinct auditory cues.

Applications have their own distinct alerts that get setup by the application developer. However, users can customize these alerts by mapping them to different alerts when they subscribe to these applications from the web interface. Every time a user browses through the applications (serial access), each application plays a short audio cue to indicate which application the user is accessing. When an application suddenly comes online or has certain content that requires user's attention, the current application is briefly interrupted for the alert to be played back. The user can activate the application that is interrupting or just ignore it. After 20 seconds, the application is removed from the pending queue, although it can still be accessed manually by activating it through browsing or through voice.

Each user also has his/her own personal alerts, which are played back when he/she tries to call another user. This alert is also used in the Chat application to notify active users that one has joined the chat. All these alerts are WAV files that are specified by an HTTP URL.

4.1.4. Push to Talk

Push-to-talk functionality is added in order to allow more effective use of the Speech Service. Push-to-talk means a button that needs to be pushed and held down to talk and released to stop talking. The push-to-talk button is the button on the upper left hand corner, which is used to enable and disable push-to-talk mode. When the button is pressed, the push-to-talk command is forwarded through the Application Manager in order to be delivered to the Speech Service. For all applications, audio is forwarded to the Speech Service when push-to-talk is enabled. Also, when the button is pressed, the client pauses the audio output to the speakers in order to prevent the output from interfering with the voice input. When a word is recognized, the client plays a short chime to indicate that there was a successful recognition. The recognized word is streamed back to the client to be displayed on the screen in order to provide feedback to the user with the recognition result.

Audio is normally streamed directly to the applications that require both speech recognition and audio input from users. However, while the push-to-talk button is pressed, the audio is forwarded to the Speech Service. This would be required when the user wants to activate another application or provide speech commands to the application. The user pushes and holds the push-to-talk button and speaks the application name in order to activate an application. The user releases the button after speaking the utterance. If something is recognized, the user will hear a chime sound.

4.1.5. Speech Commands

There are two levels of speech commands available to the users. These are the Application Manager level commands and the application level commands. The application level commands will be described in more detail in the next section. The Application Manager level commands are available globally from whatever state the user is in and they are:

Commands	**Behavior**
"What are the applications?" or "applications" for short	It tells the user which applications are online and available using synthesized speech.
"Where am I?"	Application alert is played back to indicate which application the user is currently active.
Application names such as "Music" or "News"	Activates the application and plays the application's alert as it is activated to give user feedback.

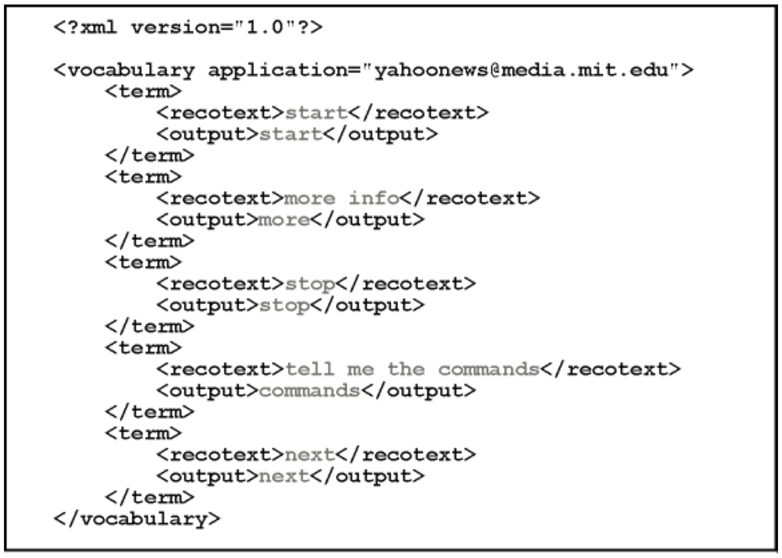
Table 1. Application Manager level commands.

If a user has been idle for a while, the user may not remember the previous state of the applications. "What are the applications?" or "applications" is used to find out what applications are available and online. "Where am I?" is used to see in which application the user is currently in. The user may have left off listening to some music and had the music playback paused. The user may decide to browse to the next application and to browse back to find out which application the user was in or he/she may say, "Where am I?" The latter speech command plays the application's alert. Finally, all the application names are available to the user to jump to different applications from wherever they are.

4.1.6. Application User Interface

Each application has a distinct user interface that consists of a combination of button presses, speech input, audio alerting and notification. All user key input from the client is forwarded to the applications by the Application Manager to be processed by the applications. The applications can support any kind of key input from the user, although the applications that we have implemented mostly respond to button presses on the IMPROMPTU client. This corresponds to a single key input, in order to keep the user interface as simple as possible. However, some applications like the Super Phone require telephone numbers or user ID's as user input which needs to be typed in through the software keyboard.

The speech interfaces for the applications are supported through the application specific dynamic vocabularies and grammars that can be loaded and unloaded as applications are activated and deactivated. The following is a sample vocabulary defined for the news application. These vocabularies constitute the application level speech commands.



```xml
<?xml version="1.0"?>

<vocabulary application="yahoonews@media.mit.edu">
    <term>
        <recotext>start</recotext>
        <output>start</output>
    </term>
    <term>
        <recotext>more info</recotext>
        <output>more</output>
    </term>
    <term>
        <recotext>stop</recotext>
        <output>stop</output>
    </term>
    <term>
        <recotext>tell me the commands</recotext>
        <output>commands</output>
    </term>
    <term>
        <recotext>next</recotext>
        <output>next</output>
    </term>
</vocabulary>
```

Figure 3. XML formatted vocabulary definition.

In the above figure, the vocabulary XML file is formatted into different "terms" that can be recognized and each term consists of a recognizable text and an output. This is to allow different words to be spoken for the same operation. For example, the users can say "tell me the commands" or "what are the commands?" and in response to either of the inputs, the speech recognizer would send "command" to the application.

Voice activation of applications is also done through the dynamic vocabulary. Each application has a name, which is defined by the application developer. This application name is added to the application vocabulary each time an application registers and it is removed when it unregisters. Users can ask "What are the applications?" in order to find out what applications they can activate.

The speech interface is very sensitive to ambient noise because the microphone used is a standard omni-directional microphone built into the iPaq. The iPaq is not easy to take apart, but if an external noise-canceling microphone could be attached, the speech interface could be made more robust.

Applications can alert users to interrupt, to notify of event changes in the application and to give feedback to the users about the status of the application. Interruption occurs when an application alerts the user when there is already another active application. Notification of event change occurs during the use of the application when the status of the application changes. Finally, the application gives feedback to the user when certain actions have been successfully or unsuccessfully processed.

4.2. Applications

In this section I describe some applications that have been implemented in the IMPROMPTU architecture. Each application defines its own application properties and these properties determine how the client interacts with the application when the application is activated. When the client receives the application information from the Application Manager, it creates an application object on the client side that interfaces with the remote application. Once this is setup, the control commands are all forwarded through the Application Manager. However, audio connections are made directly between the endpoints to reduce any latency. This was a major design requirement for IMPROMPTU since latency affects the usability of a real time audio application. Also applications are only allowed to send audio when activated through the Application Manager upon clients request. If they are not active, they will either wait until they are activated or they may send ALERT messages to the client to ask the user for permission to stream audio.

The following properties need to be set by the applications:

Property	Details
Application ID	The unique application identifier such as music@media.mit.edu
Application name	The application name, which is also used by the speech recognizer for voice activation. Therefore, one should be considerate in picking the name of the application for good recognition results.
Application RMI URL	This only exists for Java applications. The URL is of the following format: //<IP address of hosting machine>/<application ID>
Application port	The port application is listening for TCP audio connections or where UDP socket is bound for UDP audio input. The UDP port will be updated for each user when the user comes online and connects to the application. (See UDPPORT message description)
Application alert URL	The HTTP URL where the application specific alert resides.
Application transport type	Indicates whether the application supports TCP or UDP audio connections. Those applications that are sensitive to delays use UDP to transport audio, while those that are less sensitive, but require reliable audio communication, uses TCP to transport audio.
Application IO type	Indicates whether the application sends and receives audio (full duplex) or just sends only or receives only. There can also be application with "no audio" IO type. These applications are not involved directly with communicating audio with the client. They are facilitator applications that help two different clients connect or the client and another application communicate using audio.
Application sampling rate	The audio sampling rate that the application uses to play or record audio. 22050 Hz is default.
Application channels	The number of audio channels the application needs. Mono default.
Application speech requirement	Indicates whether the application requires both text to speech and speech recognition, only one of them, or none of them. Default is NORMAL, which does not use any of the speech services.
Vocabulary URL	[OPTIONAL] Vocabulary URL of the application if it uses the speech recognizer.
Grammar file name	[OPTIONAL] If the application requires grammars, the grammar file name needs to be specified. The grammar file also needs to be accessible locally by the speech recognizer.

Table 2. Properties that IMPROMPTU applications define when initializing the application.

There are applications that require direct audio channel connections to the application themselves along with those that do not require audio channels. Some applications have audio interactions directly with the client and there are some applications that act as an intermediary between two entities by managing state and intermediating control messages between those two entities. The Super Phone does not require any audio access when setting up calls between iPaqs, although some applications such as the Recorder will require duplex audio access since one can record messages and play back recorded audio directly from the application.

4.2.1. Characteristics of the Applications

IMPROMPTU applications are distributed applications that are accessible remotely from IMPROMPTU clients. Applications define their own properties during initialization, and the client uses these properties to determine how to interact with the application. Applications require registering with the Lookup Service to make themselves available to the clients. The following chart shows the different characteristics of different audio applications.

	Baby monitor	Presence info (Awareness)	News broadcast (Radio)	Phone	Personal notes/ reminders	Call screening (Preview)	Public chat (IRC type)	Private chat	Audio books	Music
Real time[1]	X	X	X	X		X				X
Archived[2]			X		X		X	X	X	X
Asynchronous							X	X		
Delay permitted	X	X	X		X	X			X	X
Continuous (No breaks)	X		X	X	X	X	X	X	X	X
Tolerate breaks	X	X				X				
Regular/ Timely[3]			X		X					
Interactive	X		X	X	X	X			X	X
Active (Pushed)	X	X	X	X						
Passive (Polled)	X	X	X		X	X				X
Single user	X	X			X	X		X	X	X
Multi user	X	X	X	X			X	X	X	X
Limited access	X	X			X	X		X	X	X
Public access			X	X			X		X	X

[1]Generated in real time
[2]Originally archived, since any audio content can be archived
[3]The content is regularly streamed

Table 3. Application characteristics

Above table illustrates the varying characteristics of different audio applications. They may produce real time audio from live situations, events or concerts or serve archived audio from existing audio contents previously recorded. Some applications do not depend on the quality of the audio as in awareness applications, since users do not need real time communication, but a sense of awareness, which can be inferred even from discontinuous audio. Delays are not permitted for real time person-to-person communication, but on other cases they can be tolerated. News type of applications generate timely content that changes with time and these contents would be adequate to be pushed when the content is considered important to the user. This would require alerting the user and these types of applications are classified as active. Applications like chat would involve multiple people accessing it and conversing through the application, but they may have different access privileges just like Internet chats. There could be public chats that anybody could join, or private chat rooms that can be used by certain family members. The following sections describe in greater detail, the applications that have been implemented for IMPROMPTU.

4.2.2. News Headlines (Java)

This is a speech-enabled application that allows users to access timely "Yahoo! News" headlines using speech recognition and text to speech. Users can speak different commands or use UP/DOWN buttons to browse through the news. When the user first activates the application, either by voice or by browsing with buttons, it plays the News application's alert and streams the first headline. Users can either say "next" or "more" to go to the next headline or get more information about the headline. The UP/DOWN buttons allow users to browse through the headlines. Users can press the FUNCTION button instead of saying "more".

A Perl script written by Stefan Marti retrieves news from the web every two hours and makes it available as an XML formatted file, which the news application retrieves to deliver to the user. The following figure shows how the news is formatted. Each item is composed of a headline and several sentences that contain more information about the headline.

```
<?xml version="1.0"?>
<news title="Yahoo! News for Friday August 10 12:20 AM ET">
                                   .
                                   .
                                   .
<item>
<headline>U.S. Offers China $34,576 for Plane-Collision Cost</headline>
<details>
<sentence>After rejecting Beijing's demand for $1 million, the
United States is sending China $34,576 to pay for support of a crippled
U.S. Navy surveillance aircraft that collided with a Chinese fighter jet
in April, U.S. officials said Thursday.</sentence>
<sentence>The officials, who asked not to be identified, told Reuters
there was no word yet on whether China would accept the payment, which
was en route to the U.S. Embassy in Beijing for transmission to the
Chinese Foreign Ministry.</sentence>
</details>
</item>
                                   .
                                   .
                                   .
</news>
```

Figure 4. XML formatted news content.

