Context Aware and Adaptive Mobile Audio

INMACULADA RANGEL VACAS

KTH Microelectronics
and Information Technology

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LCN 2005-06

Context Aware and Adaptive Mobile Audio

Inmaculada Rangel Vacas

9th March 2005

Masters of Science thesis performed at
Wireless Center, KTH
Stockholm, Sweden

Supervisor: Gerald Q. Maguire Jr.
Examiner: Gerald Q. Maguire Jr.

School of Information and Communication
Technology (ICT)
Royal Institute of Technology (KTH)
Stockholm, Sweden

Abstract

Today a large percentage of the population uses a handheld (either a mobile phone or a PDA) a laptop computer, or some other computing device. As this penetration increases, the user wants to take as great an advantage of these devices as possible. It is for that reason that communication is demanded almost everywhere. Simply having continuous access to the network is no longer sufficient thus context awareness and easy accessibility are becoming more and more relevant.

The idea of this masters thesis is to explore these ideas building on the prior work of Maria José Parajón Domínguez. The devices used for this study will be an *HP iPAQ h5550* and a laptop. A client-server application, whose components will be explained in detail in further sections, was designed to study some factors that may be taken into account when trying to satisfy the users´ demands as stated above. One of these factors could be, for example, what are the effects of having a personal voice interface on the traffic to and from the user's mobile device. The aim of this voice interface will be to provide more freedom to the user and also satisfy the demand for greater accessibility and facilitate mobile usage, not only for the common user, but also for handicapped people. Regarding the user's desire to always have connectivity everywhere, we wish to examine the effects on the traffic to and from the user's handheld, when exploiting significant local storage. Also related to the requirements on current devices to be always and everywhere connected and the huge amount of resources that this entails, it will be of interest to study the possibility of exchanging personalized CODECs (in the extreme case exchanging voice synthesis modules) and how this might affect traffic to and from the user's mobile device. This last method could potentially greatly reducing both the demands on the communication links and the cost of this connectivity.

With all these ideas in mind, this thesis aims to research an area that is nowadays continuously attracting new users and the goal is to find solutions to the demands that have resulted for these trends.

Sammanfattning

Användningen av portabla elektroniska apparater så som mobiltelefoner, handdatorer med mera är nu för tiden vida utbrett. Ju fler apparater som används desto större blir efterfrågan efter mobila tjänster för dessa. Som ett resultat ökar behovet av goda kommunikationslösningar, ofta mer komplexa än endast kontinuerlig dataåtkomst.

Syftet med detta examensarbete är att fortsätta att utforska de idéer som Maria José Parajón Domínguez presenterat. För att utföra detta kommer en HP iPAQ h5550 och en bärbar dator att användas. En klient-server applikation kommer att tas fram för att undersöka några faktorer som påverkar kommunikationslösningarna. Ett exempel på en sådan faktor skulle kunna vara effekten av att ha ett personligt röstgränssnitt för trafiken. Syftet med detta gränssnitt skulle vara att erbjuda användaren större frihet och flexibilitet för sitt mobilanvändande, oavsett om användaren lider av något handikapp eller ej. Försök kommer även att göras med att lagra mycket data lokalt på användarens apparat, detta i ett försök att minska datatrafiken då många apparater kräver ständig och intensiv datakommunikation. Det är även av intresse att studera möjligheten av utbyte av personliga algoritmer, så kallade CODEC, och hur dessa skulle kunna påverka datatrafiken till och från den portabla apparaten. Det genomgående syftet för alla dessa faktorer är att sänka belastningen på de kommunikationslänkar som utnyttjas.

Målet med denna studie är att undersöka några sätt att möta den ökade belastning på kommunikationssystemen som väntas om trenden för mobilt användande ökar.

Acknowledgements

First of all I would like to express my most sincere gratitude to my project advisor, Professor Gerald Q. Maguire Jr., for helping me when I needed, encouraging me when problems rose, answering all my doubts, and being always willing to transmit his positivism and share his knowledge.

I would like also to thank all my colleagues at the lab for maintaining such a good atmosphere that made work easier and more comfortable.

All my friends need to be mentioned here, those who were encouraging me from Spain and those who were here, thanks to all them for supporting me in bad moments and make me feel much better when it was necessary. I would like also to thank Staffan for offering me all his help from the moment I arrived in Sweden.

My family was always present during the development of this project, they have been one of the most important pillars to maintain me up and thanks to their encouragements is that I could continue when things went wrong. It is for that reason that I would like to thank deeply my father José, my mother María and my sister Eva.

And last but not least, I would like to thank with all my heart the support of my boyfriend Sergio, for encouraging me in the bad moments, making me feel that I was able to overcome them and for sharing with me all the good ones.

Table of contents

List of figures, tables and acronyms

Figures

Tables

Acronyms

API	Application Program(ming) Interface
ASCII	American Standard Code for Information Interchange
COM	Component Object Model
CPU	Central Processing Unit
DTW	Dynamic Time Warping
HMM	Hyden Markov Model
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
I/O	Input/Output
IEEE	Institute of Electrical & Electronics Engineers
IP	Internet Protocol
ISM	Industrial, Scientific and Medical (radio spectrum)
ISP	Internet Service Provider
ITU	International Telecommunication Union
JIT	Just In Time
LAN	Local Area Network
LCD	Liquid Crystal Display
MAC	Media Access Control
MIME	Multipurpose Internet Mail Extensions
MMC	MultiMedia Card
MSIL	Microsoft Intermediate Language
NAT	Network Address Translation
PC	Personal Computer
PCM	Pulse-Code Modulation
PDA	Personal Digital Assistant
PSTN	Public Switched Telephone Network
RAS	Registration/Admission/Status
ROM	Read Only Memory
RTCP	Real Time Control Protocol
RTP	Real Time Protocol
RTSP	Real Time Secure Protocol
SAPI	Speech API
SCTP	Simple Control Transport Protocol
SD	Secure Digital
SDIO	Secure Digital Input/Output
SDK	Software Development Kit
SIP	Session Initiation Protocol
TCP	Transport Control Protocol
TFT	Thin Film Transistor
TTS	Text To Speech
UDP	User Datagram Protocol

URL	Uniform Resource Locator
USB	Universal Serial Bus
VQ	Vector Quantization
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

1 Introduction

1.1 Overview of the Problem Area

The number of mobile devices that surround us is increasing day by day. Different kinds of mobile phones, personal digital assistants (PDAs) and many other handhelds are becoming increasingly essential to a new user profile. Users have started to become familiar and comfortable with these devices and associated services. The possibilities that this wide range of devices offers to the user spans from establishing a conversation everywhere to having internet connectivity and all the opportunities that this constant connectivity gives to the user.

All this seems to be wonderful, but users want even more. The advantages of being connected everywhere, has to be complemented with context awareness and improved accessibility.

Having multiple devices that don't care about the user's state has been studied by Maria José Parajón Domínguez in her masters thesis [1]. She focused on finding a solution to this problem by unifying all these devices in a single device [16]. The application that she developed to evaluate the performance of her system is explained in detail in section 2.1.

Given this earlier work and taking into account the new needs of users, new ideas have been developed to try to address to these new needs.

Context awareness, is useful because it provides a mechanism to be able to detect, in some way, the situation of the user so that applications can provide (subject to the resources available) the best possible services that may be of interest to the user at a specific moment and in a specific situation.

As far as accessibility, the advantage of a voice interface is obvious. With a graphical or textual interface, the user is forced to concentrate his/her attention uppon choosing from a menu or typing. For many tasks this is not really efficient, especially in a world where time is one of the most valuable things a user possesses. Using a voice interface, the user doesn't have to pay attention to the device or start typing a command, but can simply tell the device what he/she is interested in doing at the moment. This advantage leads to using a voice interface exploiting both speaker and speech recognition. The first helps provide a certain level of security and the second enables interpretion of the commands as dictated by the user.

Another aspect that should be taken into account is the desire of the user to have connectivity everywhere. Analyzing this, on one hand it might be really useful, but on the other hand it could be very expensive for the user, due to the resources that it will consume or that need to be reserved, even if they are not used. A solution for this problem is to take advantage of the local storage of the mobile device. By providing a sufficient local cache we could continue with an activity (such as listening to a song or message), even if the connection is lost. Another solution to try to reduce, in this case, the amount of bandwidth used, for example by exchanging personal CODECs (in the

extreme case exchanging voice synthesis modules). These potentially allow low bandwidth links to provide very high quality audio.
Having all these ideas in mind, the next step was to design and implement a prototype providing solutions to the needs stated above. A description of the platform for this study can be found in the next section.

1.2 Problem Specification

This masters thesis builds upon the previous work by María José Parajón Domínguez [1]. A description of the system and the application that she developed can be found in section 2.1.

In our case, the wearable device that we will utilize in our study is the *HP iPAQ h5550* [10]. A complete specification of this device is given in section 2.2.6. To complete our system, a laptop will be used. Both the wearable device and the laptop will be running Microsoft's Windows operating systems. In the case of the PDA, Microsoft® Windows® Pocket PC 2003 Premium, and in the case of the laptop, Microsoft's Windows XP.

The environment chosen for our development tasks is Microsoft's Visual Studio .NET 2003 using the .NET Framework for the laptop and Compact Framework for the iPAQ. A description of the features and usage of this framework can be found at section 2.2.7. When using this environment, the programming language used for the applications is C#. In some cases it was also necessary to use C++ and for those applications, the environment used was eMbedded Visual C++ 4.0.

What we want to do is to use our PDA as a mobile audio device providing it with context awareness and easy accessibility. Having this in mind we will study and compare the amount of network traffic that needs to be sent in two different situations, on one hand, we will study the amount of traffic that needs to be sent when streaming audio from the laptop computer to the PDA and on the other hand, we will obtain the amount of traffic that needs to be sent in the case that we download the audio first from the laptop computer to the PDA and then we play it locally. Another point that we are going to study is the effect of communication errors using the two different situations comented before. We would like to have a brief idea of the users opinion regarding these two different ways of using a mobile audio device. It is for that reason that we will perform a study to know their preferences regarding streaming audio or playing audio locally. The last point that we want to evaluate will be the study of the advantages and and disadvantages of having a voice interface, also from the users point of view.

A complete specification of the design of our system can be found at section 3. The scenario we will use to evaluate our solution is described in section 3.1.1. And the details of the context information we will utilize will be given in section 3.2.

2 Background

2.1 Previous and related work

2.1.1 Audio for Nomadic Users

The present master thesis extends the work of María José Parajón Domínguez [1]. The aim of her thesis was to solve the problems of having multiple wearable devices by introducing a new device, capable of combining all of them and offering an audio interface.

Smart Badge [16] was used as the wearable device and a laptop completed the system. An overview of the system she developed is shown in Figure 1.

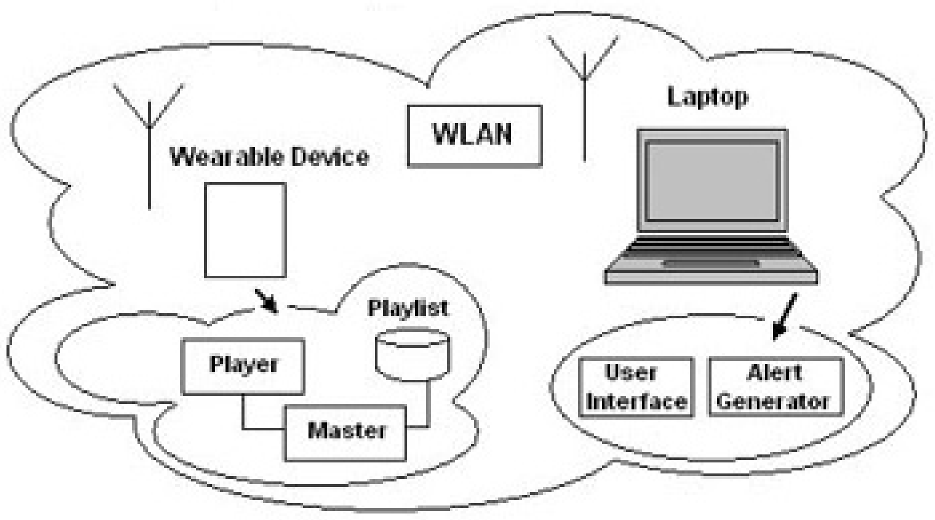


Figure 1: Overview of the system used

Both, the laptop and the wearable device, were running the Linux operating system. To test this environment, she developed a client-server application in the programming language C using the UDP protocol with the following components:

- *Master:* the server part of the application. It executes at the wearable device and its main function is to create and maintain a playlist by processing the requests from clients.
- *Player:* this client also executes at the wearable device. Its main function is to ask to the Master for the first element of the playlist and to invoke a suitable player to reproduce the content of this element.
- *User Interface:* this client is running at the laptop and its main function is to accept commands from the user and transmit them to the Master.