News from other sources can also be accessed through this application if they can be structured in the supported format.

Both the vocabulary and the content of the news are available through an HTTP server making it flexible where the application may be deployed. Further development can be done to allow the application to alert users when new news content is available, or use the Profiler Service to profile users for more customized content.

4.2.3. Music Player (Java)

There are two types of music playing applications. The first application is a server side jukebox, which pushes music to the client. The second type is the pull type of music application. However, for the user, these are not differentiated.

The client side pull player receives URLs and makes use of HTTP to transport MP3 audio. Decoding is done on the client side and code from Splay[28] has been adopted to handle this. The application responds to user input by sending back URLs as the user browses through the music list.

The push version, however, fits the IMPROMPTU architecture better. The server side application handles the retrieval, decoding and streaming of music over the network to the client. The music player plays MP3's from the user's `public_html/mp3` directory. The MP3 files need to have world read permissions. The music player also accepts various speech commands and button input. Users can say "random" to jump to a random song in their play list and use "start", "stop", "next", and "back" to browse through the songs. The FUNCTION button is used to select random songs and UP/DOWN buttons are used to skip to previous and next songs in the play list.

4.2.4. Chat (C/C++)

Ideally Chat would be a synchronous voice conferencing type of application. However, due to the difficulty in mixing audio, the Chat application implemented in IMPROMPTU is an asynchronous chat application. Chat is a multi-user application where a group of people (your family members or your friends) can join in any time freely over the wireless connection and hold a voice conversation. The chat contents are recorded and available to be browsed chronologically. Users can enter the chat room and hear previous chat content that has been recorded so that they may catch up on what has been missed instead of having to interrupt the people in the current chat. This is a feature that has been adopted from normal text chat applications. If the user needs to step out from a chat, the user can easily catch up again using this background "history" chat feature.

When the user activates the chat application, he/she may start talking to the users currently online, or the user may wish to browse through the previous chat recordings. The user presses the UP or DOWN buttons to browse. The DOWN button goes to the more recent messages. In our current implementation, each recording contains a message left from a single user just like the postings on the web bulletin boards. If there are multiple users online, any new message that gets posted by a user will be broadcasted to all the active users. If the other users are not doing anything, the message will play back immediately, but if they are browsing or recording, they will hear an alert and the message will be played back immediately after the user stops browsing or recording. The messages get queued up for play back when they are busy doing something in the Chat application.

4.2.5. Recorder (C/C++)

The recorder application serves as a personal audio note taker or as a means to record audio from other applications. It allows users to record personal voice memos, browse through the recordings, record the content of another application, and allow other applications to record. For example, Super Phone uses the Recorder to provide voice mail functionality to the users. The audio is transported using TCP. If the user activates the Recorder, the user uses it as a personal recorder. If another application is running and the user presses the record button, the audio from the application is saved to the recorder without activating the recorder. In the recorder application, this is saved into the `userID/appID` directory. Users can browse through the recordings by using UP/DOWN buttons or by saying "next", "repeat" and "back" and the recordings will be played back sequentially. Users can also press the FUNCTION button or say "delete" to delete the current recording.

When the user activates the recorder, it does not start automatically, unlike other applications. The user has to manually press the record button or say "record" to start recording. The Application Manager interprets the record button press and checks to see if the Recorder application is online. If it is online, it forwards the button press event to the Recorder, which then starts recording. If the application is not online, the button press event is ignored. If it is online, the client forwards the audio that is being played back on the client to the Recorder to save the live audio.

4.2.6. Radio (Java)

The Radio application streams real time radio from a tuner that is connected to the application host. The application uses TCP audio connections and it is a one-way, send only application. When the user activates it, radio will be streamed in real time. Multiple users can access the application, but there is only one tuner so they will all be listening to the same channel.

In order to support multiple radio channels on different frequencies, a separate tuner is needed due to the lack of a computer-to-tuner interface. With an adequate computer-to-tuner control interface, there could be more elaborate ways to implement this application to allow users to switch channels. Radio applications can also obtain content from an Internet radio station if the audio content can be decoded (e.g. RealAudio type to PCM).

4.2.7. Audio Book (Java)

The Audio Book allows the user to browse through more structured audio content. One way to implement the application is using the text to speech engine to stream structured text content. However, the content streamed from the text to speech engine is not pleasant to listen to for long periods of time.

Human voice recorded content is necessary in order to provide users with adequate listening experience for those who access the application frequently for long periods of time while on the move.

As long as the content is sequentially structured (segments of wave files divided into paragraphs or chapters), users can use the speech service to navigate using speech or using the buttons. Currently, we have several segments of audio books and a book marking functionality has been implemented. While a user is listening to a book and is interrupted to attend to another task for a while, the Audio Book application keeps track of where the user stopped listening. When the user returns to that book, it starts playing back from about twenty seconds ahead of where the user left off. This allows the user to recall the last parts of the story where he/she stopped. Users can switch books by using UP/DOWN buttons or using speech commands "next" and "previous".

4.2.8. Baby Monitor (Java)

The Baby Monitor uses the "always on" connection to monitor environments such as a baby's room and deliver the audio data to the user who is always connected and subscribed to the application. This is a special use of a more general monitoring application. The administrator of a monitoring application may restrict access to certain users or broadcast the audio to everybody in a community. Moreover, the application can alert the user when sudden changes occur in the environment. In the case of the Baby Monitor, it is activated when the volume level captured from the monitor's microphone goes above a certain level. A baby's crying immediately alerts the users and opens a real time audio channel to her mother or father, depending on the application settings.

Some monitors may have only a one-way channel established and some may have a two-way channel established so that the user can talk back to calm down the baby. Once the user activates it, the monitor starts streaming audio captured by the microphone. When the environment's audio level changes it would alert the user for attention. Such a monitoring application can also be continuously streaming in the background of the current active application if audio mixing is available, so the user can be constantly aware of certain places or certain people. The application provides a level of awareness in voice communication before an actual interaction is established, which is not possible with the traditional telephony.

4.2.9. Super Phone – Enhanced Telephony (C/C++)

This application enables new ways of connecting to people to have real time conversations. Users can use the IMPROMPTU client to make outgoing calls to and receive incoming calls from real phones. Users can also call each other on their iPaqs. While normal telephones only support three distinct states of either being connected, alerting or disconnected, IMPROMPTU allows alternative ways of getting connected to people using mobile devices by leveraging the always connectedness of IP network.

As Chris Schmandt suggested, the IP network introduces different levels of communication where one has many more alternative ways of communicating with another person. As there are means to communicate textually with different expressions through postcards and letters, one can leave a voice mail directly not to disturb or send a text message instead of trying to reach a person using voice to avoid intrusive alerting. IMPROMPTU extends this and allows different levels of connectedness including a background awareness communication channel between the caller and the receiver.

IMPROMPTU improves the limited alerting and call setup model in traditional telephony by notifying users with different alerts for different connection states established between the caller and the receiver. The following diagram describes the different connection states.

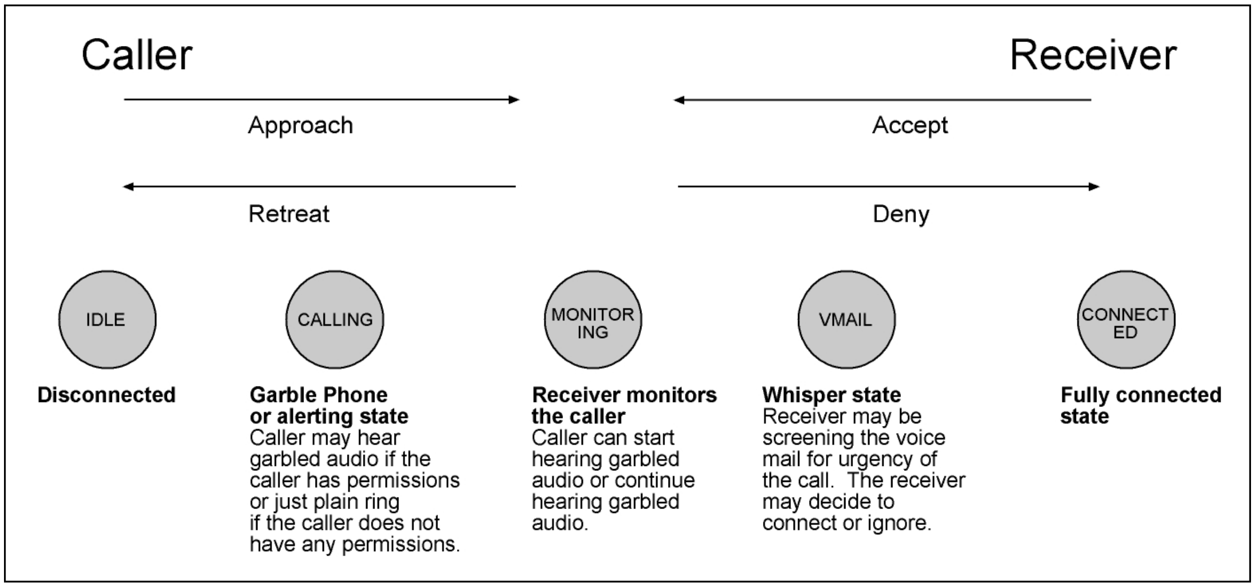

Caller
Receiver
Approach
Accept
Retreat
Deny
IDLE
CALLING
MONITOR ING
VMAIL
CONNECT ED
Disconnected
Garble Phone or alerting state
Caller may hear garbled audio if the caller has permissions or just plain ring if the caller does not have any permissions.
Receiver monitors the caller
Caller can start hearing garbled audio or continue hearing garbled audio.
Whisper state
Receiver may be screening the voice mail for urgency of the call. The receiver may decide to connect or ignore.
Fully connected state

Figure 5. States of the caller and the receiver in the Super Phone application.

The application uses the normal person-to-person physical interaction metaphor and models it over the mobile devices. The caller approaches to, or retreats from the receiver whom the caller wants to talk to, while potentially examining the receiver's state. This is similar to approaching a person to try to get his/her attention. If the person seems to be too busy, one may retreat and come back later. In the same way, if the person seems to be busy from the garbled awareness channel, the caller may decide to retreat. The receiver may decide to observe what the caller says before deciding to drop what one is doing and get connected. If the receiver is busy, the application lets the caller go to voice mail, but even at this point, the receiver can get involved in the call after screening the call. This is similar to a real person-to-person interaction where the receiver could have a view of the person trying to interrupt. The person being interrupted may see some visual signals before being actively interrupted, and if the interrupting party seems to have an urgent message, the receiver may decide to allow the interruption and talk to the person.

Garble Phone

To provide awareness to trusted friends and families about one's communication status, audio can be used to publish presence information to peers. The "always on" connection of the Internet implies that a user's device can always be monitoring one's audio environment. The user can choose to let others know about one's status by allowing the device to send out garbled audio to peers[26]. Any peer who wants to talk to the user will be able to listen to this audio before making a decision to disturb the user. Such an awareness channel would allow the caller to get a general sense of the receiver's status, but the content would be garbled and unintelligible. As this scenario suggests, the flexibility of the IP network introduces various ways of call control, where call setup decision is not only made by the receiving party, but also by the network services (by consulting the Profiler Service or Presence Service) or by the calling party through the utilization of the awareness information. In the case when calls are diverted by the system, auditory or visual cues are used to notify the user about the call state changes.

To protect garbled audio from packet sniffing, the garbling is currently done on the client. As the following diagram illustrates, six segments (currently, 2 Kbytes per segment) of audio are initially read from the microphone, and the subsequent audio segments are randomly swapped with one of the six segments before it is sent over the network.

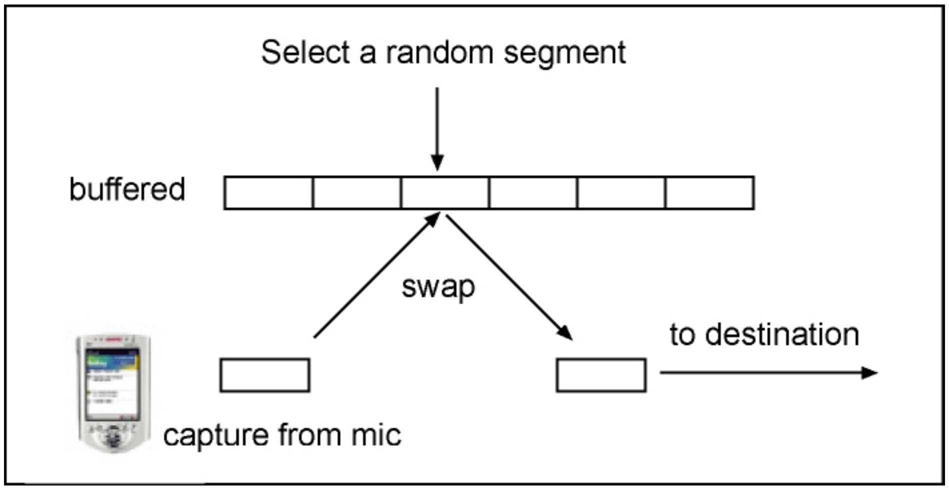


Figure 6. Audio garbling algorithm.

Whisper – Call Screening

In presenting audio streams, audio connections can change from non real-time to real-time audio connections and vice versa depending on the need for the user to respond. Call screening is a feature where the audio connection changes from non real time (delayed real time) to real time audio during the communication. Initially, when a caller is routed to voice mail, non real-time audio can be simultaneously transmitted to the user to screen the call. At this point, the user can decide that it is important to talk to the caller and decide to get connected. This makes the audio that has been forwarded through the voice mail to be directly routed to the user, and the connection will change to a real time connection.

In the current implementation however, the audio does not get forwarded through the voice mail, but the client serves as a proxy to forward the audio from the Super Phone or another client to the voice mail. The Recorder provides the voice mail functionality to the Super Phone. The Super Phone signals with a `RECORD` message when the caller is trying to leave a voice mail. When the Application Manager receives this message, it signals the Recorder to start recording and the client starts forwarding audio to the Recorder. As a result, the call can be effectively screened as it is forwarded to the voice mail. While the user is screening, the user can press UP button or say "approach" to get connected. This stops audio forwarding and recording and establishes a full duplex connection with the caller. The advantage of this implementation is that there is no additional delay when screening the call. The disadvantage is that it uses up client's bandwidth.

The Super Phone application has the most complex user interface among all the applications. Users can dial a number using the software keyboard, dial by voice or dial by browsing. When the user activates the telephony application, he/she has the choice of reaching another IMPROMPTU user or a phone number. When a phone number is entered using the software keyboard, it is just like a normal phone call and connects to that phone number. However, if someone dials from a phone to an IMPROMPTU user, the enhanced telephony features are available, including garble phone and call screening. In order to call IMPROMPTU users, one can browse through the list of users provided by the application using the UP and DOWN buttons. As one browses through the list, the client plays distinct alerts for each user in order to distinguish between different users. If the user says "call" at this point, the client starts establishing a connection to that user. One can also activate a connection by pressing the FUNCTION key or by manually typing the user ID. IMPROMPTU users that are connected may also be called using voice. To call John, one can just say "Call John" from the Super Phone and that would establish a connection to him.

The approach/retreat metaphor is used to control the call setup process. By approaching, the caller obtains a more intimate/superior connection. By retreating, the connection becomes less intimate. The caller approaches a user by going UP (pressing UP button) and retreat by going DOWN. Users can also say "approach" and "retreat." The following state diagram illustrates more clearly the different transitions that occur during a call.

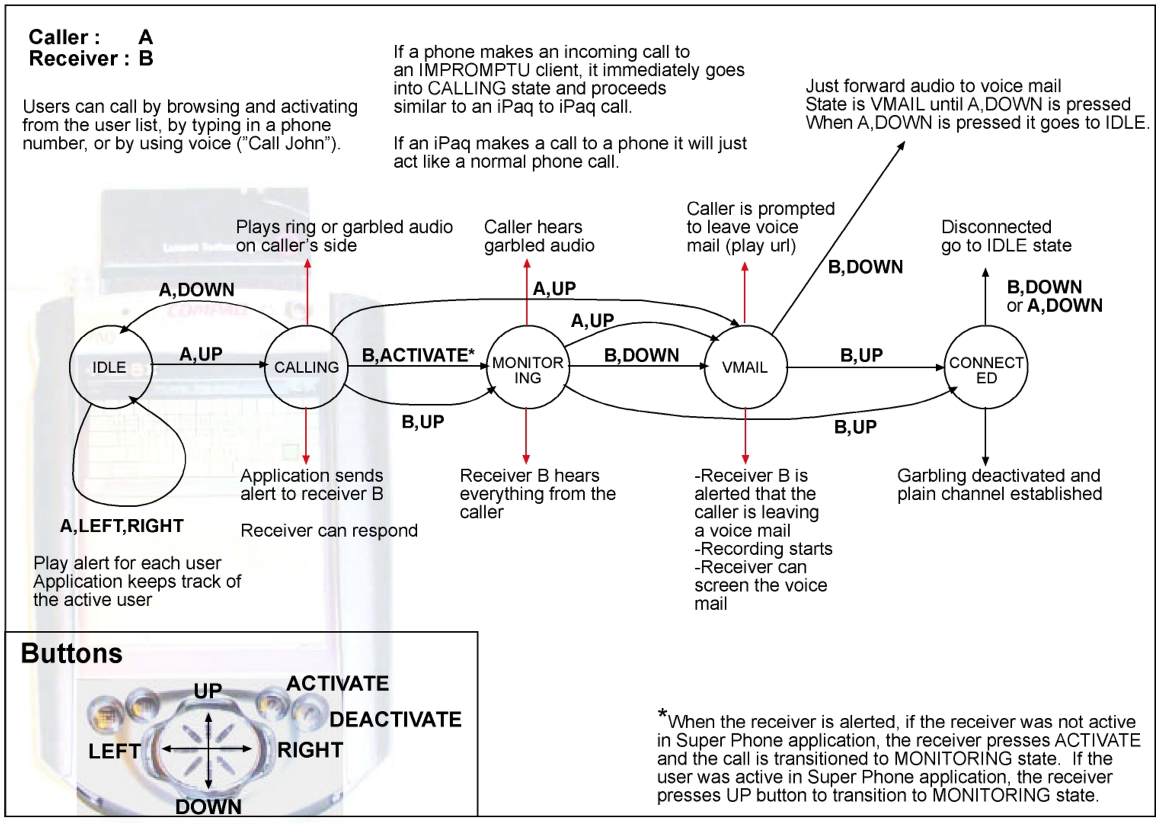

Caller : A
Receiver : B
Users can call by browsing and activating from the user list, by typing in a phone number, or by using voice ("Call John").
If a phone makes an incoming call to an IMPROMPTU client, it immediately goes into CALLING state and proceeds similar to an iPaq to iPaq call.
If an iPaq makes a call to a phone it will just act like a normal phone call.
Just forward audio to voice mail
State is VMAIL until A,DOWN is pressed
When A,DOWN is pressed it goes to IDLE.
Plays ring or garbled audio on caller's side
Caller hears garbled audio
Caller is prompted to leave voice mail (play url)
Disconnected go to IDLE state
A,DOWN
A,UP
A,UP
B,DOWN
B,DOWN or A,DOWN
IDLE
A,UP
CALLING
B,ACTIVATE*
MONITORING
B,DOWN
VMAIL
B,UP
CONNECTED
B,UP
B,UP
Application sends alert to receiver B
Receiver can respond
Receiver B hears everything from the caller
-Receiver B is alerted that the caller is leaving a voice mail
-Recording starts
-Receiver can screen the voice mail
Garbling deactivated and plain channel established
A,LEFT,RIGHT
Play alert for each user
Application keeps track of the active user
Buttons
UP
ACTIVATE
DEACTIVATE
LEFT
RIGHT
DOWN
*When the receiver is alerted, if the receiver was not active in Super Phone application, the receiver presses ACTIVATE and the call is transitioned to MONITORING state. If the user was active in Super Phone application, the receiver presses UP button to transition to MONITORING state.

Figure 7. Enhanced telephony with Garble Phone and Whisper.

IDLE state is when there is no active connection and when users can make calls. When the user makes a normal phone call from the IMPROMPTU client, it transitions from IDLE to CALLING state. When the receiver picks up the phone, it transitions to CONNECTED state. This is exactly the same as a normal phone call.

When the caller calls a user on the IMPROMPTU client, the caller has the enhanced telephony features available. If the caller calls and has permissions during CALLING state, the caller hears garbled audio from the receiver side. Based on this audio, the caller may decide to approach further (UP) or retreat (DOWN). If the caller approaches further, the caller is transitioned to the VMAIL state. The receiver may actively respond to any incoming call at any point during the transition. If the receiver accepts the call by moving up during the CALLING state, the call is transitioned to the MONITORING state where the receiver can listen to the caller's audio in a clear channel. At this point, the receiver may decide to accept the call or send it to voice mail. If the call is in VMAIL state, the receiver may press UP to screen the call. Based on the screening, the receiver may decide to get connected for a live conversation or ignore.

4.2.10. Summary

The various applications described in this section have all been implemented and are functional. It illustrates the flexibility of the IMPROMPTU architecture to support various types of audio applications. The speech interface provides all the application specific commands. The button interface complements the speech interface to provide serial navigation functionality and some special operations using the FUNCTION key.

CHAPTER 5. SYSTEM SERVICES AND ARCHITECTURE

This chapter describes the underlying system services, architecture and the protocol that supports the applications and the user interactions described in the previous chapter. The IMPROMPTU architecture consists of several distributed services and distributed applications that are accessed using the IMPROMPTU client. The client is designed to be as thin as possible and the Application Manager for the client handles most of the resource management required to support the applications. It also arbitrates the activation and deactivation of applications. All applications are always on, but may not be sending any audio data since only one application can usually be active at a single moment due to the client's single audio input (microphone) and output (speakers) channels.

This chapter is divided into four sections. The first section describes the architecture of the IMPROMPTU system, including the high-level design and functionality of different components in the architecture. In the second section, an overview of the software architecture is presented with details of system implementation. The third section details the messaging protocol used among the distributed components of IMPROMPTU. Finally, the fourth section examines how the distributed components operate together to provide application services to the user.

5.1. High Level Architecture

This section provides a high level architectural overview of the components of IMPROMPTU. The IMPROMPTU system consists of several distributed components that cooperate to provide application services to the user client. What these components are and what they do will be presented. The following diagram illustrates the different components and how they are interconnected to provide service to the client.

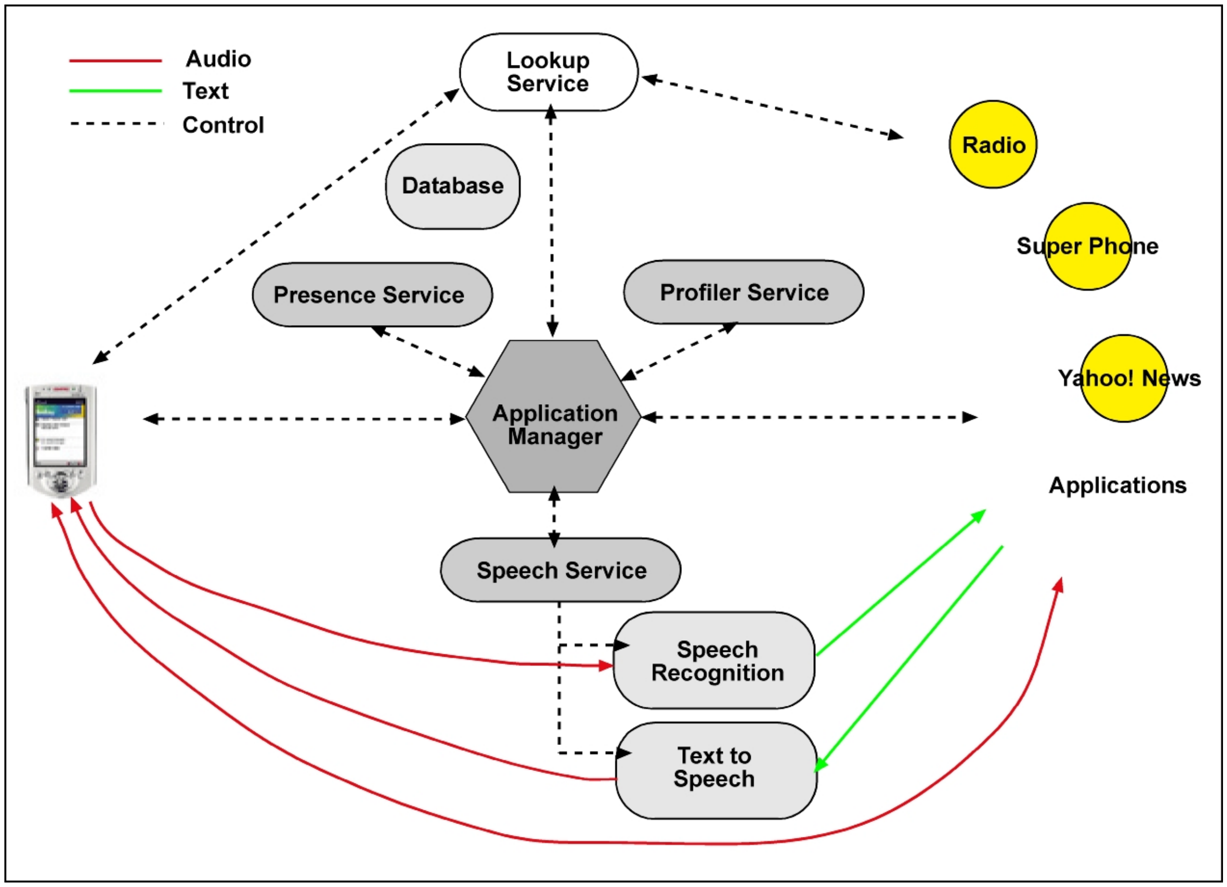


Figure 8. High-level architecture diagram.