- *Alert Generator:* this client also executes on the laptop, accepts textual input and transforming it into audio alerts. For this María José Parajón used and modified a client developed earlier by Sean Wong [2].

More information about her thesis can be found at [1].

2.1.2 Pocket Streamer

David Evans' open source Pocket Streamer project [29] has been very useful to us, not only to compare two different ways of playing audio (in this case streaming vs. local storage). It also gave us new ideas.

Pocket Streamer consists of two parts: a server and a client. The server runs at the laptop and the client at the PDA. The requirements to run this application are:

- At the laptop:
    - Windows Media Player 9 [30] and SDK [32],
    - Windows Media Encoder 9 [31] and SDK [33], and
    - .NET Framework.

- At the PDA:
    - Windows Media Player 9,
    - Pocket Pc 2000, 2002 or 2003, and
    - .NET Compact Framework.

Information about Windows Media, can be found at [30], [31], [32], and [33].

To run this application, the first step is to start the server on the laptop, this application will appear as a system tray icon. The next step is to start the client at the PDA. In this client, the user is able to obtain the content at the Media Library in the laptop and select a track or playlist. When *play* is pressed at the PDA, the Windows Media Player [30] is started both at the laptop and at the PDA. As the audio is locally stored at the laptop, the output of the sound card is redirected to be the input of the Windows Media Encoder [31] which starts a broadcast session sending the audio to the PDA. From this moment on, a relationship between both players is established and the user is able to control the session from the PDA, e.g., going to the previous track, to the next one, and other controls typical of a normal media player.

An overview of this system can be found at the following figure. We see that the architecture is similar to the earlier thesis builds upon the previous one, [1], but in this case, we use a PDA as a wearable device.

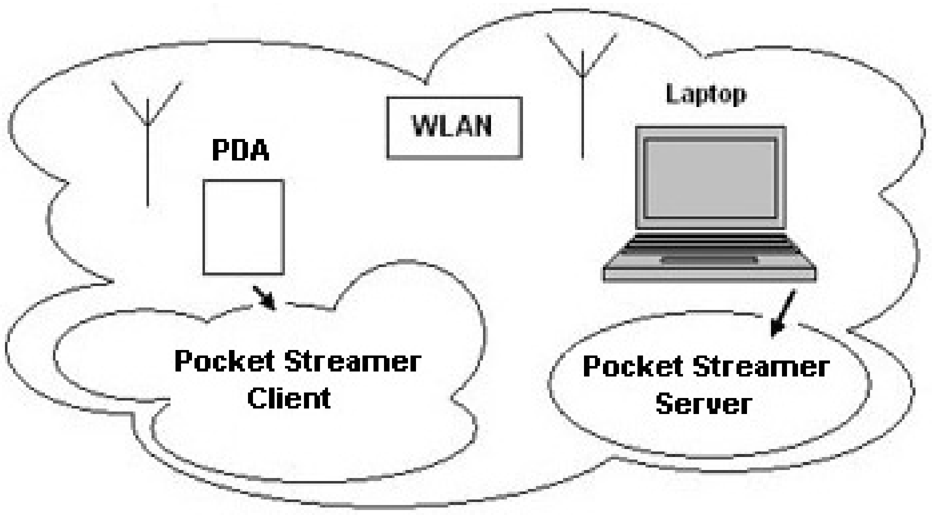


Figure 2: Overview of the Pocket Streamer schema

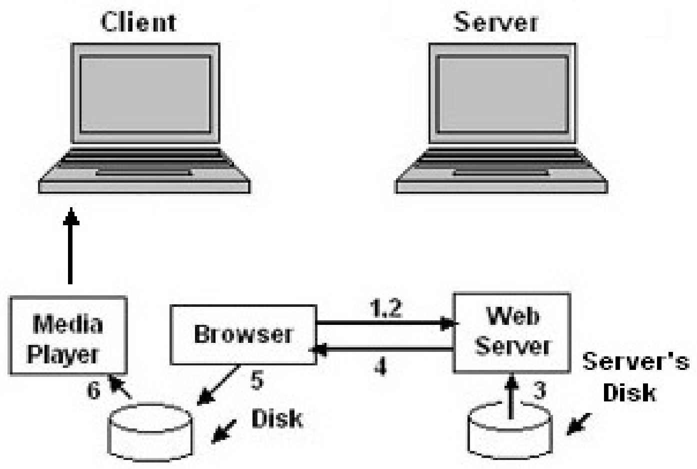
2.2 Useful concepts

Some basic concepts, useful for the reader (to understand the rest of the thesis) are introduced in this section.

2.2.1 Audio transmission

2.2.1.1 Streaming audio

The Internet has many web sites, many of which list song titles that users can click on to play the songs. This process is illustrated below:



1. Establish TCP connection.
2. Send HTTP GET request.
3. Server gets file from disk.
4. File sent back to browser.
5. Browser writes file to local disk.
6. Media Player fetches file block by block and plays it.

Figure 3: Process of obtaining an audio file from the server

The figure above shows the process that starts when the user clicks on a song. Their browser (step 1) establishes a TCP connection to the web server (i.e., where the song is hyperlinked). In step 2 the browser sends a HTTP GET request to request the song. Next (steps 3 and 4), the server fetches the song (which might encoded as MP3 or some other format) from the disk and sends it to the browser. If the file is larger than the server's memory, it may fetch and send the file in blocks.

Using a MIME type, for example, audio/mp3, (or the file extension), the browser determines how it is supposed to display the file. Normally, there will be a helper application such as RealOne Player [17], Microsoft's Windows Media Player [18], or Winamp [19], associated with this type of file. Since the usual way for the browser to communicate with a helper is to write the content to a scratch file, it will save the entire (music) file as a temporary file on the disk (step 5), then it will start the media player and pass it the name of the scratch file. In step 6, the media player fetches the content and plays the music, block by block.

In principle, this approach is completely correct and will play the selected music. The only trouble is that the *entire* song must be transmitted over the network *before* the music starts. If the song is 4 MB (a typical size for an MP3 song) and the transfer rate is 56 kbps, the user will wait for almost 10 minutes (in silence) while the song is being downloaded.

To avoid this problem without changing how the browser works, music sites have come up with the following scheme. The file linked to the song title is **not** the actual music file. Instead, it is what is called a *metafile*, a very short file that simply **names** the music. A typical metafile might be only one line of ASCII text, such as:

```
rtsp://eva-audio-server/song-0014.mp3
```

When the browser gets this 1-line file, it writes it to disk in a temporary file and starts the media player as a helper handing it the name of the temporary file, (as usual). The media player reads this file and sees that it contains a URL. The player then contacts the eva-audio-server and asks for the actual song to be streamed to it. Note that the browser is no longer involved.

In most cases, the server named in the metafile is **not** the same as the web server. In fact, it is generally not even an HTTP server, but rather it is a specialized *media* server. In this example, the media server uses the Real Time Streaming Protocol (RTSP), as indicated by the URL scheme name "rtsp".

The media player has four major taks:

- provide a user interface,
- handle transmission errors,
- decompress and decode the music,
- and eliminate (or at least try to hide) jitter.

As noted, the second job is dealing with errors. Real-time music transmission rarely uses TCP because if there where an error the resulting TCP based retransmission might introduce an unacceptably long delay, leading to a break in the music. Instead, the actual transmission is usually done using a protocol such as RTP [20]. Like most real-time protocols, RTP is layered on top of UDP, so that packets may be lost. However, it is up to the player to deal with these losses.

The media player's third job is decompressing and decoding the music. Although this task is computationally intensive, it is fairly straightforward.

The fourth job is to eliminate (or hide) jitter. All existing streaming audio systems start by buffering about 10–15 seconds worth of music before starting to play, thus they are able to hide a very large amount of jitter.

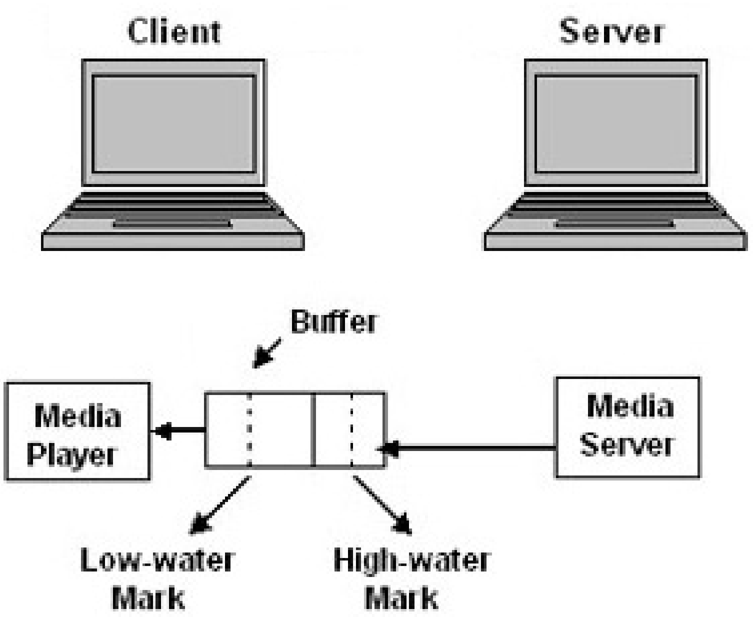


Figure 4: Transmission of an audio stream

Two approaches can be used to keep the buffer full. With a pull server, as long as there is room in the buffer for another block, the media player sends a request for an additional block to the server. Its goal is to keep the buffer as full as possible. The disadvantage of a pull server is all the *unnecessary* data requests. The server knows that it has to send the entire file, so why does the player need to keep asking? For this reason, this approach is rarely used.

With a push server, the media player sends a PLAY request to the server and the server continues to push data to it. There are two possibilities: the media server runs at normal playback speed or it runs faster. In both cases, some data is buffered before playback begins. If the server runs at normal playback speed, then the rate at which data arrives from the server should be the same rate that the player removes data from the front of the buffer (for playing). As long as everything works perfectly, the amount of data in the buffer remains constant in time. This scheme is simple because no control messages are required in either direction.

The other push scheme exploits the fact that the server can send data faster than it is needed (i.e., faster than the playout rate). The advantage is that if the server cannot execute at a constant rate, it has the opportunity to catch up if it ever gets behind. However a problem here, is that potentially the buffer can overflow if the server pumps out data faster than it is consumed (note that out of content it has to be able to do this to avoid the player encountering gaps, i.e., out of content).

The solution is for the media player to define a low-water mark and a high-water mark in the buffer. Basically, the server only sends data until the buffer is filled to the high-water mark, then the media player pauses. Since some data will continue to arrive before the server has gotten the pause request, the distance between the high-water mark and the end of the buffer has to be greater than the bandwidth-delay product of the network. After the server has stopped, the buffer will begin to empty. When the amount buffered reaches the low-water mark, the media player tells the media server to start transmitting again. The low-water mark has to be positioned so that a buffer underrun does not occur.

To operate a push server, the media player needs a remote control protocol. RTSP provides the necessary mechanism for the player to control the server. However, it does not specify the data stream, which is usually sent using RTP. The main commands provided by RTSP are:

Command	***Server action***
DESCRIBE	List media parameters
SETUP	Establish a logical channel between the player and the server
PLAY	Start sending data to the client
RECORD	Start accepting data from the client
PAUSE	Temporarily stop sending data
TEARDOWN	Release the logical channel

Table 1: Commands from the player to the server

2.2.1.2 Internet radio

There are two general approaches to “Internet radio”, the provision of a service similar to a radio broadcast. In the first, the programs are prerecorded and stored on disk. Listeners can connect to the radio station's archives and pull up any program and download it for listening. In fact, this is exactly what is done with the streaming audio we just discussed. It is also possible to store each program just after it is broadcast live, so that the archive is only perhaps, half an hour, or less behind the live feed. The advantages of this approach are that it is easy to do, all the techniques we have discussed also work here, and listeners can pick and choose from among **all** the programs in the archive.

The other approach is to broadcast in real-time over the Internet. Some of the techniques that are applicable to streaming audio are also applicable to live Internet radio, but there are some key differences.

One key difference is that streaming audio can be pushed out at a rate greater than the playback rate; since the receiver can stop the server when the high-water mark is hit. Potentially, this gives the server time to retransmit lost packets, although this strategy is not commonly used. In contrast, live radio content is always broadcast at exactly the rate that it is originated and played. Another difference is that a live radio station usually has hundreds of thousands of simultaneous listeners whereas streaming audio is a client-server application. Given these differences, Internet radio uses multicasting (when it can) with the RTP/RTSP protocols. This is clearly the most efficient way to operate such a service.

Unfortunately, current, Internet radio does not work this way. What generally happens is that the user establishes a TCP connection to the station and the content (feed) is sent over a TCP connection. This is because most internet providers (ISPs), do **not** support multicast and in addition most firewalls (and NATs) do not support multicast. Hence the use of one-to-one transmission via lots of TCP connections.