5.1.1. Lookup Service

The Lookup Service is a global service in IMPROMPTU, which is shared by all users. There is only one instance of the Lookup Service in the IMPROMPTU network and it maintains information about the users and the applications that are online. It handles the resource registration and resource discovery in the IMPROMPTU network. Users need to register and authenticate with the Lookup Service when they join the IMPROMPTU network. When applications come online and register with the Lookup Service, they obtain information about the users that are online and which Application Managers to contact to register with those users. When a user client comes online, the Lookup Service provides a reference to an Application Manager that can serve the client.

5.1.2. Application Manager

The Application Manager is the core component of the IMPROMPTU architecture that handles client requests and manages applications for the client. It also communicates with the user's personal services (Speech, Presence and Profiler Services). Therefore, it serves as a communication hub for the client, applications and personal services. When the client initially logs on to the Application Manager, it registers with the user's personal services and contacts the Lookup Service for online applications. The Presence Service provides a list of applications that the user is subscribed to. Depending on the implementation of the application, the Application Manager needs to initialize them differently since C/C++ applications use a text messaging protocol for signaling, while Java applications use Remote Method Invocation (RMI). RMI allows Java components to transparently call methods of other Java components over the network without worrying about the networking details. However these calls are limited only to Java components. Please refer to the Software Architecture section(pg. 59) for details of how this is implemented.

When newly subscribed applications come online, they register with the Application Manager, which sets up the control channel with these applications. The Application Manager then notifies the client that these applications are online. Once the client is notified, audio connections are setup between the client and the application, allowing the client to access these applications immediately.

The Application Manager handles user inputs and other events from the client and application. When necessary, it forwards appropriate messages to either entity. The Application Manager also communicates with the user's personal services when needed. When an application that requires speech recognition is activated, the Application Manager sends the appropriate activation commands to the Speech Service. The Application Manager may also communicate with the Profiler Service to log the user's activities as the user accesses different applications.

5.1.3. Client

The following four major elements were the requirements for a right platform for the mobile audio client. The device had to:

1. Be **programmable** in order to support different types of applications.

2. Support **wireless LAN** to provide mobility and IP based communication.

3. Support **full duplex audio** to support real time two-way communication.

4. Be of a **small form factor** so that it is mobile without being bulky.

Various mobile computing devices and notebook computers have been explored. However, mobile phones had almost zero programmability. Handheld computers rarely possessed good audio capabilities. Many of them could just play very simple low quality audio snippets and some supported half duplex audio at most. This indicated how primitive the auditory user interfaces were on these mobile devices. Wireless LAN connectivity was also difficult to obtain on these devices. As for the notebook computers, although they supported full duplex high quality audio and wireless bandwidth, they were always too big. As a result, the Compaq iPaq was chosen as the hardware platform for the IMPROMPTU client since it met the audio, network, programmability, and form factor requirements.

The IMPROMPTU client runs on a Compaq iPaq with Familiar Linux 0.4[29] distribution - a distribution ported to support the 200 MHz Intel StrongARM[30] processor on the iPaq. The iPaq supports full duplex audio capabilities, and high bandwidth wireless connectivity through a standard 802.11b wireless LAN card. However, it has some limitations in terms of the audio playback and record capabilities. It does not support audio at 8 kHz or 8 bits, making it not possible to save bandwidth by sending lower quality audio for certain applications. The iPaq can only playback 16-bit audio at sampling rates above 16 kHz. The IMPROMPTU client currently uses 22050 Hz, 16 bit mono audio by default, but the format can be changed dynamically depending on the active application settings.

The client software is a framework that supports different types of audio applications developed for the IMPROMPTU architecture. The client supports multiple applications through a common interface to the remote applications. The microphone and the speakers are shared resources on the client and the active application has access to these audio resources along with receiving user input from the client. The user input is captured and forwarded through the Application Manager and a text messaging protocol is used to communicate with the Application Manager.

5.1.4. Speech Service

IMPROMPTU uses the IBM ViaVoice speech recognizer and text to speech synthesis, running on the Linux operating system. We developed an audio library for these engines to handle audio input from the network, and a Java layer to integrate it with other IMPROMPTU components. The speech recognizer can use different vocabularies and grammars. It processes Application Manager commands (such as activating applications using voice) and application specific commands that are loaded when applications are activated.

When a user logs on to the Application Manager, the Speech Service is activated along with other services. The user ID that is passed to the Speech Service identifies the user it will be serving. The Speech Service builds a vocabulary of applications from the application names that it obtains from the Application Manager. It also builds the application specific vocabularies from the applications' vocabulary URLs (a web accessible XML file) so that the client may access the applications using speech.

When the client activates an application that requires the Speech Service, the Application Manager activates the application by passing the Speech Service information that includes the IP addresses and port numbers of the speech recognizer and the text to speech engine. The application uses this information to connect to the speech resources. The Speech Service is subsequently activated with the user ID and the activated application information. The Speech Service makes a decision whether to activate both the speech recognition and the text to speech or just one of them. It also lets the speech recognition engine know the application specific vocabulary. The recognition engine finds the socket connections of the client and the application from the hash table. It then receives input from the client to be recognized and sends the recognized text to the application. The text to speech engine operates in the opposite way. After finding the socket connections, it starts receiving text from the application to synthesize it and stream it to the client. The speech recognition engine has a push-to-talk interface, so it only receives input from the user when he/she pushes the push-to-talk button.

Applications may use just a dynamic vocabulary, just a grammar, or a grammar and a dynamic vocabulary. A grammar is a set of rules for recognizing phrases with certain patterns. For example, in order to recognize "call <someone>" we use a grammar that matches anything that starts with "call" and ends with a user name. Grammars require the speech recognizer to have a compiled grammar file accessible locally to the speech recognizer. We use the network file system (NFS) to make application specific grammar files accessible to the recognizer, but a file transfer protocol could be used to transfer the compiled grammar to the speech recognizer during activation. But, this would delay initialization of the speech recognizer due to downloading time. Dynamic vocabularies are just a list of words that can be recognized and they can be dynamically loaded or unloaded. Commands such as "start", "stop" and "next" make up an application's vocabulary and they are defined through a vocabulary XML file.

5.1.5. Profiler Service

The Profiler Service profiles a user's activities through the client device by monitoring his/her application usage. It also maintains the priority of applications and peers according to their usage patterns and activity. This information is also used for prioritizing incoming requests by preventing certain peers or applications from interrupting when one is busy. Users can easily setup their profiles through IMPROMPTU's web interface. Currently applications and users are ranked in four levels, VIP, TRUSTED, NORMAL, and UNTRUSTED. The rule for interruption is that the applications on the same level or below can be interrupted, while any application with higher trust level cannot be interrupted.

The Profiler Service is important for mobile communication due to the need to control information overload and manage interruptions for mobile users[31]. The user's personal profile can be used to provide a more customized service to the user. Currently, the Profiler Service is very simple, but the architecture supports any extensions to it. For example, providing context awareness to mobile phones is an active research area[32], and the Profiler Service may implement the functionality that analyzes data about the context of the user and provides a means to the Application Manager to use that information for intelligent management of the applications.

5.1.6. Presence Service

The Presence Service keeps track of subscribed users and applications and also notifies other users when a subscribed user's communication state changes. Users can ask for subscriptions to different users and applications through the Presence Service.

Presence is defined in [33] as subscription to and notification of changes in the communication state of a user. This communication state consists of the set of communication means, communication address, and status of that user. A presence protocol is a protocol for providing such a service over the Internet or any IP network[34].

5.1.7. Database

InstantDB, an open source database from Enhydra (http://www.enhydra.org), is used to store various user settings and application settings of IMPROMPTU. The database is implemented 100% in Java making it easy to integrate with the Java components of IMPROMPTU. It supports the standard Java Database Connectivity (JDBC) API and is RMI enabled, allowing the database to be run on a separate machine from other IMPROMPTU components.

The database stores user authentication information, user settings, application settings, subscription information and profiles of users. Authentication information consists of a user ID and a password. User settings are the user's full name, RMI URL's of the user's personal services, and the user's personal alert URL. The alert URL is the HTTP URL of a WAV audio file where the alert is stored. Application settings are application ID, application name, and the application alert URL. The subscription information indicates the user's subscribed applications and the user's buddy list. Finally, the database also monitors application usage of the user, which is used to prioritize applications. The information in the database is currently accessible from the IMPROMPTU administration website at http://luz.media.mit.edu:8080/impromptu. Users can modify their personal information and view their access logs. The administrator maintains application settings and subscription approval to the applications.

5.2. Software Architecture

This section provides more details about the lower level implementations of the system. Different components are described and what modules constitute them.

5.2.1. Applications

The applications are implemented either in Java or C/C++. The architecture is similar for both implementations, but Java applications use RMI to communicate with the Application Manager, while the C/C++ applications use text messaging over plain sockets to communicate with the Application

Manager and Lookup Service. The applications can be implemented in either language. Java application development is simpler, but audio handling in Java is slower and less powerful than in C/C++.

Java Applications

Java applications inherit the `impromptu.application.ApplicationBase` class. The base class implements a common interface `impromptu.application.IApplication`. The `ApplicationBase` implements registration/unregistration methods, activation/deactivation methods, subscription/unsubscription methods and audio resource setup methods. The actual application inherits from the `ApplicationBase` and implements individual user handling logic and application specific activation/deactivation methods.

Once the user is registered, the client makes necessary audio connection with the application and the network connection is stored in the `ConnectionInfo` class, which the application maintains in the `m_currentConns` hash map. When users activate the application, the application accesses the connection information to send or receive audio from the user.

C/C++ Applications

The C/C++ application architecture is similar to the Java application architecture, but it is a little more complex due to the lack of RMI for network communication. The applications inherit `ApplicationBase`, which provides similar functionality as the Java `ApplicationBase`. It uses the `MessageSocket` class wrapped with the `IManager` class in order to communicate with the Application Managers.

The `ApplicationBase` class also has two listener threads if an application supports audio connections and only one if it does not. The application needs a listener thread to accept connections from the Application Managers and to receive control messages from the client. The second listener would listen for incoming audio connections.

For each user that connects to the application, there is a corresponding `IManager` stored in a hash map (`m_currentUsers`). Applications can send messages and respond to client requests by using the IManager interface stored in this data structure.

For each active user (a user who has activated the application at least once), a user handler thread is created and the user specific information and thread controlling parameters (`UserConnInfo`) are stored in a hash map (m_activeUsers) that maps user ID's to the UserConnInfo based objects. UserConnInfo contains various public members including user specific data and user thread control parameters. Applications will usually create an application specific class that inherits from UserConnInfo to store application specific data used when handling users.

AppInfo

Applications use the AppInfo class to initialize application properties. The application ID, the application name, the RMI URL of the application (for Java applications), the IP address of the machine where the application is running, and the port that the application listens to for audio connections, the alert URL of the application, whether the application requires TCP or UDP audio connection, and whether the application sends or receives audio from client are all setup when the AppInfo object is initialized(Table 2). The audio format can be specified explicitly by assigning the sampling rate and number of channels. If they are not specified, 22050 Hz and mono channel is used by default. Currently only 16 bit audio is supported. Also, the application's usage of local audio resources can be specified if it requires recording from local audio input or playing back to speakers on the local machine that the application is running. If it is not specified, the application is assumed not to consume any local audio resources. Finally, the speech requirements of the application and the vocabulary URL are specified if it is needed. If this is not specified, the application is assumed to not require any speech resources.