2.2.1.3 Voice over IP

Initially, public switched telephony systems, were used for carrying voice. Some years later, the movement of data bits over this system started increasing. Today, far more bits of data are carried than voice calls. Together with the cost advantages of packet-switching networks, today even traditional network operators are very interested in carrying voice over their data networks.

2.2.1.3.1 H.323

One thing that was clear to everyone from the start was that if each vendor designed its own protocol stack, interconnected systems would never work. In 1996, ITU issued recommendation H.323 entitled *"Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-Guaranteed Quality of Service"* [21]. The recommendation was revised in 1998, and this revised H.323 was the basis for the first widespread Internet telephony systems.

H.323 is more of an architectural overview of Internet telephony than a specific protocol. It references a large number of specific protocols for speech coding, call setup, signaling, data transport, and other areas rather than specifying these things itself. The general model is depicted in Figure 5. At the center is a gateway that connects the Internet to the traditional public switch telephone network (PSTN). The H.323 protocols are used on the Internet side and the PSTN protocols on the telephone side of the gateway. The communicating devices are called terminals. A LAN may also have a gatekeeper, which controls the end points under its jurisdiction, called a zone.

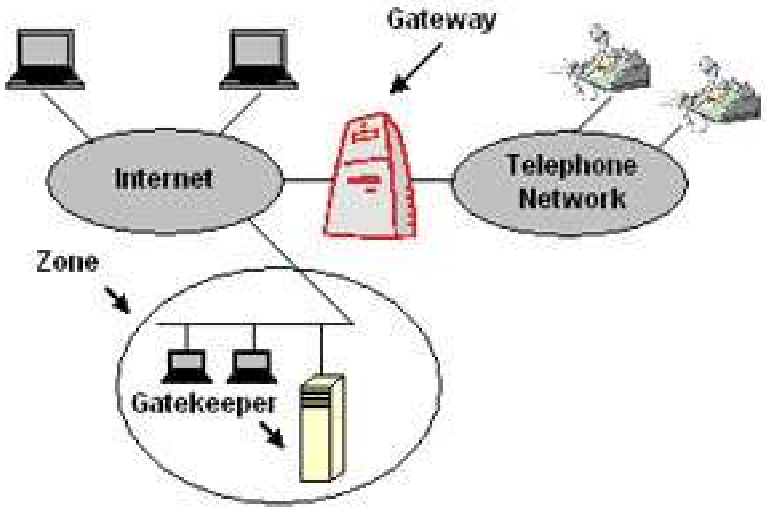


Figure 5: The H.323 architectural model for internet telephony

A telephony network utilizes a number of protocols. To start with, there is a protocol for encoding and decoding speech. A Pulse Code Modulation (PCM) system is defined in ITU recommendation G.711 [22]. It encodes a single voice channel by sampling it 8000 times per second and encoding it as an 8-bit sample, resulting in uncompressed speech at 64 kbps. All H.323 systems must support G.711. However, speech compression protocols are also permitted (but not required). They use different compression algorithms and encodings, traditionally based on making different trade-offs between quality and bandwidth.

Since multiple compression algorithms are permitted, a protocol is needed to allow the terminals to negotiate which algorithm they are going to use for a given session. This protocol is called H.245. It also negotiates other aspects of the connection, such as the bit rate. Also required is a protocol for establishing and releasing connections, providing dial tones, generating ringing sounds, and the rest of the standard telephony. ITU Q.931 [46] is used for this. Additionally the terminals need a protocol for talking to the gatekeeper (if present), for this purpose, H.225 [47] is used. The PC-to-gatekeeper channel is called the Registration/Admission/Status (RAS) channel. This channel allows terminals to join and leave the zone, request and return bandwidth, and provide status updates, among other functions. Finally, a protocol is needed for the actual data transmission. RTP is used for this purpose, and as usual it is managed by RTCP, as usual. The relations between all these protocols is shown in Figure 6.

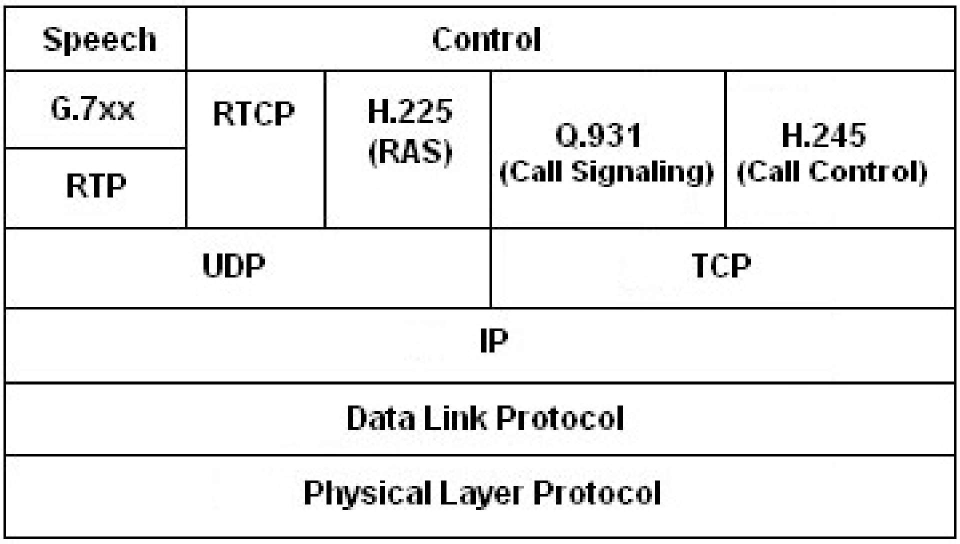


Figure 6: The H.323 protocol stack

2.2.1.3.2 SIP – Session Initiation Protocol

Because H.323 was designed by ITU, many people in the Internet community saw it as a typical telecommunication standard: large, complex, and inflexible. Consequently, IETF set up a committee to design a simpler and more modular way to provide voice over IP. The major result to date is the Session Initiation Protocol (SIP). This protocol describes how to set up Internet telephone calls, video conferences, and other multimedia connections. Unlike H.323, which is a complete protocol suite, SIP has been designed to interwork with existing Internet applications. For example, it defines telephone numbers as URLs, so that Web pages can contain them, allowing a click on a link to initiate a telephone call (the same way the "mailto" URL scheme allows a click on a link to cause the browser to bring up a program to send an e-mail message).

SIP can establish two-party sessions (ordinary telephone calls), Push-to-talk [23] multiparty sessions (where everyone can hear and speak), and multicast sessions (one sender, many receivers). The sessions may contain audio, video, or data, the latter being useful for multiplayer real-time games, for example. SIP only handles setup, management, and termination of **sessions**. Other protocols, such as RTP/RTCP, are used for data transport. SIP is an application-layer protocol and can run over UDP, TCP, or SCTP. SIP supports a variety of services, including locating the callee (who may not be at his home machine) and determining the callee's capabilities and preferances as well as handling the mechanics of call setup and termination. In the simplest case, SIP sets up a session from the caller's computer to the callee's computer, so we will examine that case first.

The SIP protocol is a text-based protocol modeled on HTTP. One party sends a message in ASCII text consisting of a method name on the first line, followed by additional lines containing headers for passing parameters. Many of the headers are taken from MIME [48] to allow SIP to interwork with existing Internet applications. The six methods defined by the core specification are listed in Table 2.

Method	***Description***
INVITE	Request initiation of a session
ACK	Confirm that a session has been initiated
BYE	Request termination of session
OPTIONS	Query a host about its capabilities
CANCEL	Cancel a pending request
REGISTER	Inform a redirection server about the user’s current location

Table 2: SIP’s methods

2.2.2 Speaker recognition

We can differentiate between speaker *identification*, which means *identifying* an user from a set of known users, or speaker *verification*, which consists in *verifying* if the user is who they claim to be. Figure 7, illustrates a speaker recognition system. It is composed of the following modules:

1. *Front-end processing*

   The "signal processing" part, which converts the sampled speech signal into a set of *feature vectors*, which characterize the properties of speech that can distinguish different speakers. Front-end processing is performed both in *training*- and *recognition* phases.

2. *Speaker modelling*

   Performs a reduction of feature data by modelling (typically clustering) the distributions of the feature vectors.

3. *Speaker database*

   The speaker models are stored here.

4. *Decision logic*
Makes the final decision about the identity of the speaker by comparing a unknown set of feature vectors to all models in the database and selecting the best matching model, thus identifying the speaker.

As the set of possible speakers who might use a given device is often small, we can use speaker recognition to *personalize* the device, i.e., we automatically install a given user profile for this device.

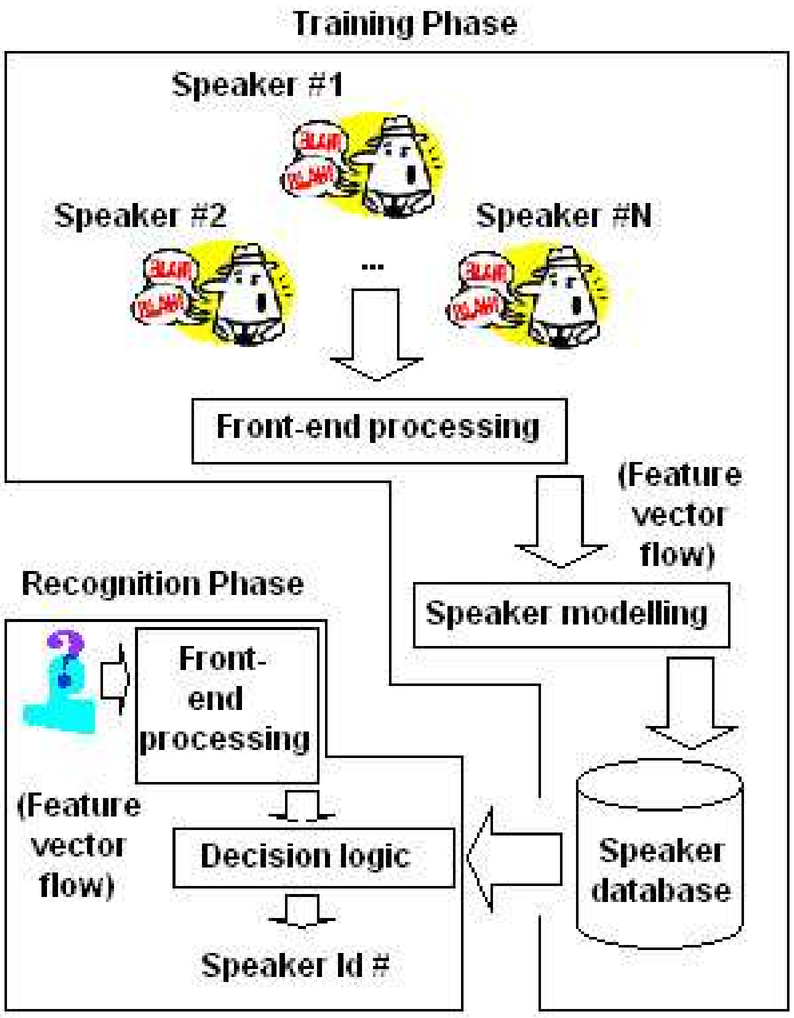


Figure 7: Speaker recognition system modules

2.2.2.1 Speech Signal Acquisition

Initially, the acoustic sound pressure wave is transformed into a digital signal suitable for voice processing. A microphone or telephone handset can be used to convert the acoustic wave into an analog signal. This analog signal is conditioned with antialiasing filtering (and possibly additional filtering to compensate for any channel impairments). The antialiasing filter limits the bandwidth of the signal to approximately the Nyquist rate (half the sampling rate) before sampling, to prevent aliasing. The conditioned analog signal is then sampled to form a digital signal by an analog-to-digital (A/D) converter. The result is a digital encoding of the speech signal as a time series.

2.2.2.2 Feature Selection

The speech signal can be represented by a sequence of feature vectors. Traditionally, pattern-recognition paradigms to be applied to these vectors are divided into three components: feature extraction and selection, pattern matching, and classification.

Feature extraction is the estimation of variables, called a feature vector, from another set of variables (e.g., an observed speech signal time series). Feature selection is the transformation of these observation vectors to feature vectors. The goal of feature selection is to find a transformation to a relatively low-dimensional feature space that preserves the information pertinent to the application while enabling meaningful comparisons to be performed using simple measures of similarity.

2.2.2.3 Pattern Matching

The pattern-matching task of speaker *verification* involves computing a match score, which is a measure of the similarity of the input feature vectors to some model. Speaker models are constructed from the features extracted from the speech signal. To enroll users into the system, a model of the voice, based on the extracted features, is generated and stored (possibly encrypted on an smart card). Then, to *authenticate* a user, the matching algorithm compares/scores the incoming speech signal in comparison with the model of the claimed user while for speaker recognition we simply return the closest match to the input feature set.

There are two types of models: stochastic models and template models. In stochastic models, the pattern matching is probabilistic and results in a measure of the likelihood, or conditional probability, of the observation given the model. For template models, the pattern matching is deterministic.