5.2.2. Client

The client keeps track of active applications in a circular list. The active application pointer is used to interact with the application. Each application is encapsulated with the Application class. The Application class maintains the properties of the application and is used to send audio to, and receive audio from, the application.

There are four threads: user input thread, messaging thread, read and playback audio thread, and record and write audio thread. The user input thread captures input from the user and processes browsing, activation/deactivation, recording, and application specific inputs. The messaging thread processes messages received from the Application Manager and it also receives recognized text from the speech recognizer. The read/playback audio thread receives audio from the network and plays back to the speakers. The record/write audio thread captures input from the microphone and sends audio to the network. The client may send audio to the application, to another entity signaled by the application, or the speech recognizer. When the push-to-talk button is pressed, the audio is sent to the speech recognizer and the playback of audio is disabled.

Alerting is done through the Splay module, which is an open source WAV and MP3 streaming audio player. When alerts are played, it requires opening the sound device independently, so it uses condition variables along with other threads to share access to the audio resources.

5.2.3. Lookup Service

The Lookup Service consists of

```
impromptu.lookup.LookupService
impromptu.util.Listener
impromptu.lookup.LookupMsgHandler
impromptu.lookup.LookupInfo
```

The `Listener` class accepts connections from the client and applications and lets the `LookupMsgHandler` handle messages from them. The client only registers with the Lookup Service, while the applications register and unregister with it. Client unregistration is handled by the Application Manager when the client goes offline. During initialization, the LookupService checks to see if the IP address in `lookup.xml` is the IP address of the machine it is being deployed. The `lookup.xml` is accessible through an HTTP server. The `LookupInfo` class provides methods to obtain Lookup Service information from `lookup.xml`. This is a consistency check to prevent other components from not finding the Lookup Service, since the other components locate the Lookup Service through the `lookup.xml`. The Lookup Service maintains a list of Application Managers that are available to serve the users, a map of user ID's to Application Managers and a map of applications that are online. When a new user comes online, the Lookup Service authenticates the user and maps an available Application Manager to the user. The Lookup Service is connected to the database in order to handle user authentication.

5.2.4. Application Manager

The main components of the Application Manager are

```
impromptu.manager.AppManager
impromptu.util.Listener
```

```
        impromptu.manager.ManagerMsgHandler
        impromptu.application.CApplication
```

The `AppManager` class contains registration methods to register with the Lookup Service, activation/deactivation methods that also activate the Speech Service when necessary, subscription/unsubscription methods that communicate with the Presence Service and logging methods that communicate with the Profiler Service. The `AppManager` class also keeps track of a user's subscribed applications in a hash map.

The `Listener` listens for incoming connections from the client and C/C++ applications. Java applications use RMI, so there are no plain socket connections to the Application Manager. Once a connection from a C/C++ application is accepted, the Application Manager registers the application and creates a `CApplication` object, which is a thread that abstracts messaging through plain sockets between the Application Manager and the application. When a message to the application is received from the client, the Application Manager accesses the application through objects implementing the `IApplication` interface from the `m_applications` hash map, and forwards the message to the application. The Java applications implement `IApplication` and the `CApplication` class implements `IApplication`, providing a common interface for the Application Manager to interact with the application regardless of the implementation.

5.2.5. Speech Service

The Speech Service consists of three distributed components. The Speech Service Interface Layer (`impromptu.speech.SpeechService`), which is considered the Speech Service to the Application Manager, manages communication with the speech recognition engine and the text to speech engine. We use the IBM's ViaVoice speech recognition and text to speech engines. The engines are implemented in C, but we modified them to accept audio input from network sockets instead of a microphone. We also developed Java layers and Java RMI enabled them[35] to make them accessible remotely from different components of IMPROMPTU.

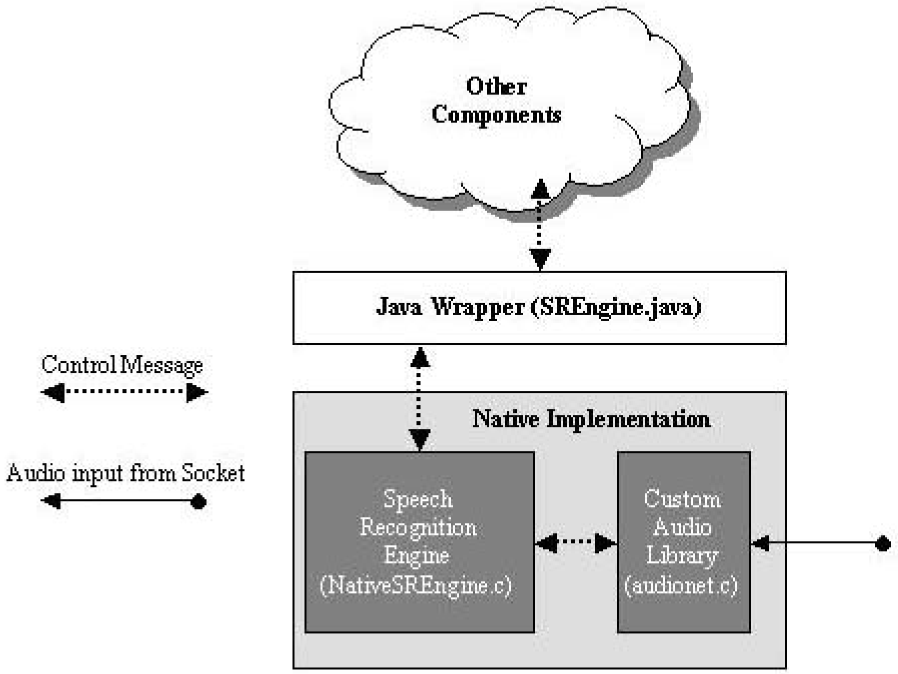

Other Components
Java Wrapper (SREngine.java)
Control Message
Audio input from Socket
Native Implementation
Speech Recognition Engine (NativeSREngine.c)
Custom Audio Library (audionet.c)

Figure 9. Speech Recognition Component[35].

The above diagram illustrates the different modules of the speech recognizer. The text to speech engine shares a similar architecture.

5.3. Messaging Protocol

The message protocol is described in this section. All messages consist of a command and a list of parameters. The command is in all capital letters and the parameters are strings separated by commas, followed by a new line indicating the end of the message. Currently the command and the parameters are delimited by commas, but since the message handling functionality has been modularly implemented, the messaging format can be easily modified.

5.3.1. Client and Lookup Service

The client obtains the IP address and the port number of the Lookup Service from the default `lookup.xml`, available through HTTP.

```
ONLINE,<userID>,<password>
Example) ONLINE,kwan@media.mit.edu,kwan
```

is sent to the Lookup Service in order to notify that the user has come online.

The Lookup Service returns the IP address and the port number of an Application Manager that is online and available to serve the user. The following message:

```
MANAGER,
<manager URL>,
<manager IP address>,
<manager port number>,
<user’s full name>
Ex) MANAGER,//luz.media.mit.edu/kwan.mngr,18.85.45.60,2343,Kwan Lee
```

provides the client with which Application Manager to connect to. The `<manager URL>` is the RMI URL of the Application Manager, which is not used for our default client implemented in C/C++. Java clients will be able to make use of this in order to make remote method invocations to the Application Manager. The Lookup Service obtains the user’s full name from the database after authenticating the user.

5.3.2. Client and Application Manager

The client then connects to the Application Manager that is specified by the message. It sends:

```
ONLINE,<userID>,<password>,<user’s full name>
```

to the Application Manager.

The Application Manager registers the client and obtains the RMI URL’s of the personal services of the user from the database. The personal services are the Speech Service, Presence Service, and the Profiler Service, which have been described previously. The Application Manager makes connection to these services and also acquires the list of applications that are online. It also lets the applications know that the user has come online. The Application Manager responds back to the client with the Speech Service Information

```
SPEECH,
<IP address of speech recognizer>,
<port number of speech recognizer>,
<IP address of text to speech>,
<port number of text to speech>
```

In addition, information about the applications that are online is sent to the client. The number of applications that are online is sent first.

```
NUMAPPS,<number of applications>
```

Then a `REGISTER` message is sent to the client for each application.

```
REGISTER,
<application ID>,
<application name>,
<application IP address>,
<application port>,
<application alertURL>,
<application transport type>,
<application IO type>,
<application sampling rate>,
<application channels>,
<application speech requirement>
```

Audio connections to the applications are setup at this time (if a connection is required to the application) using the application information obtained from the applications. The client creates an Application object for each application information that it receives from the Application Manager. The Application objects are placed into a circular list, which is browsed serially through button presses. Each application name is added to the speech recognizer vocabulary to enable random access to applications through voice activation.

5.3.3. Applications and the Lookup Service

When an application comes online or starts up, it needs to register with the Lookup Service so that future users may be notified about the application's presence. The application registers with the Lookup Service by sending its application information (`AppInfo`) to the Lookup Service. For Java applications, the remote method `registerApp()` in the `ILookupService` interface is used. For C/C++ applications it sends a `REGISTER` message to the Lookup Service. The `REGISTER` message sent by the C/C++ applications is of the following format.

```
REGISTER,
<application ID>,
<application name>,
<application IP address>,
      <application port>,
      <application alert URL>,
      <application transport type>,
      <application IO type>,
      <application control port>,
      <application sampling rate>,
      <application number of channels>,
      <application speech requirement>,
      <optional: speech recognition vocabulary>
```

`<application control port>` is the application port where the Application Manager makes a TCP connection to send control messages to the application.

The Lookup Service responds with the information about the Application Managers that are handling the users that are online. A `MANAGER` message is sent for each Application Manager that is serving an active user. This information is used by the application to notify its presence to those users. C/C++ applications have a one to one mapping of the `IManager` object to the Application Managers to interface with each Application Manager.

5.3.4. M

The text messaging protocols used here apply only to C/C++ applications since Java applications use RMI to communicate with the Application Manager. When an application comes online, it registers with the Application Managers so that online users may be notified about the applications that have come online. The same `REGISTER` message that is sent to the Lookup Service is sent to the Application Manager. The Application Manager then sends a `REGISTER` message to the client as described above to notify that the application has come online and the client can establish a connection with the new application.

If the application is a C/C++ application, the Application Manager creates a `CApplication` object that abstracts the socket communication for controlling the application. `CApplication` object implements the `IApplication` interface just as the Java applications, which provides a common interface for the Application Manager to interact with the applications, regardless of the application's implementation.

If a client comes online after the application, the Application Manager creates the `CApplication` object after acquiring the application information from the Lookup Service. However, it waits to receive

```
      UDPPORT,<application ID>,<port number>
```

message from the application, if the application uses UDP for audio connections. This updates the UDP port number in the `AppInfo` object in the Application Manager so that the client may connect to that port. This is required since UDP audio applications bind to a new UDP port for each user.

```
      GETINFO,<application ID>
```

message is sent by the application in order to obtain the user information from the Application Manager. The Application Manager responds with a `GETINFO` message.

```
      GETINFO,
      <user ID>,
      <user name>,
      <IP address of user client>,
      <UDP port number>,
      <TCP port number>,
      <alert URL>,
      <user status>
```

5.3.5. Client and Application

All control messages from the client to the applications and vice versa are forwarded through the Application Manager. The Application Manager replaces `<application ID>` with `<user ID>` for all messages that are sent from the client to the application. All audio connections are made between two end points in order to reduce any delays that might be caused by forwarding it.

When the client browses through different applications, the applications are automatically activated. The client can go into the sleep application (a dummy application that does not do anything) in order to enter into an inactive state. Once an application is activated, the client sends

```
      ACTIVATE,<application ID>
```

message to the Application Manager. The Application Manager then sends

```
      ACTIVATE,<user ID>
```

to the application to notify that the user with `<user ID>` has activated the application. Once the application is active, all input from the user is considered to be application specific, except for the browse buttons. When user input is detected by the client, the client sends an

```
APP,<application ID>,<input>
```

message to the Application Manager. This is forwarded to the application in the following format:

```
APP,<user ID>,<input>
```

The application ID is replaced with the user ID by the Application Manager. The application interprets the `<input>`.