Pattern-matching methods include dynamic time warping (DTW), hidden Markov model (HMM), artificial neural networks, and vector quantization (VQ). Template models are used in DTW, statistical models are used in HMM, and codebook models are used in VQ. For more information see [4] and [5].

2.2.2.4 Classification and Decision Theory

Having computed a match score between the input speech-feature vector and a model of the claimed speaker's voice, a *verification* decision is made whether to accept or reject the speaker or to request another utterance (or, without a claimed identity, an identification decision is made). The accept or reject decision process can be an accept, continue, time-out, or reject hypothesis-testing problem. In this case, the decision-making, or classification, procedure is a sequential hypothesis-testing problem.

An example of the implementation of a speaker verification and speech recognition system can be found in [24].

2.2.3 Speech Recognition

Speech recognition refers to the process of translating spoken phrases into their equivalent strings of text. A possible approximation of this process is described at the following figure:

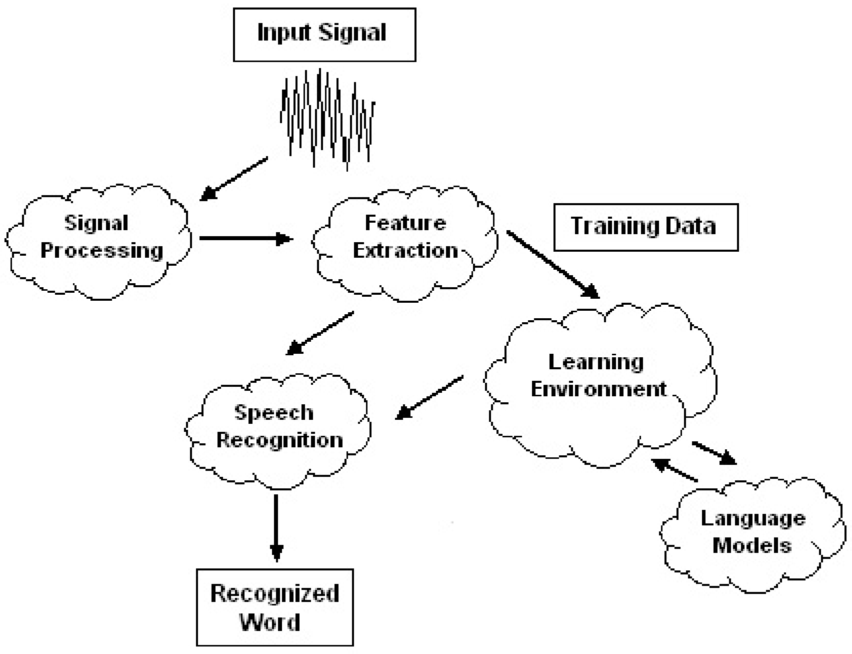


Figure 8: Speech recognition system modules

Having this scheme in mind, now we can explain this process in more detail:

1. Preparing the signal for processing

   After capturing the signal by means of a device as a microphone, the first step is preparing it for the recognition process. One of, the most important treatments of the signal is the one to detect the **presence of** speech in the signal, thus discarding those parts of the signal corresponding to silences. Once we have identified the parts of the signal containing silences, we are able to isolate the words that form the spoken phrase.

2. Signal modelling

   This step consists of representing the spoken signal as an equivalent sequence of bits and extract parameters from it that will be useful for posterior statistic treatments.

3. Vector quantizations

   Vector Quantization VQ is the process where a continuous signal is approximated by a digital representation (quantization) utilizing a set of parameters to model a complete data pattern (i.e., a vector).

4. Phone estimations

   A phone is the acoustical representation of a *phoneme*. Thus, the “sound” emitted when a “letter” is pronounced, would be the correspondent phone of that particular phoneme. The goal of phone estimation in speech recognition technology is to produce the most probable sequence of phones that represent a segmented word for further classification with other higher level recognizers (word recognizers). In this phase, the distance between training vectors and test frames is computed to produce a pattern-matching hypothesis.

5. Word recognition

   The last step is word recognition, here the most probable world obtained during all the processing is returned as output.

Additionally, higher level reasoning (spell checker, grammar checker, speaker models, …) can also be used.

2.2.4 Microsoft Speech SDK

The Microsoft Speech SDK (SAPI 5.1) [27], provides a high-level interface between the application we want to build and the underlying speech engines. The SAPI implements all the low-level details needed to control and manage the real-time operations of various speech engines.

There are two basic types of SAPI engines available, text-to-speech (TTS) systems and speech recognizers. TTS systems synthesize text strings and files into spoken audio using synthetic voices. Speech recognizers convert (human) spoken audio into (readable) text strings and files.

2.2.4.1 API for Text-To-Speech

Applications can control text-to-speech (TTS) using the ISpVoice Component Object Model (COM) interface [49]. The first step in creating a TTS application using this API is to create an ISpVoice object. Subsequently the application only needs to call ISpVoice::Speak() to generate speech output from some text data. In addition, the

IspVoice interface also provides several methods for changing voice and synthesis properties, such as speaking rate (ISpVoice::SetRate), output volume (ISpVoice::SetVolume), and changing the current speaking voice (ISpVoice::SetVoice).

2.2.4.2 API for Speech Recognition

The equivalent of ISpVoice as the main interface for speech synthesis, is for speech recognition the interface ISpRecoContext.

An application has the choice of two different types of speech recognition engines (ISpRecognizer). A shared recognizer that could possibly be shared with other speech recognition applications is recommended for most speech applications, mainly those using a microphone as input. In this case, the SAPI will set up the audio input stream, and select the SAPI's default audio input stream. For large server applications that would run alone on a system, and for which performance is important, an InProc speech recognition engine is more appropriate. Here the audio input stream will be set to a file which will contain the audio to be recognized. However, as we will see, the later approach has greater delay.

Once we have set the input for the recognizer, be it shared or InProc, the next step is to define the events that are of interest to us. We can subscribe the recognizer to many different sets of events, but the most important will be “Recognition”. This set of events will be raised each time that a recognition takes place, then its event handler will invoke the code that we want to be executed (each time).

Finally we need to define a grammar containing the words that we want to use.

2.2.5 Wireless Local Area Network (WLAN)

Wireless Local Area Networks (WLANs) are designed to cover limited areas such as, buildings and office areas. Today they are becoming more and more widely used not only in office and industrial settings, but also on the university campus and at users’ homes.

Just as in an Ethernet LAN, every device has its own Media Access Control (MAC) address in order to be able to distinguish the link layer end points of the transmissions. IP addresses, can be statically or dynamically mapped to these MAC addresses.

IEEE 802.11 [50] is the family of specifications developed by IEEE for WLAN technology. Some of the members of this family include:

1. *802.11*
   Wireless LAN up to 2 Mbps transmission in the 2.4 GHz band, ISM band.

2. *802.11a*
   *A*n extension to 802.11 providing up to 54 Mbps in the 5 GHz band.

3. *802.11b*
   *A*n extension to 802.11 providing up to 11 Mbps in the 2.4 GHz band.

4. *802.11g*
   Provides 20+ Mbps in the 2.4 GHz band.

2.2.6 HP iPAQ h5550 Pocket PC

Some of the interesting features of this hand held device are the following:

- Integrated biometric fingerprint reader can be used to protect the information stored in the Pocket PC. Software allows the user to easily authenticate himself/herself to the device using his or her fingerprints, or a combination of a PIN code and/or fingerprints.

- Increased memory capacity (128 MB RAM) enables the user to store many programs and files. With the iPAQ File Store, up to 17 MB Flash ROM, enables the user to store data in a safe place protected from battery discharge or device resets.

- An integrated IEEE 802.11b WLAN interface enables high speed wireless access to the internet or intranet.

- Integrated Bluetooth® wireless technology allows printing to a Bluetooth equipped printer, access to the Internet via a Bluetooth enabled mobile phone, or use of a Bluetooth headset.

Further detailed specifications of this PDA are shown in the following table:

Operating system preinstalled	Microsoft® Windows® Pocket PC 2003 Premium
Enhanced security	Biometric Fingerprint Reader
Connectivity	Integrated Bluetooth® wireless technology, WLAN 802.11b
Expansion slot	SD slot: SD, SDIO, and MMC support
Processor	Intel® 400 MHz processor with Xscale™ technology
Memory, std.	128 MB SDRAM, 48 MB Flash ROM
Display	Transflective TFT LCD, over 65K colors 16-bit, 240 x 320 resolution, 3.8" diagonal viewable image size
Input type	Pen and touch interface
Audio	Microphone, speaker, and a four pole 3.5 mm headphone jack providing output and mono input to/from a headset
External I/O ports	USB slave and serial I/O
Dimensions (L x W x H)	13.8 x 8.4 x 1.6 cm.
Weight	206.5 g

Table 3: HP iPAQ h5550 specifications

2.2.7 Microsoft's .NET Framework and .NET Compact Framework

The Microsoft's .NET Framework is made up of four parts: a Common Language Runtime, a set of class libraries, a set of programming languages, and the ASP.NET environment. This framework was designed with three goals in mind. First, it was intended to make Microsoft's Windows' applications much more reliable. Second, it was intended to simplify the development of Web applications and services that not only work in the traditional sense, but also work on mobile devices as well. Lastly, the framework was designed to provide a single set of libraries that would work with multiple languages.

The .NET Compact Framework is its equivalent for portable devices. The .NET Compact Network, as it name states, is a *compact* version of the .NET Framework, it contains *most* of the features of the .NET Framework, but some features are missing due to the differences between the architectures and operating systems of Windows for portable and non-portable devices.

One of the most important features of the .NET Framework, is the portability of the code. Using Visual Studio .NET, the code that is output by the compiler is encoded in a language called Microsoft Intermediate Language (MSIL). MSIL consists of a specific instruction set that specifies how the code should be executed. However, MSIL is not an instruction set for a specific physical CPU, but rather MSIL code is turned into CPU-specific code **when** the code is run for the first time. This process is called *"just-in-time"* compilation (JIT). A JIT compiler translates the generic MSIL code to machine code that can be executed by the CPU we are currently using.

By installing the .NET Framework in our laptop computer and the .NET Compact Framework in our portable device, we obtain JIT compilers for both of them, thus we can generate MSIL code and afterwards the JIT compiler can generate the specific CPU code for either or a laptop or a handheld, i.e., the specific device we want to use to run our application.

To install and configure both frameworks in our system, given that we had already installed Microsoft's Active Sync, required installing the following:

1. Microsoft's Visual Studio .NET 2003 [25]
   This provides an integrated development environment for C# for mobile applications.

2. Microsoft's Pocket PC 2003 SDK [13]
   This provides the specific libraries and emulators for use when developing applications for use a Pocket PC 2003 equipped device.

3. Microsoft's .NET Compact Framework 1.0 SP2 [11] and [14]
   This provides the specific libraries for use on top of the Pocket PC 2003 operating system.

4. Microsoft's Windows Mobile Developer Power Toys [15]
   Some useful tools for developing and testing mobile applications.

Inside the directory created after installing Microsoft's Windows Mobile Developer Power Toys, we can find several useful tools. Here we specifically mention two of them, contained in the folders named: RAPI_Start and PPC_Command_Shell. These tools provide the ability to remotely initiate an application and a shell window on the device respectively. Following the instructions of the "readme" files in both folders installs both tools.

To remotely install of all these components, the reader should follow the recommendations of [7] and [8]. After installing all this software in our laptop and then via Active Sync to our PDA, we were ready to start developing mobile applications.

2.2.8 Playlists

Today we are surrounded by mobile audio devices, many of them have a very large amount of storage, hece the usage of playlists is essential. Not only for mobile audio devices, but also for non-portable ones, playlists are really useful, otherwise the user would have to manually or randomly select the next song to play.

A playlist can be described as a metafile that contains the required information for playing a set of pre-selected tracks. The format of these files can vary, depending on the player that is going to be used. Some examples of possible formats can be: *.asx, .m3u, . wvx, .wmx,* and the most gereric one, *.xml*. We say "the most generic one" because most of the possible extensions used for building playlists, are proprietary or dependent on the specific player that is going to be used. In the case of Extensible Markup Language (XML), a playlist can be described such that a simple application can play the desired audio content using whichever player necessary.

More information about playlists formats can be found in [34], [35], [36], and [37].

2.2.9 Extensible Markup Language (XML)

Extensible Markup Language (XML) [38] is markup language very similar to HyperText Markup Language (HTML) [39], but with some differences:

- XML was designed to **describe** data and to focus on *what* the data is.
- HTML was designed to **display** data and to focus on how data *looks*.
- HTML's focus is about displaying information, while XML's is describing information.

The tags used to mark up HTML documents and the structure of these tags are predefined. XML, on the contrary, allows the author to define his own tags and his own document structure.

It is important to understand that XML is not a replacement for HTML. Future web development most likely will use XML to describe the data, while HTML will continue to be used to format and display the content. When HTML is used to display data, the data is stored inside HTML. With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for data layout and display, while being sure that changes in the underlying data will not require any changes to your HTML. XML data can also be stored inside HTML pages as "Data Islands". Thus continuing to use HTML only for formatting and displaying the data.

One of the main features of XML is that data is stored in plain text format, this enables XML to provide a software- and hardware-independent way of sharing data. This makes it much easier to create data that different applications can work with.

XML can also be used to store data in files or in databases. Applications can be written to store and retrieve information from the data base, and generic applications can be used to display the data.

2.2.10 Microsoft's ActiveSync

Microsoft's ActiveSync [51] is a tool to provide synchronization between a computer and a handheld device. It offers the possibility of synchronizing e-mail, favourites, and shared files between the computer and the hanheld device. This feature for sharing files is the most interesting for our study because it will provide us a method to exchange files between both machines.

ActiveSync enables the use of synchronization services over a serial link, USB, infrared, or over TCP/IP (which could run over WLAN, Bluetooth, or an additional interface network card). Regarding the first two possibilities, the handheld has to be docked in its cradle to perform the synchronization and at the same time, the cradle has to be connected to the computer which we want to synchronize with. When using infrared, then IrDA ports both at the computer and the device have to be active, pointed at each other, and ready to send and receive data. For mobile device, the most interesting way of performing synchronization is wirelessly, over TCP/IP. In this case, ActiveSync listens on port 5679 of the host PC for a PDA attempting network synchronization. When a PDA is synchronized through the cradle, port 5679 is closed.

2.2.11 Windows Mobile Developer Power Toys

Windows Mobile Developer Power Toys [15], are a set of tools whose main purpose is to allow the developer to test mobile applications as they are being built. The most interesting "toys" are:

- CeCopy: given a file on the laptop computer that we are running, using CeCopy, we can copy it to the PDA using the following statement:
  - CeCopy [options] <Source_FileSpec> <Destination>

- RapiStart: enables the user to launch a program remotely from the laptop computer to the PDA using the following statement:
  - RapiStart <executable> <arguments>

- CmdShell: a shell (on the PDA) for executing commands.

2.2.12 Context Information

Context information can be described as the set of data related with the user, device being used, situation, environment, time, and all the possible combinations that can be considered of interest for a concrete purpose. After retrieving this information and processed it, we draw conclusions and make decisions according to them.

When an application uses context information, it is said to be "Context-Aware" and its context-awareness depends on the context information that it uses. Not all of the possible data available is relevant for an application. Some of the possible context items that an application can use are shown at the following table:

Network-Awareness	Transmission rate, link quality, RSSI, AP being used
Memory-Awareness	Total capacity, available capacity
Storage-Awareness	Total capacity, available capacity
Battery-Awareness	Percentage of availability, remaining life time

Table 4: Some possible Awareness for a mobile device

3 Design

3.1 Overview

For our study, we want to introduce and compare two different systems. The first one corresponds to Pocket Streamer (an explanation about it and an overview can be found in section 2.1.2). Remember that Pocket Streamer consisted of a server (running at the laptop) and a client (running at the PDA). First the user has to start the server, then the client, and after selecting a playlist at the client, the audio is streamed from the laptop to the PDA.

The second system that we are going to use for our study is shown in the following image:

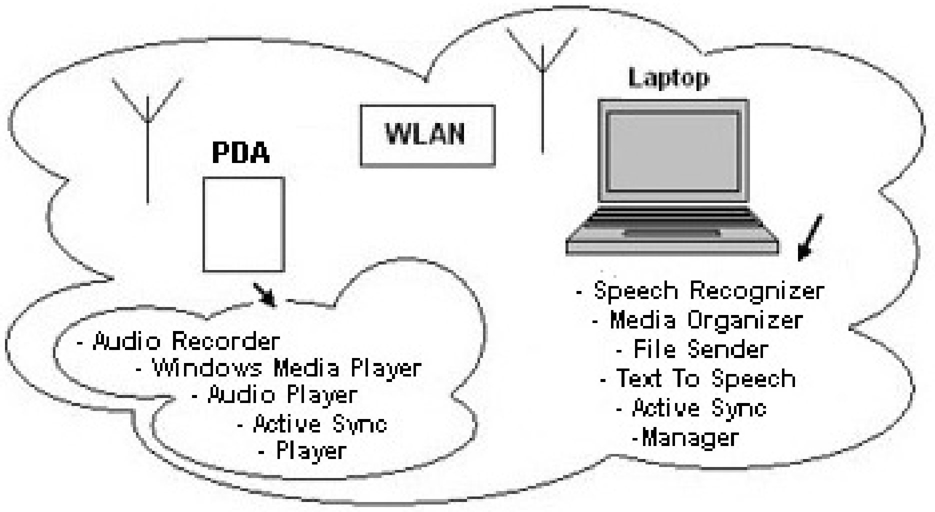


Figure 9: Architecture of the second system

As can be seen in Figure 9, we use a PDA and a laptop computer, both are running Microsoft's Windows Operating Systems, in the case of the PDA, Microsoft® Windows® Pocket PC 2003 Premium, and in the case of the laptop, Microsoft's Windows XP. Microsoft's Active Sync 3.7 is also installed in both devices. A possible schema of this system running (when using the voice interface) is shown in Figure 10. An explanation of all the applications that are going to be used, can be found in the following sections.

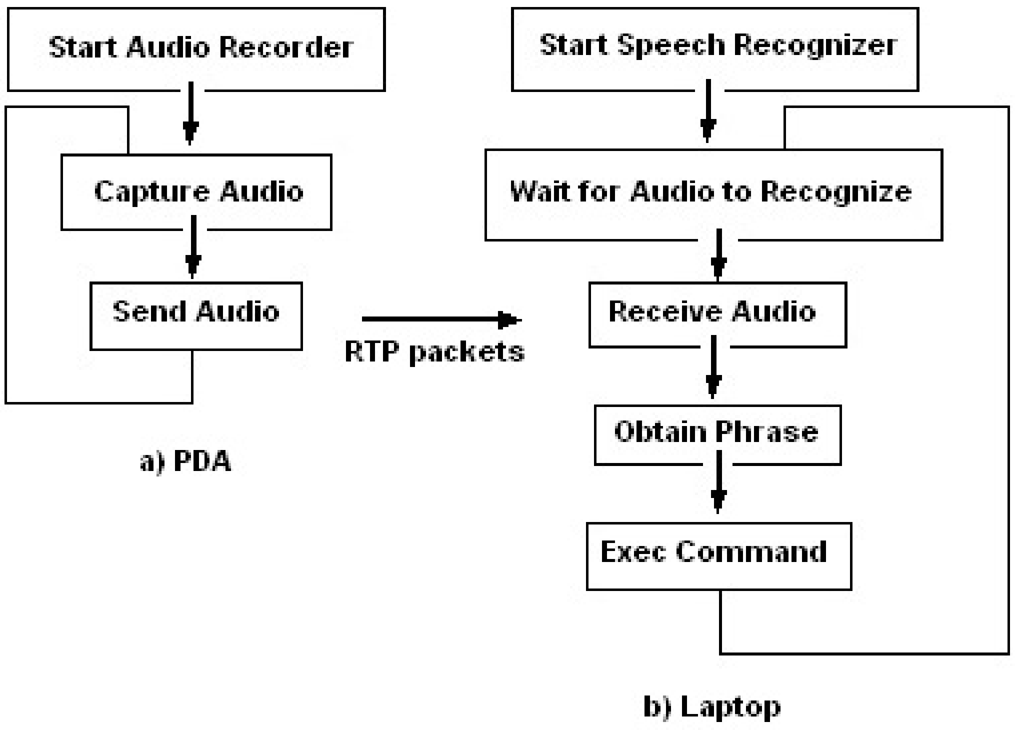


Figure 10: Flow of execution of the system

Figure 10 a) shows the execution on the PDA and the Figure 10 b) the laptop. The Audio Recorder captures audio input at the PDA, encapsulates it in RTP [20] packets and sends them to the Speech Recognizer. The later application is always waiting for audio to recognize, when it receives the audio packets, it extracts the audio from them, transforms the data received into a stream and passes it as input to the recognition engine. The possible words or phrases recognized (i.e., commands to execute) are:

Word or Phrase Recognized	***Command to Execute (Action Performed)***
Start	Start Player at the PDA
Close	Close Speech Recognizer at Laptop
Play	Start playing selected track at Player at the PDA
Stop	Stop playing at Player at the PDA
Previous	Play the previous track at Player at the PDA
Next	Play the next track at Player at the PDA
Exit	Close Player application at the PDA

Table 5: Available commands

3.1.1 Methodology

In our study, we want to compare the two different systems described above with regard to the following points:

- Compare the amount of traffic which needs to be sent during the network usage peak period via high cost network connection versus the possibility of being able to send traffic only when we have a large amount of low cost bandwidth.

- The effects of errors in the case of streaming audio versus the case in which we are caching and have cached data.

- Brief comparison, from the user's point of view, of both systems. What do users like and dislike about having cached files based on a playlist versus only streamed content.

- Regarding the voice interface, what are the advantages and disadvantages of having voice commands versus typing on the screen of the PDA.

To facilitate this study we propose to study two different system configurations, from this moment on, System 1 will refer to Pocket Streamer (i.e., the case in which audio is streamed) and System 2 will refer to the case in which the audio is stored locally at the PDA.

3.1.1.1 Scenario using System 1.

Eva loves listening to music. She has received as a present a new PDA for her birthday and now she wants to enjoy it as much as possible. Looking at the web she has found an interesting application called Pocket Streamer, she has downloaded it and installed both the server and the client that the application requires.

Once she has installed Pocket Streamer, she decides to organize all the media content that she has at her laptop. For that purpose she starts Windows Media Player and opens the utility Media Library. At this point she selects her favourite songs and adds them to the Media Library, then closes Windows Media Player.

Before going to visit her friend Susana, she decides to take her new PDA with her to listen to music on the way to Susana's house. She starts a Pocket Streamer Server on her laptop and Pocket Streamer Client on the PDA and leaves. On her way, she refreshes the list of media content, previously organized at the laptop, to the PDA, selects a playlist and starts listening to her favourite songs.

When she arrives at Susana's house she stops the currently playing track to resume for her way back home.

3.1.1.2 Scenario using System 2

Eva was generally very happy with the previous system, but she found that there were places where she lost the contact with the server. Some days later she hears about another possibility and decides to test it, too. For this new system, she installs Player, Audio Player, and Audio Recorder on the PDA and Speech Recognizer, Media Organizer, TextToSpeech, Manager, and File Sender on her laptop.

As she had already organized her media content some days before using Windows Media Player and Windows Media Library on the laptop, there is no need to do it again.

Following the instructions for this new system, she decides to start the Media Organizer and select her favourite songs to form a new playlist. Once she has decided the order of all the songs she exits the Media Organizer after creating an XML file containing her desired playlist.

While she does her homework, she decides to transfer the audio files to the PDA to have it prepared for when she will go out later. To do this, she starts the Player program on the PDA and the Manager program on the laptop computer. Then at the Player she selects to download new content by entering the following file name:

"MyPlayList.xml"

The Player program will verify if it is an XML file and if so it will send a message to the Manager to ask for information about "MyPlayList.xml". The Manager will check that the file exists and if so it will send an answer to the Player containing the number of Megabytes and the playout minutes of the playlist. With this information, the Player will first check, if it has storage enough to save the audio files, then it checks if it has battery enough to download all the audio content and finally it checks the network state. If all these parameters are favourable then a message will be sent to the Manager in order to start downloading the playlist by means of the File Sender program. While the transmission takes place, a connection is established between the Player and the File

Sender program in order to be able to monitor the network state and recover from errors or stop and wait until better conditions exist, if necessary.

The new audio content will be stored in a new 512Mb SDIO memory card inserted into the PDA that Eva received also as a present from her parents and her sister.

While Eva finishes her homework, the audio content is downloaded to her PDA. Now she has finished studying, she starts the Audio Recorder on the PDA and the Speech Recognizer on the laptop and goes for a walk to have some fresh air after a long study session.

On her way she says to her PDA: “Start”. The Audio Recorder gets this audio and sends it to Speech Recognizer at the laptop. The phrase is recognized and Eva sees that the Player is started on the PDA. She loads an existing playlist and presses “Play” to start listening to this audio. After listening for some seconds to this song she decides that she doesn’t like it so much so she wants to go to the next one. For this purpose she has two options, either say: “Next” or press the “Next” button on the screen.

Eva’s parents have told her to call them at 18:35, to remind herself, she decides to add a new audio alert. By pressing the “Audio Alerts” button she starts this process. First she enters the time when the audio alert has to be played, in this case, 18:35, and then the text which in this case will be “Remember to call your parents”. The message with the text will be sent to the Manager program and via the TextToSpeech program the message will be synthesized and the resulting .wav file stored at the PDA. The audio alert will be now ready to be played at 18:35. When the time arrives, the current playing is stopped to be able to play the audio alert and when this has finished, the Player program will continue with the current track.