Applications may send application specific messages to the client. These messages are also forwarded by the Application Manager. The following are some application specific messages, which are interpreted by the client.

```
UI,<application ID>,<list of items separated by comma>
```

The `UI` message is used by the applications to ask the client to display any application specific information (`<list of items>`) on the client's screen.

```
ALERT,<application ID>,<audio URL>
```

This message is used by the applications to alert the user. The client queues the application when it receives this message requesting the user's attention. The user can explicitly activate it or ignore it. After the user ignores the application for more than 20 seconds, the Application Manager sends a `POP` message to notify the client to remove the application from the alert queue.

```
POP,<application ID>
```

When applications like the music player want to ask the client to play a longer piece of audio such as music, the application sends a `PLAY` message.

```
PLAY,<application ID>,<audio URL>
```

This message is different from the `ALERT` message because the client responds with an `APP` message when it is done playing the audio.

When an application would like the client to record audio streamed by the application, such as Super Phone requesting voicemail to be recorded, it sends

```
RECORD,<application ID>,<1 or 0>
```

The application ID is the application requesting recording and 1 is to start and 0 is to stop recording. When the Application Manager receives this message, it first checks if the Recorder application is available and sends

```
APP,<user ID>,<application ID>
```

to the recorder. This tells the Recorder to record audio for `<user ID>` and that the audio is originating from the `<application ID>`. The Recorder keeps track of the recording state, so no 1 or 0 needs to be transmitted. The audio is actually forwarded by the client to the Recorder application.

When the application would like to ask the client to make specific audio connections during the use of the application, it sends

```
START,
<application ID>,
<number of connections: 0, 1 or 2>[,
<transport type:UDP or TCP>,
<io type:duplex,send only or receive only>,
<IP address>,
<port number>,
(optionally: <IP address>,<port number>)]
```

If the number of connections is 0, then the parameters are ignored. A `START,<application ID>,0` message would be sent by applications that do not require the client to send or receive audio as soon as the application is activated, such as the Super Phone application. The client will start reading and writing to the default connection to the application when it receives this message. The optional IP address and port number is to indicate that the input and output channels require separate network connections.

```
STOP,<application ID>
```

is sent by the application in order to tell the client to stop sending and receiving audio.

When an application would like the client to garble audio, it sends a `GARBLE` message to the client.

```
GARBLE,<application ID>,<1 or 0>
```

1 is for activating garbling and 0 is for deactivating garbling.

Once the user starts browsing (after the user is done with the current application), the current application is automatically deactivated. The client sends

```
DEACTIVATE,<application ID>
```

message to the Application Manager. The Application Manager forwards the message to the application with the `<application ID>` replaced by `<user ID>`.

5.3.6. Application Manager and Personal Services

All communication between the Application Manager and the personal services (Speech Service, Profiler Service, and the Presence Service) is done through RMI since they are all implemented in Java. Please refer to online documentation[36] for the details of these methods.

Speech Service and Application Manager

Register and unregister methods, activate and deactivate methods, and vocabulary manipulation methods are available to the Application Manager.

Presence Service and Application Manager

Subscription related methods, querying for subscription methods and status update methods are available to the Application Manager.

Profiler Service and Application Manager

Logging methods on usage and prioritization query methods are available to the Application Manager.

5.4. Operation

5.4.1. Registration and Setup Process

When the client initially comes online, it obtains the IP address and port number of the global Lookup Service from an HTTP server such as http://www.media.mit.edu/~kwan/impromptu/lookup.xml. This is to allow flexibility in where the Lookup Service can be deployed. When the client connects to the Lookup Service and authenticates itself, the Lookup Service returns the IP address and port number of an available Application Manager that can serve the client. The client registers to this Application Manager, which signals the Application Manager to initialize and register with the user's Presence, Profiler, and Speech Services. The Speech Service establishes connections with the speech recognition engine and text to speech engine at this time. The Application Manager also obtains the list of subscribed applications available from the Lookup Service.

The Application Manager creates a `CApplication` object with the `AppInfo` object obtained from the Lookup Service if the application is a C/C++ application. Then `CApplication.connect()` is called, which establishes control connections with the applications. If the application requires a UDP audio connection, a new datagram port (`getNewDgrmPort()`) is obtained. The `CApplication` object explicitly receives a `UDPPORT` message from the application. The Application Manager then notifies the applications that the user has come online and shares the user information with the application (`registerUser()`). `CApplication's registerUser()` method does not do anything when the Application Manager notifies the applications about the user, since the applications obtain the `UserInfo` from the Application Manager immediately after the Application Manager connects. This is required since the application needs the user ID in order to maintain a map of the connections to the Application Managers.

During registration, the client also establishes audio connections with the applications that require static audio connections directly with the application itself. Some applications will require dynamic audio connections that require the client to setup connections dynamically during the use of the application. The connections may be to the application or to another entity. End points of a static connection remain unchanged during the use of the application while with dynamic connections the destination of where the audio is sent is changed during the usage. For those applications requiring dynamic audio connections such as the Super Phone application, audio connections are not setup during registration.

Therefore, the client would not know where to send or receive data from when initially the application is activated. However, as the user calls someone the application notifies the client where to send audio.

When an application comes online, it goes through a similar registration process. It first registers with the Lookup Service and acquires a list of subscribed users that are online. The Lookup Service returns Application Manager information for these users and the application uses it to establish control connections with the Application Managers. The application sends the application information (`AppInfo`) to the Application Manager during registration. If the application requires UDP audio connections, the application information contains the new UDP port number. When the client is notified that the application has come online, the client establishes any static audio connections that are required for the application. Once an Application Manager connects to the application, an Application Manager wrapper class (`IManager`) is instantiated by the application to communicate with the Application Manager without worrying about the lower level sockets. Each `IManager` on the application side communicates with a `CApplication` object on the Application Manager side.

The Application Managers register with the Lookup Service when they are instantiated. These Application Managers are made available to clients that come online. The Application Manager uses the client's user ID to obtain personal settings for the user from the database. Each user is identified by an e-mail like ID. Each application in IMPROMPTU is also identified by an e-mail like ID (e.g. radio@media.mit.edu).

5.4.2. Activation and Deactivation

Once the client is fully registered, the user is ready to access different applications. As the user browses through the list of applications, the applications are immediately activated. When an application is activated, applications may start sending or receiving audio immediately, or wait for some user input. If the application is capable of speech based interaction, the Application Manager requests the Speech Service to load the application specific vocabulary. The XML vocabulary is loaded by the Speech Service when the application first registers, since building a vocabulary from the XML file takes quite a lot of time. The built vocabulary is sent to the speech recognizer when the application is activated. When the user switches to another application, the Speech Service unloads the vocabulary of the deactivated application.

Applications usually maintain separate threads for each user or at least a separate data structure that keeps track of the state of the user's application usage. If an application is a UDP application that sends audio to the client, it sends the audio to the user's datagram port. The client binds to only one datagram port (6040 by default) and receives all datagrams through this port. The application receives audio from the client at the port number specified by the application during registration. For TCP based applications, a stream channel is setup between all clients and all applications.

5.4.3. Going Offline

When a user goes offline, the Application Manager that used to serve the user becomes available to other users that come online. The Application Manager information is returned to the queue in the Lookup Service and the user information is removed from the Lookup Service. When an application goes offline, the Lookup Service removes the application information from its hash table.

Applications remove user specific resources (data structures and user handling threads) when the user unregisters from the application and goes offline. When applications go offline, the Application Manager cleans up the application specific resources and notifies the client to remove the application from its application list.

5.5. Summary

This chapter presented the details of IMPROMPTU's design and implementation. The system has been designed to support different types of applications and to operate in a distributed manner. Applications with different characteristics and different resource requirements have been abstracted on the IMPROMPTU client into similar application entities, to be accessible by the user in a coherent manner. In addition, new applications that conform to the IMPROMPTU architecture can be deployed easily to make them available to the users.

CHAPTER 6. EVALUATION

This chapter presents some evaluations on the user interface, the system architecture, and the system performance of IMPROMPTU. The system went through several iterations of design, implementation and testing. The users that tested the system and the user interface include my advisor, three graduate students in the Speech Interface Group and two undergraduate students who worked with me in the implementation. The evaluation presented here are lessons we learned as we informally tested the system during the different phases of the development. No formal user studies or usability testing have been performed. Current limitations of the system are presented along with some future improvements that could be made.

6.1. User Interface

6.1.1. Appropriate Alerting

Initially we had several random alerts for different events. Each application had alerts, but they were just random alerts and people got really confused because many different sounds were occurring, continuously overloading their auditory senses. The alert length also varied from short (0.5 seconds) to very long that ran for four seconds. After some feedback, we refined our alerts and made them around one second or less, and made their sound relate to the content of the application. The users seemed to have a better experience and less confusion when the alerts occurred.

Synchronizing alerts with different events that occur was sometimes difficult to do. Sometimes it takes a few seconds for a thread to unlock the play thread and that delays the play back of the waiting thread. Currently, applications may send `ALERT` messages to alert the user and play audio cues on the client. When playing audio cues, it is sometimes better to just stream the cues instead of sending an `ALERT` message to synchronize with the content being played back on the client. For example, when using the Recorder, an audio cue is played to inform the user that it has finished playing back a recording. Initially, an `ALERT` message was sent when the Recorder tried to indicate the end of a play back of a message. However, the streamed message did not finish playing back to the speakers when the client received the `ALERT` message. As a result, the alert was played, interrupting the play back of the message and the remaining message (about one second of audio) finished playing back after the alert. We fixed this by streaming the audio cue from the application after it finished streaming the recorded message. In this case, the audio cue was played back at the right time when the play back of the message finished. One solution to alleviate this problem is by storing the audio alert locally on the device. Since alert audio files are small, they could be cached locally for quicker playback when necessary.

6.1.2. Speech Interface

Using the speech interface for controlling the applications was frustrating when the speech recognizer performed poorly. There were two main problems that resulted in such low recognition rates (under 10%). The first problem occurred because the speech recognizer was not receiving all the data that the client was sending. The Java layer received audio from the client and forwarded the audio data to the native recognizer, but wrong number of bytes was getting forwarded making the recognition impossible. The second problem was caused by audio data with the wrong sampling rate. The speech recognizer accepts audio only at 22050 Hz, but for some applications, the audio device's sampling frequency is modified to accommodate different qualities. For example, the Music application requires 44100 Hz audio. When one tried to send a speech command while accessing the application, 44100 Hz audio was sent instead of 22050 Hz audio. This occurred because the audio device was not reset when the push-to-talk button was pressed. These problems were detected and resolved by recording and listening to the audio that was received by the speech recognizer. With these problems resolved, the speech recognition rate went up and eyes free navigation through the applications became possible.

Initially, the speech recognition was either always on or off depending on the application. If the application required speech recognition, all audio input from the microphone was sent to the speech recognizer. As a result, applications that required direct audio input from the user could not access the speech interfaces. "Push to talk" functionality was implemented in order to allow more control over when the audio got sent to the speech recognition instead of the application. However, initially the push to talk button toggled the recognition state with each button press without giving the users feedback on the current state of the speech recognition. Many users pressed the button too hard and the input was interpreted as pressing it multiple times. When using this toggle button, users got confused whether the speech recognition was listening to them or not. The visual output was the only indication on whether the recognition was on and whether there was anything being recognized.

This was completely redesigned into a push to talk button with audio feedback when a word was recognized. The button had to be held down to do recognition and released to stop recognition. The tactile and audio feedback gave a better sense about the state of the recognition. When the button is pressed down, audio is forwarded to the speech recognizer for recognition and when something is recognized, a short chime indicates positive recognition. If the users do not hear a chime in a second, they can conclude that the recognition failed. Additionally, the output audio from the client is silenced when the push to talk button is pressed to reduce any interference during recognition.

Speech recognition is very sensitive to the noise level of the surrounding environment, but unfortunately we cannot use an alternative microphone due to hardware limitations. Better noise canceling microphones can be used in the future to enhance speech recognition results. The speech recognition rate also depended heavily on the recording volume level and the position of the device relative to the mouth. Usually such adjustment had to be made intuitively.

Loading vocabulary from an XML file takes from five to ten seconds. As a result, applications that used vocabulary could not be accessed immediately when they were activated because the initial design loaded the vocabulary when the user first activated the application. This created significant delays to play the first information, such as in the News Headline application, since the activation call was synchronous and it involved loading the vocabulary. This was modified so that the vocabulary loads up when the application registers and the vocabulary loading was performed in a separate thread to eliminate any delays of client being notified, during application registration. Unless the user activates the application as soon as the application comes online, the delay problem is unnoticeable.