Suddenly, she realizes that the cached audio content will not be enough for all the time she is going to be out and that she would like to get some additional tunes. She presses the button “New Content”, and the same process as before at home is started for downloading extra content from the laptop computer. In the mean time, she continues listening to the local cached audio content.

When she comes back home she decides to stop the Player, again she can, either say “Exit” or press the “Exit” button.

3.2 Context information use with these applications

As stated in section 2.2.12, context information can be described as the set of data related to the user, the device being used, the user’s situation, the local environment, time of the day and all the possible combinations. In that section, some examples of the possible items of information that could be used to provide an application with context-awareness were given.

The items of information that we consider as most interesting for our purposes and that we are going to use to provide our system with context-awareness are the following:

- *Storage-Awareness*
  - Available capacity: refers to the total amount of free space, measured in Mb, in all the locally available storage devices.

- *Battery-Awareness*
  - Percentage of availability: the available battery measured in percentage.
  - Remaining life time: minutes of remaining battery life.

- *Network-Awareness*
  - Link quality: the quality of the link measured in percentage.
  - RSSI: Received Signal Strength Indicator.

To illustrate the use of the *Network-Awareness'* a simple experiment was performed. As an access point we have used a D-Link AirPlus G+ Wireless Router [53] which was situated inside a room. To make this experiment more realistic (i.e., more similar to a normal use of our system), we have taken our PDA and gone to the street so as to be able to measure how the signal strength that we receive from this access point decreases with the distance. The measures were taken directly from the PDA. A graph containing the results obtained is shown in figure 11:

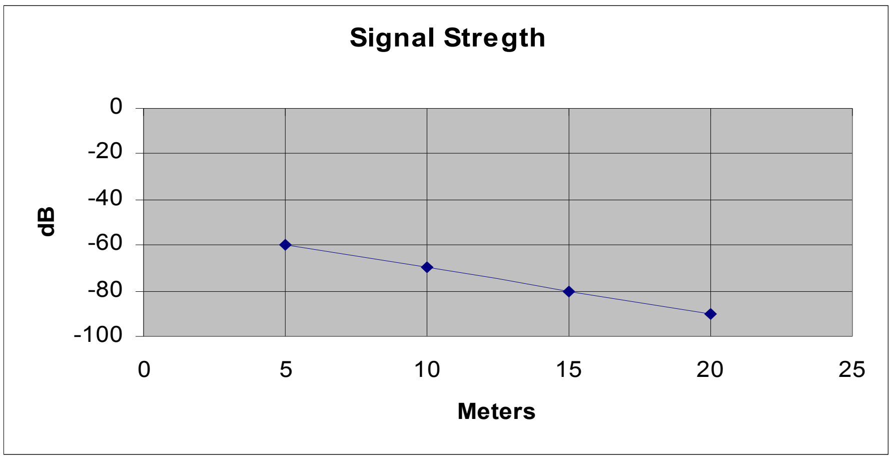


Figure 11: Signal Strength

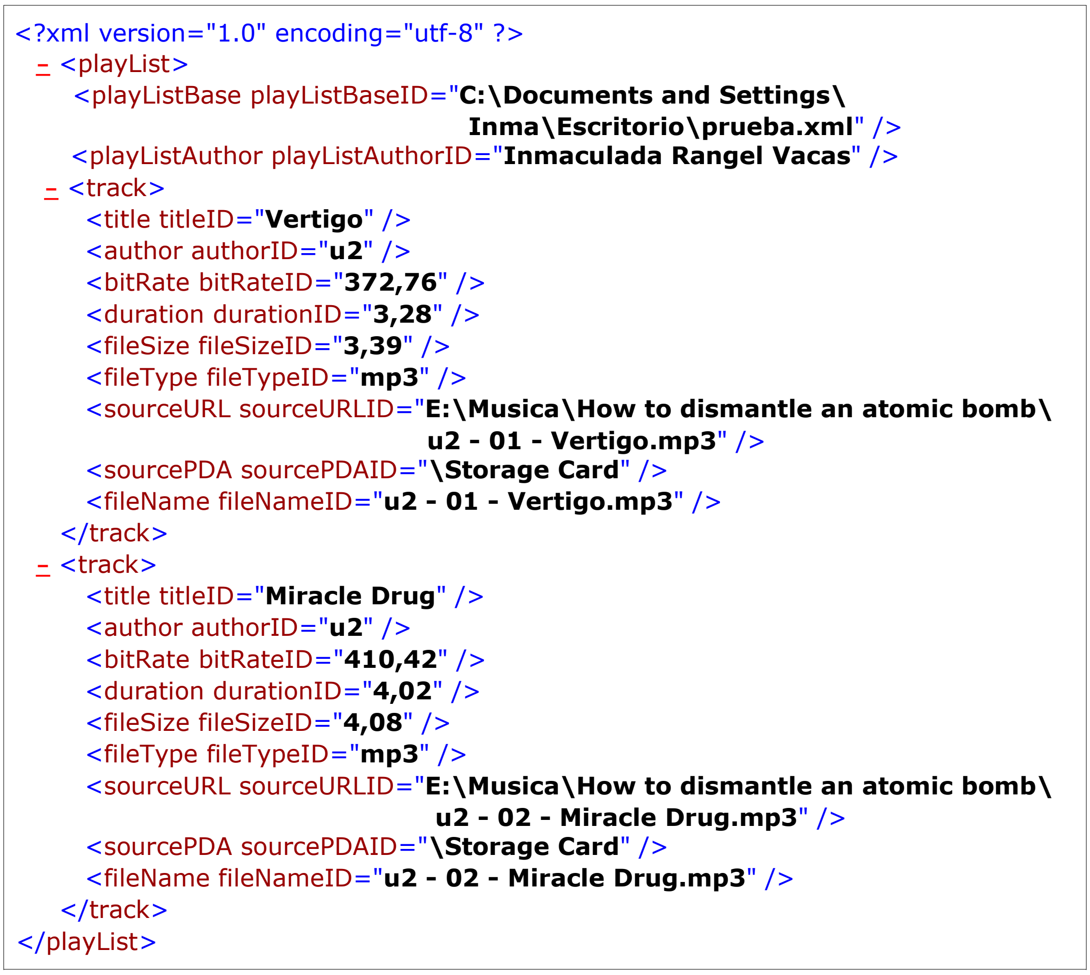
3.3 Playlists representation

As stated in section 2.2.8, a playlist can be considered to be a metafile containing information about a set of audio content to be played at some later time. It was also stated that there are several formats for a playlist. For our system, we have decided to use XML [38], to represent our playlist. The reasons for using this format and not another are mainly due to the fact that we did not want to force the user to use a specific format of playlist which might limit them to a specific player. By using XML the default player that we are using, can easily be substituted by another player.

The elements and attributes that we have used in our playlist are the following:

- playListBase / playListBaseID: the full name and location of the XML file that contains the playlist.
- playListAuthor / playListAuthorID: the author of the playlist.
- track
    - title / titleID: the title of the track.
    - author / authored: the group or solist author of the track.
    - bitRate / bitRate: bit rate of the track in bits per second.
    - duration / durationID: duration of the track in minutes.
    - fileSize / fileSizeID: size of the file in Mb.
    - fileType / fileTypeID: type of the file (mp3, wav …).
    - sourceURL / sourceURLID: location of the file at the laptop.
    - sourcePDA / sourcePDAID: location where the file will be at the PDA.
    - fileName / fileNameID: name of the file (without location).

An example of a possible playlist is shown in figure 12, only two tracks have been added in order to simplify the example:

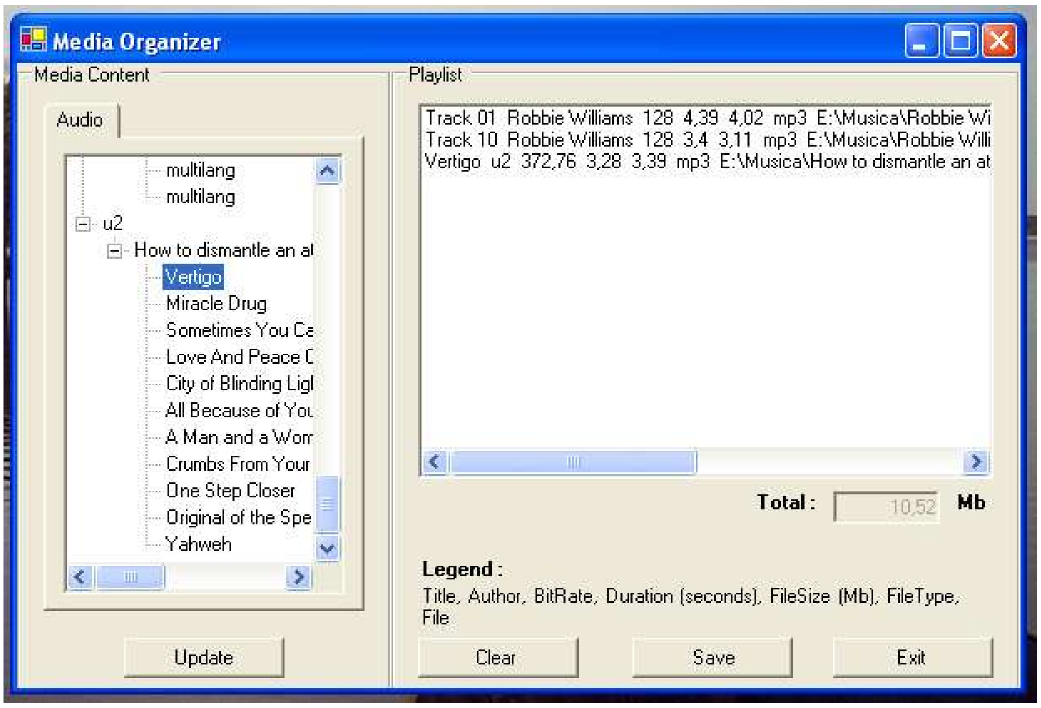


```
<?xml version="1.0" encoding="utf-8" ?>
 - <playList>
     <playListBase playListBaseID="C:\Documents and Settings\
                                   Inma\Escritorio\prueba.xml" />
     <playListAuthor playListAuthorID="Inmaculada Rangel Vacas" />
  - <track>
      <title titleID="Vertigo" />
      <author authorID="u2" />
      <bitRate bitRateID="372,76" />
      <duration durationID="3,28" />
      <fileSize fileSizeID="3,39" />
      <fileType fileTypeID="mp3" />
      <sourceURL sourceURLID="E:\Musica\How to dismantle an atomic bomb\
                              u2 - 01 - Vertigo.mp3" />
      <sourcePDA sourcePDAID="\Storage Card" />
      <fileName fileNameID="u2 - 01 - Vertigo.mp3" />
    </track>
 - <track>
      <title titleID="Miracle Drug" />
      <author authorID="u2" />
      <bitRate bitRateID="410,42" />
      <duration durationID="4,02" />
      <fileSize fileSizeID="4,08" />
      <fileType fileTypeID="mp3" />
      <sourceURL sourceURLID="E:\Musica\How to dismantle an atomic bomb\
                              u2 - 02 - Miracle Drug.mp3" />
      <sourcePDA sourcePDAID="\Storage Card" />
      <fileName fileNameID="u2 - 02 - Miracle Drug.mp3" />
    </track>
</playList>
```

Figure 12: Playlist representation

3.4 Description of the Media Organizer program

This application was developed using Microsoft's Visual Studio .NET 2003 as environment and C# as the development language. The device that will be used to run this application will be a laptop computer. As this application has a Graphical User Interface (GUI), perhaps the best way to introduce it is to show screenshots of its execution.



```
Track 01  Robbie Williams  128  4,39  4,02  mp3  E:\Musica\Robbie Wi
Track 10  Robbie Williams  128  3,4  3,11  mp3  E:\Musica\Robbie Willi
Vertigo  u2  372,76  3,28  3,39  mp3  E:\Musica\How to dismantle an at
```

Total : 10,52 **Mb**

Legend :
Title, Author, BitRate, Duration (seconds), FileSize (Mb), FileType, File

Figure 13: Media Organizer

When the program is first launched, the only buttons available are "Update" and "Exit". If "Update" is pressed, then a call to the Windows Media Library will be performed to obtain the audio content that the user has previously added to the library. This audio content is displayed as a tree at the Audio tab. By browsing this tree, the user can add tracks to the playlist using a popup menu. After the first track is added to the playlist, the buttons "Clear" and "Save" are enabled. Once we have more than one track, it is also possible, by means of a popup menu, to change the order of the tracks or to delete a selected track.

By pressing "Clear" all the items currently in the playlist will be deleted. If the user presses "Save" a dialog will ask the user the name of the XML file in which the plyalist is to be written. Once a name has been entered, the dialog will also check if the file exists and if so, it will query the user if he wants to overwrite it. Finally, the content will be written into the XML file as explained in the previous section.