Synchronizing between events that occur on the client such as button press and text to speech being streamed was difficult. Once some text is fed into text to speech, the text to speech does not have mechanism to abort the synthesis. Consequently, while one is listening to a news headline and decides to skip to the next one, the user has to wait for the whole headline to be played before the next headline can be played back. Initially, text was streamed from the application every time it got a "continue" message from the text to speech engine. The "continue" message indicated that the text to speech finished processing the text input. However, since the speech output speed on the client is slower than the text to speech synthesis speed, when the user gave a command to the application, the command was processed in reference to several headlines ahead of when the user executed the command. As a result, the application had to incorporate some delays before sending text to the text-to-speech engine. The delay was relative to the number of characters of the output text that was sent to the text to speech engine. This resulted in a more adequate response to the user commands and avoided overflow of text-to-speech output on the client.

6.1.3. Enhanced Telephony

Initial impressions regarding Garble Phone(Section 4.2.9) were skeptical among visitors and sponsors who were concerned with privacy issues. But after it was understood that such feature would be used among trusted groups of people and after hearing the quality of the garbled audio, it was better accepted. Originally we used a simple algorithm that just swapped the audio packets as they were captured from the microphone, but this was unacceptable since the resulting audio content was still understandable. Consequently, we tried garbling audio with four segments of buffering and swapping each segment read from the microphone with a random one among the four segments. Later we compared the garbled audio with six segments of buffering, and we adopted this because it sounded less intelligible while providing similar state information about the receiver.

Some users also noted that the MONITORING state in the Super Phone is too explicit and transitioning the call to voice mail at the receiver's request from MONITORING state was socially awkward, since it resembles someone closing their doors on a visitor's face. It was suggested that the transition to MONITORING state be made less explicit so that the caller does not notice that he/she is being monitored, or make the transition to VMAIL state, upon receiver's request, more kindly so that the caller is informed with a message indicating that the receiver is really busy and cannot take the call. Another option is to make the MONITORING state available only to trusted callers. A long term user testing would be necessary to evaluate the performance of garbling and the usability of the Super Phone.

6.1.4. Modes of Control

The serial browsing is scalable in the sense that new applications do not change how the user browses through the applications, although it will require more browse time on the average to reach the desired application. The speech interface makes random access navigation scalable, although the applications should be named well in order for effective speech recognition. We had applications such as News and Music that would confuse the speech recognizer. Furthermore, when dynamically loaded vocabulary words were not part of normal English(e.g. proper nouns), the speech recognizer performed very poorly.

In order to solve this problem, a vocabulary pool of user names was created. With the existence of the vocabulary pool, the recognizer could recognize the user names pretty well. This was a built in functionality of the speech recognizer that we initially did not know about.

Initially, navigating through the applications was the only way of knowing which application one was active and what were the online applications. As a result, two speech commands "Where am I?" and "What are the applications?" were added to provide better overview of the status of the applications. As the number of the applications increases, the user will have more difficulty keeping track of available applications. During the testing stage, there were several occasions when users tried to access unavailable applications. The user initially thought the speech recognition was having problems, but actually the application was offline. This is because users would know whether something has come online or gone offline with distinct alerts, but they would not know which application actually came online or went offline. This was modified so that the application specific alerts were incorporated into "coming online" and "going offline" alerts to give the users a better sense of the changing status of the applications.

There are only a limited number of buttons available on the iPaq to be used for application specific commands: UP, DOWN and FUNCTION. On the other hand, the speech interface was much more scalable and new commands could be added very easily. As a result, these buttons were used to complement the speech interface to help navigation and implement specific operations for the applications. The buttons can especially be used to access the applications when speech interface is not effective due to the noise in the environment.

6.1.5. Audio Processing

[37] made a comparison of interactions between using voice over IP and using normal phones for customer service. The study found that voice delay affected users' perceptions of the ease of speech interaction. On the average, it took 45% more time to complete customer service tasks. Currently, in IMPROMPTU, the audio delays in applications such as the Super Phone make the user experience less adequate than a normal phone although it uses higher quality audio and provides more features. Initially the delays were in the range of 500 msec to about a second. Network traffic also introduces intermittent breaks in audio. However, we discovered there was buffering on the sound device of the client that was causing the delay. With buffer adjustments (decreased buffer size) the audio delay between iPaq to iPaq

full duplex voice communication was brought down to about 200 msec. In the current IMPROMPTU client, we are using one buffer size for all the applications and the buffer settings should be modified for different applications in order to adjust better for their communication requirements. Sometimes we also experience an asymmetric delay between communication entities as in Baby Monitor. The delayed audio is less of a problem for the monitoring application than a phone application, and in order to make real time communication acceptable on IMPROMPTU, delay problem needs to be investigated further and resolved.

Currently, there is no encoding done on the audio sent over the network since the network we used supports up to 11 Mbps and normally 2 Mbps of wireless bandwidth. IMPROMPTU seemed more limited by computational power than bandwidth. Initially, when the MP3's were decoded on the client, there were a lot of audio breaks. When just an MP3 player was run on the client, the player ran without trouble. However, in the IMPROMPTU client there are other threads that are running in the background and that seemed to interrupt the play back of streamed audio. When the same IMPROMPTU client code was compiled and run on a Pentium III 850 MHz PC with 512 MB of memory, it ran perfectly. However, when it was run on a Pentium Pro 200 MHz PC with 128 MB of memory, we observed the same behavior as on the iPaq with 32 MB of memory. As a result, we have concluded that the problem was more dependent on the current iPaq's processing power than bandwidth or memory. In the current state of the real world, it is more reasonable to assume that there will be more computation available on the devices before more wireless bandwidth would be available and in this case, encoding would be necessary to save wireless bandwidth.

These problems all relate to buffering problem on the IMPROMPTU client. As a result, I have concluded that a more dynamic buffering scheme needs to be implemented on the client to adjust to each application's audio requirements.

6.2. Architecture

6.2.1. System Design

The current IMPROMPTU architecture is able to support multiple audio applications in a seamless way. We have implemented eight applications with different characteristics. They are all supported through the same system components and messaging protocol. Since the applications are characterized by their properties, these properties could be modified or additional properties could be added if necessary to support different types of applications.

The distributed architecture allows flexibility in deploying the system. Each component can be deployed anywhere as long as it knows where the Lookup Service is. The personal services require modification of user settings in the database in order to deploy them elsewhere, since these settings are used by the Application Managers to locate the services after user authentication.

6.2.2. Extensibility and Scalability

IMPROMPTU can support different types of applications and practically any number of applications due to its support for distributed applications. The application properties could be extended to support new types of applications and the messaging protocol can be extended to support new types of messages.

There are several issues in terms of scalability: the number of applications that the system can support, the number of users that the system can support and the number of users the applications can support. In supporting multiple applications, the main bottleneck would be the Application Manager and the client's memory. Although the client is made as thin as possible, it needs to keep track of the audio connections to each application and few other states that are needed.

The system has also been designed to support multiple users as long as there are computing resources available to deploy the components necessary to serve each user. Every component can be run on a different host. However, personal services are not shared between users. Separate speech recognition and text to speech components are needed for each user. All the components are distributed so each user can deploy their own services independently from each other. However, the Lookup Service becomes a bottleneck since it is a centralized service, and the scalability of the system would be limited by the physical memory available on the host machine of the Lookup Service. This could be resolved by making the Lookup Service a distributed system or by using a peer-to-peer computing model.

Each application also currently supports multiple users through multithreading. However, there is a limit to multithreading. Possibly, the same application may be deployed with a different ID's and allow different users to subscribe to each other. For a commercial deployment, an automatic load balancing mechanism would be a necessary to support multiple users.

6.2.3. Shared Resources

The applications compete to access the microphone and the speakers of the client device. There are also multiple threads that handle messages from the applications and user inputs that require access to the audio resources. This required locking critical sections in the program, which also introduced deadlock problems. Many of these problems have been resolved currently, but it still remains to be seen where the client may hang.

6.2.4. Software Reliability

A major challenge in the development of IMPROMPTU was debugging the distributed system because the point of failure could be anywhere. The complexity was multiplied due to components that were multithreaded. Currently, IMPROMPTU supports multiple applications pretty well, although it locks up once in a while. The lock up usually occurs due to some blocking request in the network. Timeouts and selects are used to overcome this and to prevent blocking of reads and writes to sockets.

Recovering from failures is an issue that has not really been dealt well during the development of IMPROMPTU due to time constraints. The Lookup Service for example is currently the single point of failure, and it requires restarting of the whole system when it crashes. A graceful recovery mechanism is needed so that the system does not have to be brought down completely when one component fails. This would require maintaining consistent states in the system when a component fails.

CHAPTER 7. CONCLUSION

7.1. Contributions

IMPROMPTU shows the feasibility and desirability of one device that can handle various audio applications. The result is a multi-functional audio appliance that seamlessly gives access to networked audio applications. IMPROMPTU architecture developed in this thesis provides guidelines for managing multiple applications on a mobile device that supports sufficient wireless bandwidth for streaming audio over the IP network. The auditory user interface provides means to distinguish between different applications and to manage the state of current interaction with different applications. Distinct audio cues for each application allow a user to distinguish between different applications as they are navigating through the applications. When applications are in the background, they may alert the user for attention and a one-touch interface is available to the user to bring these alerting applications to the foreground. The design criteria for the user interface required making access to audio information and communication as simple as possible for the users. Two ways of selecting applications are provided. One is through buttons where one can serially browse through the list of applications. The second method is by using voice activation for random access to the applications.

Eight different applications were implemented to prove the feasibility of the architecture. News Headline application uses speech recognition and text to speech to deliver updated news headlines. Baby Monitor is an awareness application that allows parents to monitor their baby's that are out of their sight. Radio application streams real time radio to the users. Personal MP3 music collection can be enjoyed through the Music Player. Audio Book allows users to listen to different audio books with the capability to bookmark each book. Recorder allows users to record personal memos or record audio from different applications. Chat is an asynchronous multi-user application that makes chat history available to the users. Finally, the Super Phone implements new ways of setting up calls by providing Garble Phone and call screening functionality that gives the users more flexibility during call setup negotiation.

IMPROMPTU client is designed to be an easy to use appliance to access all these applications through speech and tactile interfaces. Through iterative design and testing, the user interfaces such as the push-to-talk button, Application Manager and application level speech commands and auditory cues were tuned to make the appliance more usable. Although, several iterations of testing and debugging have made the system pretty usable, it is still not reliable enough to be used in the real world environment. However, lessons learned from the implementation of IMPROMPTU can provide guidelines for designing user interfaces and an extensible architecture and protocols for IP based mobile devices that will in the near future support multiple audio based applications.

7.2. Future Work

The following is a list of future works that would make IMPROMPTU client a more exciting and entertaining appliance to have.

- As there is no need for large screen space with IMPROMPTU, the actual device could be reduced to a much smaller size as long as it supports wireless IP connection and full duplex audio. Currently, the size and weight are constrained by the wireless LAN interface and the battery. However, since the device does not have to support such a nice screen as the one on the iPaq, the battery size could be reduced and the battery life improved.
- Current system is stable enough to run some short-term formal user evaluations of the user interface. Such an evaluation would provide valuable feedback on what should be redesigned or improved.
- Reliability of the system needs to be improved so that long term user testing could be performed. Daily usage of IMPROMPTU and its social impact would be a very interesting research issue. This will also provide what interactions really work and what really do not work.
- Choosing appropriate alerting is difficult especially with increasing number of applications. However, Mynatt's research[38] indicates that users want to choose their own alerts. So, we have allowed users to customize application alerts through the web interface. It would be interesting to observe what kind of alerts get chosen by different people. Initially, during development, we were not too annoyed with long alerts and some interesting alerts were used for entertainment.

- Dynamic prioritization that changes the priority of the applications with the time of the day or the location of usage would be interesting. For example, the News application might take priority during commute hours in the mornings and the evenings while it becomes less important during the work hours and at night.
- Different audio applications would require smart caching capabilities to make some specific contents available offline. Determining criteria for what to store locally and what to update periodically to conserve bandwidth is an interesting question. Sean Wheeler in the Speech Interface Group is investigating these issues.
- Capability to mix audio efficiently would allow more interesting interactions. Chat application could implement real time mixing for synchronous chat. Audio on client could be presented more efficiently by mixing audio and presenting it simultaneously. Currently everything is played sequentially and requires interruption of current play back when an alerting occurs.
- Audio encoding would be beneficial especially for varying network connectivity where high bandwidth is not always available. However, as seen from the client side MP3 decoding experiment, it might require more computational power to handle audio encoding well.
- Several users have suggested that a visual interface would greatly complement the current interfaces. Visual pop up windows would indicate more clearly what is happening. Currently online applications could be more easily accessible through a visual interface rather than asking for it and listening to the list of applications. Visual interfaces would enforce the audio cues and help indicate different status of the applications.
- Further user testing can reveal when Garble Phone and call screening functionality really work well and where it really does not work well. Currently, we believe that it works well among people with pretty close and intimate relationships.
- Implementing time scaling capabilities on the applications would allow more flexibility when users need to browse through audio content, such as when they have to browse through chat history. A related issue involves efficiently summarizing an audio content.