3.5 Description of the File Sender program

This program is a console application built using Microsoft Visual Studio .NET 2003 as the IDE and C# as the development language. The input that this program receives is the full name of an XML file (its full pathname). An example of a call to this program could be:

FileSender.exe “E:\Proyecto\PlayLists\MyPlayList.xml”

The program will check if the file exists, if it is valid and if so it will perform the following actions:

- Create a new process to start the tool “CeCopy” to copy to the PDA the XML file containing the playlist.

- Start reading the XML file and for each track, copy it to the PDA using the same procedure as above. The procedure of copying each track is explained in more detail later.

- For each audio file sent to the PDA, the program will first check if the WLAN link existing between the laptop computer and the PDA is still active, if so, it will ask the Player application about the state of this link. If the link quality is below a certain threshold, then the File Sender will wait and check again 10 seconds later. When these parameters will be favourable again, then the transmission will continue.

This program uses the “CeCopy” tool for downloading files to the PDA and “CeCopy” at the same time is using ActiveSync to copy files from the laptop computer to the PDA. The reason for using this tool rather than another is because we have found that when using Microsoft’s Windows operating systems in both the laptop computer and the handheld device, the most common way of transmitting files between them is using ActiveSync that is already installed in both machines. Some good features have been found when using “CeCopy” by means of ActiveSync these include:

- Transmission speed ~ 400 Kbytes/s.

- The tool checks if the file to download already exists. If so, then it checks if it is identical to the previous one. In this case it will only perform the download if the “/is” option is provided. For our purposes, we are only interested in download files that have **not** yet been downloaded yet or in the worst case, existing files but that have changed, for this reason do not give the “/is” option.

Not everything is *perfect* when using “CeCopy” and ActiveSync, we have also encountered some problems such as:

- When network connection is lost and we are running ActiveSync wirelessly (as in our case) then ActiveSync is disconnected and the user has to connect it (manually) again when the connection is recovered.

- As “CeCopy” works over ActiveSync, when a disconnection occurs, then the transmission will be stopped and **no** error recovery is provided.

Balancing the good points and the bad ones, we decided that “CeCopy” and ActiveSync were a very effective way of transmitting our audio files but we need to provide our File Sender program with some features to come over the problems:

- As stated above, for each audio file sent, we check the network state.

- If we are disconnected that means that perhaps the last file has not been completely delivered. To solve this we try to send again the file when the connection is recovered.

- Observe that, when the file has not been copied completely, then “CeCopy” will detect that the files are not *identical* so it will transmit it again, while if the file was successfully sent, then it will discover that the files are *identical* so it will continue with the next audio file instead of retransmitting the file.

To illustrate the File Sender performance, two different simple experiments are shown. In the first one, we will call it *Case1*, no errors are expected at the network. Taking this into consideration we can evaluate the performance of sending files **without** checking the network state. The results of this experiment are shown at the following table:

Number of tracks	***Megabytes***	***Minutes***	***Kbytes/s***
11	50,01	2,26	377,66
24	101,35	4,55	380,16
34	150,61	7,21	356,51
46	200,22	9,45	361,60
56	246,66	12,00	350,81
68	302,82	14,44	357,90
79	350,25	16,59	360,31
90	403,52	19,38	355,35
100	453,58	22,09	350,36

Table 6: File Sender performance (*Case 1*)

Now we are going to evaluate a second situation, called *Case 2*, in which network errors can occur and for this reason File Sender will be communicating with the Player program in order to check the connection state. No errors were generated but an increase of the delay when sending the same amount of audio content but being *aware* of the network state can be observed at the following table:

Number of tracks	*Megabytes*	*Minutes*	*Kbytes/s*
11	50,01	2,57	332,10
24	101,35	6,08	284,49
34	150,61	9,05	284,02
46	200,22	12,07	283,11
56	246,66	14,53	289,72
68	302,82	18,12	285,22
79	350,25	21,05	283,97
90	403,52	24,05	286,35
100	453,58	27,04	286,22

Table 7: File Sender performance (*Case 2*)

To be able to evaluate the increase of delay when being *aware* of the network state we have created the following graph comparing the results obtained in *Case 1* and *Case 2*:

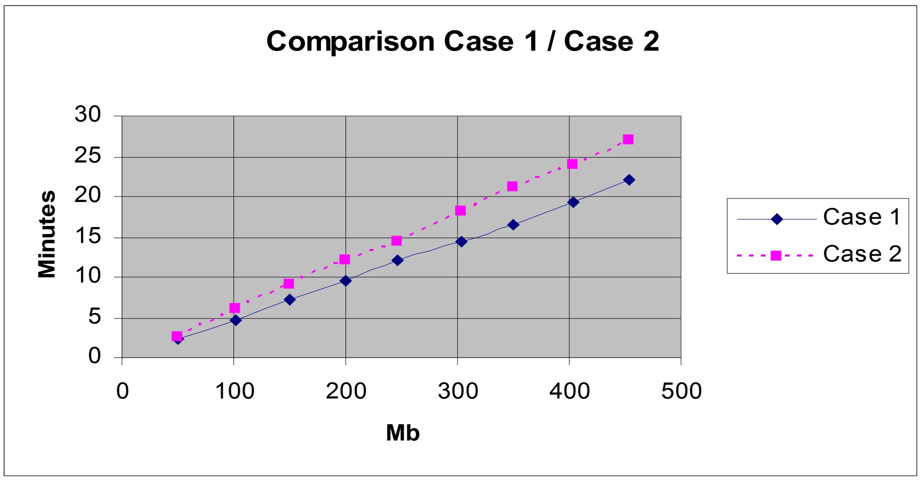


Figure 14: Comparison between *Case 1* & *Case 2* regarding File Sender performance

As can be observed at the last graph, the difference between the time that the audio content needs to be sent in both situations increases with the amount of Mb. The explanation for this fact is that for each audio track that is sent, File Sender is checking the network state so the more tracks we are sending (also more Mb) the more delay we are experimenting due to the network state checking. To be able to evaluate this in more detail, the following table shows the relationship between the amount of tracks sent and the time to complete the transmission. The first column shows the amount of tracks sent, the second one the amount of minutes that takes to sent them all without checking the network state (*Case 1*), and the third one, the same but being aware of the network state (*Case 2*). With all this information, we can add a last column that will perform the difference between the minutes obtained in both cases and divide between the number of tracks in order to get the delay produced by the network state checking.

Tracks	***Minutes (Case 1)***	***Minutes (Case 2)***	***Delay (Minutes)***
11	2,26	2,57	0,03
24	4,55	6,08	0,06
34	7,21	9,05	0,05
46	9,45	12,07	0,06
56	12	14,53	0,05
68	14,44	18,12	0,05
79	16,59	21,05	0,06
90	19,38	24,05	0,05
100	22,09	27,04	0,05

Table 8: Network state checking delay

3.6 Description of the Audio Recorder program

This program was developed by Johan Sverin in [54] using C# as a programming language under Microsoft's Visual Studio .NET 2003 environment. It runs on the PDA and a flowchart of its execution can be found below:

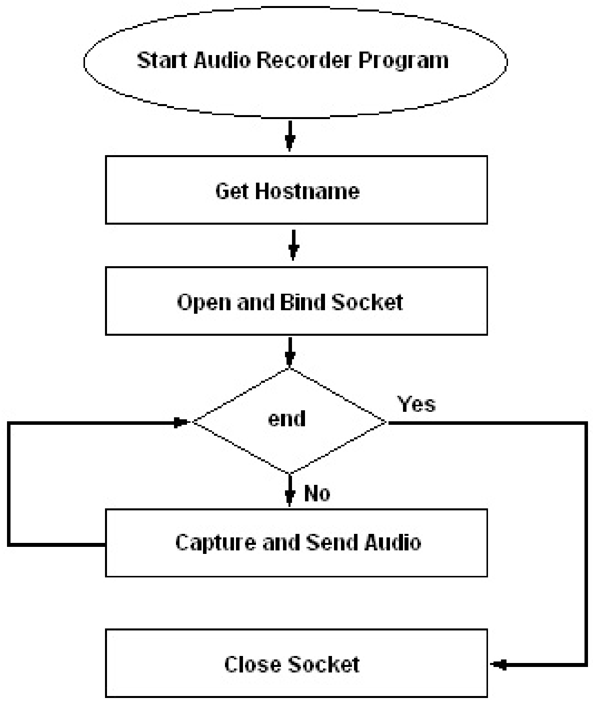


Figure 15: Flowchart of Audio Recorder

As it can be seen in the flowchart, when the Audio Recorder program starts, the first thing that it does is to get the hostname and to open and bind a socket to communicate with the Speech Recognizer at the laptop. Thus it will always be capturing new audio and sending these samples to the Speech Recognizer program.

3.7 Description of the Audio Player program

This is an application developed with Microsoft Visual Studio .NET 2003 as an IDE and using C# as the programming language. The program is run at the PDA and its main purpose is to play a .wav file. It receives as input a string containing the file name (including the full path to the file), checks if the file exists and if it is valid and starts the process to play the file. This uses the Microsoft's Waveform Audio Interface, [40].

3.8 Description of the Player program

This application was built using Microsoft's Visual Studio .NET 2003 as a development environment and C# as the programming language. It has a Graphical User Interface, (GUI), and it runs on the PDA. Screen captures of the program at the start of its execution, are shown in figures 16 and 17.

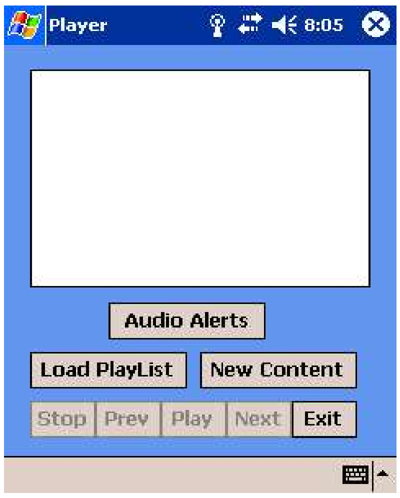


Figure 16: Player Main Form

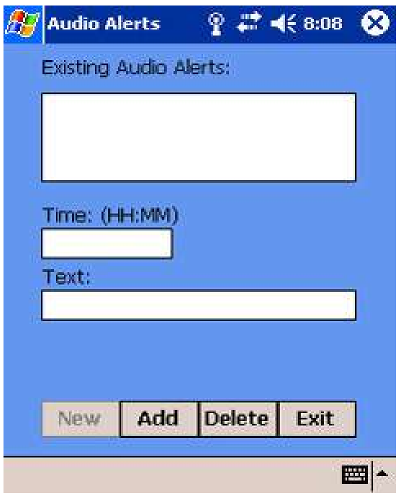


Figure 17: Player Audio Alerts Form

As can be seen, initially, only the buttons "Load PlayList", "New Content", "Audio Alerts", and "Exit" are enabled at the Player Main Form. If "Load PlayList" is pressed, an open file dialog queries the user to select an XML file containing the playlist. Once a file has been selected, the application checks if the file is valid. If so, the content of the file will be read and the playlist displayed within the listbox.

At this point, the "Play" button will become available and if pressed it will start to play tunes in the playlist. In this case two different actions can take place depending on the type of the current file to be played:

- If the file type is mp3 or wma, then Windows Media Player will be launched to play the file.
- If the file type is wav, then the Audio Player will be launched to play this file.

Once the “Play” button is pressed, the buttons “Stop”, “Next”, and “Prev” are enabled. These functions are the same as in all the common players, i.e., stop playing, go to next track, and go to the previous one, respectively.

When “Exit” is pressed, the application is terminated. This ends the current playout.

One of the main features of this application is its context awareness. By pressing the button “New Context”, a dialog will be shown to the user asking for the name of a file containing a playlist. When the name is entered, the Player sends a message to the Manager application at the laptop asking for information about this file. If the file exists and is valid, the reply contains the total size in Mb of the playlist and its playout duration (in minutes). Given this information the Player starts the process of deciding **if** the playlist should be downloaded or not based on the current context information.

First, the Player will check if there is enough space to store all the audio content locally. If so, it will check whether the PDA has enough battery to receive all the audio content. If so, the last check will examine the network state. If the link quality and the RSSI are favourable, then the playlist will start automatically being downloaded. If the conditions are not favourable, then a message box will alert the user that the network conditions are not favourable for downloading and ask if it should proceed anyway or defer to later.

During the transmission of the playlist content, the Player will be answering for the requests of the File Sender about the state of the network. When all the content has arrived to the PDA, then a message box will be shown to the user to inform him/her that the new audio content is ready.