- Synchronizing visual interfaces with sound interfaces was an issue that was dealt with during the development of Desktop Audio. If visual interfaces are developed for IMPROMPTU applications, similar issues will have to be resolved.
- The distributed architecture of the Speech Service implies that speech engines from other languages could be utilized to provide multilingual capability to IMPROMPTU.
- Finally, Norman suggested that "free sharing of information is critical to the appliance vision." Currently, audio is shared as a medium of delivery, but the data is not easily shared. The closest to this kind of interaction occurs during recording an application's content with the Recorder application. The NFS or the HTTP server serves as an infrastructure that makes archived audio accessible to other remote applications. A better sharing of data would allow me to share my music with a friend while I converse over the Super Phone.

Most appliances we use today are passive. We use it when we need it. If we do not use a microwave, it will not alert me for attention to use it unless an alarm is set. It just sits there waiting to be used. However, communication appliances are always alerting and always making noise. Especially with the emergence of the Internet, communication and information has become much more flexible and dynamic. The world of information and communication changes more rapidly than what normal humans can keep up with. In the telecom world, as mobility becomes more commonplace, people need to carry around different devices (pagers, mobile phones, palms), for different types and levels of communication. These devices independently try to get user's attention and these devices do not share the same channel of information, making certain data such as e-mail inaccessible when you do not have your two-way pager.

IMPROMPTU is an architecture that integrates these different communication channels and instead of having to attend to several devices, it allows the users to access numerous applications from one device. One device handles all their audio communication, information access and entertainment. Through a simple auditory interface, it also makes it very easy to attend to these applications when they ask for user's attention. Audio has been a ubiquitous medium for communication and information access and as computing becomes more ubiquitous along with the Internet Protocol becoming the standard protocol for information exchange and communication, IMPROMPTU will allow users to more easily manage and access what they want, whenever they want, while constantly and conveniently being connected with their loved ones not through text but through a richer medium.

CHAPTER 8. APPENDICES

8.1. Software Setup

When running Java based services, the `CLASSPATH` environment variable need to be setup correctly. Also JDK 1.3 or above must be installed with the `PATH` environment variable pointing to `/usr/java/jdk1.3/bin` which contains JDK 1.3 executables (`rmiregistry, java`) required to run Java applications.

The following libraries are required to compile Java components.

- Tritonus Java Sound (http://www.tritonus.org) - is an open source implementation of Java Sound that allows access to sound resources from Java applications.
- JDOM (http://www.jdom.org) and Xerces parser (http://xml.apache.org) are Java based XML parsers and API's that are used to parse vocabularies and setup information.
- Ant is a build tool like "make" that is used to build Java applications.
- RmiJdbc (rmijdbc.jar) is the remote interface for accessing the database from remote Java components.
- InstantDB libraries (idb.jar) are Instant DB related Java classes.

The following libraries are needed for C/C++ development.

- XML 2 library is used to parse XML documents from the applications.
- Pthread library is used to support multi-threading from the applications.
- AudioFile library is used to read and write WAV files on the disk.
- Splay library is used to play streaming WAV and MP3 files.

8.2. Notes on Development

The services are all implemented in Java, and Remote Method Invocation (RMI) is used to communicate among the distributed services. The Application Manager is also implemented in Java and it follows the application architecture for RMI. The stubs (components that translate remote method calls to lower level socket calls and vice versa) are located at an HTTP server in order for the components to get access to the stubs of each distributed components when they make remote calls. These stubs are placed into http://www.media.mit.edu/~kwan/classes when they are built.

All the source files are currently maintained in the CVS (Concurrent Versions System) repository hosted on ventolin.media.mit.edu. The files can be viewable through the web browser at http://ventolin.media.mit.edu. Ant is used to build the Java components and the C/C++ components are built with the make utility.

8.3. Audio Formats

IMPROMPTU uses 22050 Hz, 16 bit, mono audio by default for most audio communication. For WAV files, the files need to have audio data that begin at 44 (2c hex) offset. Other offset values will not work. One can convert the audio on Linux using `sox`. The command to convert the audio to 16000 Hz 16 bit is

```
sox <input file> -r 16000 -w <output file>
```

One can check the audio format using the `sfinfo` program. `sfinfo <wav file>` prints out the following information for a WAV file.

```
File Name      yo.wav
File Format    MS RIFF WAVE Format (wave)
Data Format    16-bit integer (2's complement, little endian)
Audio Data     57344 bytes begins at offset 44 (2c hex)
               1 channel, 28672 frames
Sampling Rate  16000.00 Hz
Duration       1.79 seconds
```

CHAPTER 9. REFERENCES

[1] D. Searls, "PocketLinux Gives Jabber Its First Hand(held)," *Linux Journal*, 2001, http://www.acm.org/pubs/articles/journals/linux/2001-2001-82es/a12-searls/a12-searls.html.

[2] "Mobile Phone Usage Up in 2000," 2000, http://www.advisor.com/Articles.nsf/aidp/OLSEE108.

[3] W. Forum. "WAP Forum," 2000, http://www.wapforum.org/.

[4] W. W. W. Consortium. ""Voice Browser" Activity," 2000, http://www.w3.org/Voice/.

[5] F. Rose, "Rocket Monster: How DoCoMo's wireless Internet service went from fad to phenom - and turned Japan into the first post-PC nation," in *Wired*, vol. 9, 2001, pp. 126~135.

[6] D. A. Norman, *The Invisible Computer*. Cambridge: MIT Press, 1998.

[7] D. Hindus, M. S. Ackerman, S. Mainwaring, and B. Starr, "Thunderwire: a field study of an audio-only media space," in the proceedings of CSCW, Boston, MA, 1996, pp. 238-247.

[8] E. R. Pedersen and T. Sokoler, "AROMA: Abstract Representation of Presence Supporting Mutual Awareness," in the proceedings of ACM CHI, Atlanta, GA, 1997, pp. 51~58.

[9] B. Raman, H. J. Wang, J. S. Shih, A. D. Joseph, and R. H. Katz, "The Iceberg Project: Defining the IP and Telecom Intersection," *IT Professional*, pp. 22~29, 1999, http://iceberg.cs.berkeley.edu/publications.html.

[10] C. Schmandt, S. Angebranndt, R. L. Hyde, d. H. Luong, and N. Siravara, "Integrating Audio and Telephony in a Distributed Workstation Environment," in the proceedings of USENIX, Nashville, Tennessee, 1991, pp. 419~435.

[11] I. T. Union, "IP Telephony Workshop," International Telecommunication Union IPTEL/03, May 29 2000, http://www.itu.int/osg/sec/spu/ni/iptel/workshop/iptel.doc.

[12] P. E. Jones. "H.323 Information Site," 2000, http://www.packetizer.com/iptel/h323/.

[13] H. Schulzrinne and J. Rosenberg, "Internet Telephony: Architecture and Protocols -- an IETF perspective," in *Computer Networks*, vol. 31, 1999, pp. 237~255.

[14] W. E. Witowsky. "IP Telephone Design and Implementation Issues," 1998, http://www.telogy.com/promo/white_papers/IP_phone/index.html.

[15] R. Bennett and J. Rosenberg, "Integrating Presence with Multi-Media Communications," dynamicsoft, Inc. 2000, http://www.dynamicsoft.com/resources/whitepapers.html.

[16] M. Day, S. Aggarwal, G. Mohr, and J. Vincent. "Instant Messaging/Presence Protocol Requirements," 2000, http://www.ietf.org/rfc/rfc2779.txt.

[17] N. DoCoMo. "All About i-mode," 2000, http://www.nttdocomo.com/i/index.html.

[18] TellMe. http://www.tellme.com.

[19] B. A. Nardi, S. Whittaker, and E. Bradner, "Interaction and Outeraction: Instant Messaging in Action," in the proceedings of CSCW ACM, Philadelphia, PA, 2000, pp. 79~88.

[20] H. J. Wang, B. Raman, C.-N. Chuah, R. Biswas, R. Gummadi, B. Hohlt, X. Hong, E. Kiciman, Z. Mao, J. S. Shih, L. Subramanian, B. Y. Zhao, A. D. Joseph, and R. H. Katz, "ICEBERG: An Internet-core Network Architecture for Integrated Communications," in *IEEE Personal Communications (2000): Special Issue on IP-based Mobile Telecommunication Networks*, vol. 7, 2000, pp. 10~19.

[21] M. L. Dertouzos, "The Oxygen Project : The Future of Computing," in *Scientific American*, 1999.

[22] R. W. DeVaul and S. Pentland, "The Ektara Architecture: The Right Framework for Context-Aware Wearable and Ubiquitous Computing Applications," The Media Laboratory, Massachusetts Institute of Technology 2000, http://www.media.mit.edu/~rich/DPiswc00.pdf.

[23] M. Weiser, "Some Computer Science Problems in Ubiquitous Computing," in *Communications of the ACM*, vol. 36, 1993, pp. 75~84.

[24] N. Sawhney, "Contextual Awareness, Messaging and Communication in Nomadic Audio Environments," M.S. Thesis, Massachusetts Institute of Technology (1998).

[25] D. K. Roy and C. Schmandt, "NewsComm: a hand-held interface for interactive access to structured audio," in the proceedings of Human Factors in Computing Systems, Vancouver, Canada, 1996, pp. 173~180.

[26] S. Marti, N. Sawhney, and C. Schmandt. "GarblePhone : Auditory Lurking," 1998, http://www.media.mit.edu/speech/projects/garblephone.html.

[27] N. Sawhney and C. Schmandt, "Nomadic Radio: Scaleable and Contextual Notification for Wearable Audio Messaging," in the proceedings of ACM SIGCHI, Pittsburgh, Pennsylvania, 1999, pp. 96~103.

[28] M. Hedin. "Splay (A software that plays MP3 on Linux)," 2001, ftp://ftp.arm.linux.org.uk/pub/armlinux/people/nico/old/splay-0.8.2-fp1.tgz.

[29] A. Guy, C. Worth, and K. Causey. "the Familiar Project," 2001, http://familiar.handhelds.org/.

[30] J. G. Dorsey, "ARM Linux on the Intel Assabet Development Board," Department of Electrical Engineering and Computer Engineering, Carnegie Mellon University, Pittsburgh, ICES 0x-yy-00, September 11 2000, http://www.cs.cmu.edu/~wearable/software/assabet.html.

[31] P. J. Rankin, "Context-Aware Mobile Phones: The difference between pull and push, Restoring the importance of place," in the proceedings of HCI International, New Orleans, LA, 2001.

[32] H.-W. Gellersen, M. Beigl, and A. Schmidt, "Sensor-based Context-Awareness for Situated Computing," in the proceedings of International Conference on Software Engineering, Limerick, Ireland, 2000, http://www.teco.edu/~albrecht/publication/sewpc00/sensor-based-context.pdf.

[33] M. Day, J. Rosenberg, and H. Sugano. "A Model for Presence and Instant Messaging," 2000, http://www.ietf.org/rfc/rfc2778.txt.

[34] J. Rosenberg, D. Willis, R. Sparks, B. Campbell, H. Schulzrinne, J. Lennox, B. Aboba, C. Huitema, D. Gurle, and D. Oran. "SIP Extensions for Presence," 2000, http://www.ietf.org/internet-drafts/draft-rosenberg-impp-presence-00.txt.

[35] J. S. Kim, "Impromptu Speech Service:Distributed Speech Recognition and Text to Speech Servers," Massachusetts Institute of Technology, Cambridge, AUP Paper 2001, http://www.media.mit.edu/~jangkim/Research/AUP.pdf.

[36] K. H. Lee. "Impromptu Online API Documentation," 2001, http://impromptu.media.mit.edu.

[37] Q. Zhang, C. G. Wolf, S. Daijavad, and M. Touma, "Talking to Customers on the Web: A Comparison of Three Voice Alternatives," in the proceedings of CSCW ACM, Seattle, WA, 1998, pp. 109~117.

[38] E. D. Mynatt, M. Back, R. Want, M. Baer, and J. B. Ellis, "Designing Audio Aura," in the proceedings of ACM CHI, Los Angeles, CA, 1998, pp. 566~573.