Another interesting feature of this program is that it offers the possibility of inserting audio alerts for the user. By pressing the button “Audio Alerts” at the Player Main Form, another form will be displayed, in this case the Audio Alerts Form. At the beginning, the Audio Alerts Form will just show a list box containing the pending audio alerts. If the user presses the “New” button, then the process of creating a new audio alert is started. Two text boxes are now displayed for the user to insert both the time at which the audio alert has to be played and the message of the audio alert. If these two values are correct (i.e., the time for the audio alert to be played has to be later than the current time), then a message will be sent to the Manager program with the text of the audio alert. The Manager program will process the message and pass the text string to the TextToSpeech program. This last program will perform the synthesis of the text into voice and send the file containing the audio alert back to the PDA.

3.9 Description of the Speech Recognizer program

This application is built using C# using Microsoft’s Visual Studio .NET 2003 IDE. This program runs on the laptop. A flowchart of the Speech Recognition program is shown in figure 18.

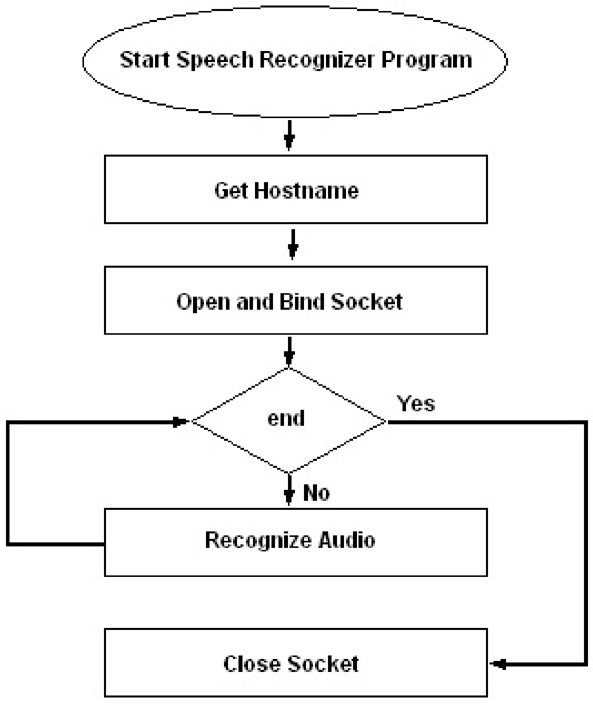


Figure 18: Flowchart of Speech Recognizer

As shown in the flowchart, when the Speech Recognizer program starts, it first gets the hostname of the laptop computer in which is running, and tries to open and bind a socket for communication with the Audio Recorder at the PDA. Subsequently it will always be listening for new incoming audio samples to recognize.

3.10 Description of the Manager program

This application was developed using C# as the programming language using Microsoft's Visual Studio .NET 2003 IDE. This program runs on the laptop. Its flowchart is shown at the following image:

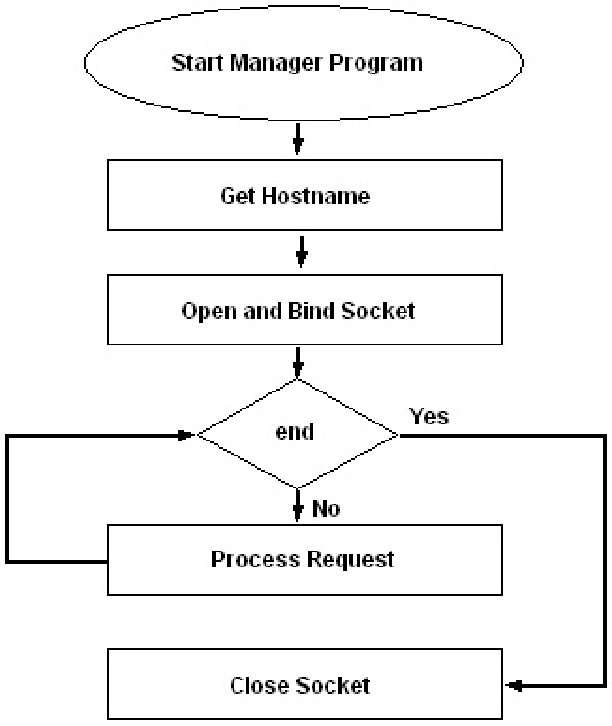


Figure 19: Flowchart of Manager

As described in the flowchart, when the Manager program starts, the first thing it does is to get the hostname of the laptop computer in which it is running and to open and bind a socket for communication with the Player program at the PDA. Subsequently it will always be listening for new incoming requests.

3.11 Description of the TextToSpeech program

This application was developed using C# as the programming language using Microsoft's Visual Studio .NET 2003 IDE. This program runs on the laptop. The main goal of this program is to synthesize a string of text that receives as input and return to create a .wav file.

4 Design Evaluation

Taking into account the design described in the previous section, now we want to evaluate it with regard to the points also described in section 3.1.1.

4.1 Amount of traffic

In this section we evaluate the amount of traffic that has to be sent using the two different systems that we have proposed. Clearly when using System 1 to stream audio content from the laptop we require constant connectivity. Conversely in System 2, when local storage is available, most of the time the player is able to play audio from its local storage and not download extra content, hence it does **not** need constant connectivity.

We will consider the voice interface not being used and we will study its effect on the network traffic later.

4.1.1 Amount of network traffic using System 1

For this study we created a playlist using Windows Media Library and we observed how much traffic had to be sent while the user was listening to the audio specified in this playlist.

Some interesting details of the playlist are that it contains 2 hours of audio distributed over 29 audio tracks, and the total size of these audio files is of 149.94 Mb.

To study the amount of traffic and its patterns we use Ethereal [45]. The results obtained are presented are the following table:

Between first and last packet	7561,207 sec
Packets	121862
Avg. packets/sec	16,12
Avg. packet size	585,682 bytes
Bytes	71372360
Avg. bytes/sec	9439,281
Avg. MBit/sec	0,076

Table 9: Statistics from Ethereal when using System 1

4.1.2 Amount of network traffic using System 2

To be able to compare properly these two systems according to the amount of traffic, we created the same playlist using the Media Organizer program and requested it as new content from the Player program.

We are going to differentiate between two situations, *Case 1*, in which we consider that no errors can occur during the transmission so we don't have to check the network state, and *Case 2*, in which we will be *aware* of the network state during the transmission. This *awareness* will be necessary in the normal use of System 2 to be able to perform secure transmissions but obviously it will increase the network traffic due to the communication between applications to check the network state.

4.1.2.1 *Case 1*

The following table shows the statistics obtained with Ethereal [45] when sending our playlist:

Between first and last packet	443,679 sec
Packets	164737
Avg. packets/sec	371,298
Avg. packet size	987,312 bytes
Bytes	162646858
Avg. bytes/sec	366586,592
Avg. MBit/sec	2,933

Table 10: Statistics from Ethereal when using System 2 (*Case 1*)

4.1.2.2 *Case 2*

The following table shows the statistics obtained with Ethereal when sending our playlist:

Between first and last packet	643,536 sec
Packets	165946
Avg. packets/sec	257,835
Avg. packet size	981,144 bytes
Bytes	162797255
Avg. bytes/sec	252972,980
Avg. MBit/sec	2,024

Table 11: Statistics from Ethereal when using System 2 (*Case 2*)

4.1.3 Comparison between System 1 and System 2 regarding the amount of network traffic

In this section we compare the results obtained in previous sections regarding to the following points:

- Packets sent / received for each system.
- Bytes sent / received for each system.
- Time to perform the transmission in seconds.

4.1.3.1 Packets Sent / Received for each system

The following graph shows the number of packets sent and received for each system while performing the transmission of the stated playlist.

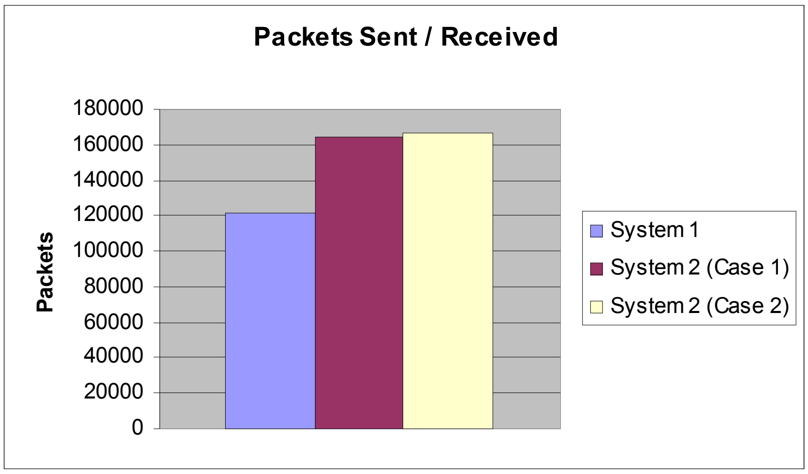


Figure 20: Packets Sent / Received for each system

As it can be seen the worst case is the System 2 in a *Case 2*. This is because in addition to the packets containing the audio tracks, the system is also checking the network state so it needs to exchange more packets due to this network state checking. The following table shows the number of packets sent by each system:

	System 1	*System 2 (Case 1)*	*System 2 (Case 2)*
Amount of packets	121862	164737	165946

Table 12: Amount of packets sent / received by each system

From this table, we can obtain easily that the fact of checking the network state adds an amount of 1209 packets when sending the previous described playlist which represents the 0,73% of the packets sent / received by System 2 (*Case 2*).

Now we are going to have a look at the total amount of bytes sent and received for each system.

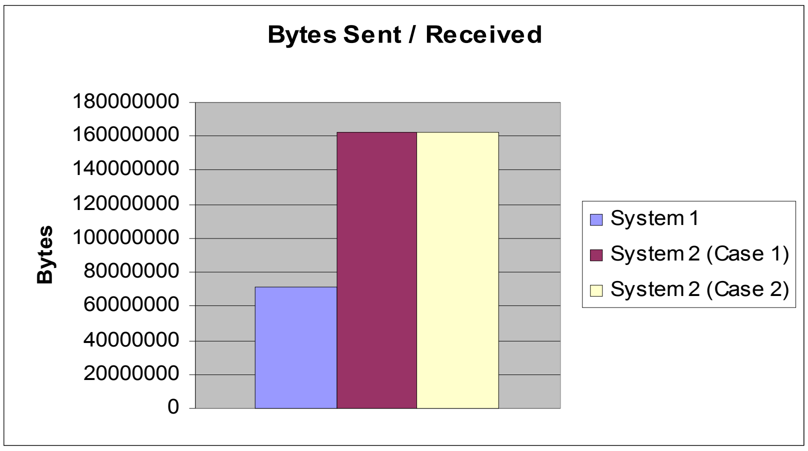


Figure 21: Bytes Sent / Received for each system

Again we find that the worst case corresponds to System 2 (*Case 2*). The following table shows the total amount of bytes transmitted for each system:

	System 1	*System 2 (Case 1)*	*System 2 (Case 2)*
Amount of bytes	71372360	162646858	162797255

Table 13: Amount of bytes sent / received by each system

From the previous table we can see that the amount of bytes added by the network state checking when transmitting the previous stated playlist is of 150397 bytes. This value represents the 0,09% of the total amount of bytes transmitted by System 2 (*Case 2*).

When considering transmission time the results are very different. In this study the worst case is System 1 as it is streaming the audio content *while* the user is listening to it. The results are shown in the following graph:

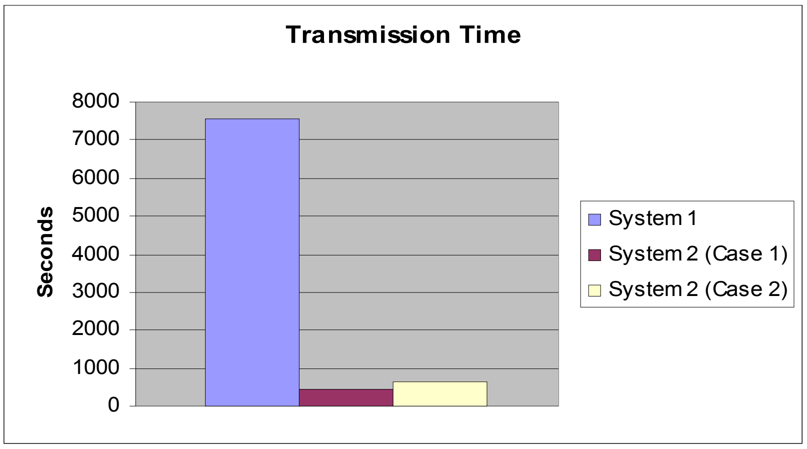


Figure 22: Transmission Time

The implication of this is that the use of System 2 requires being in a WLAN coverage area for at most 643 seconds vs System 1 which requires the user to have good network connectivity for the entire two hours.

4.2 Effect of communication errors

4.2.1 Effect of errors when using System 1

To study the effect of errors while using System 1 (remember that System 1 corresponds to the case of streaming audio), we have used the following procedure. The Pocket Streamer Server was running on a laptop and the Pocket Streamer Client was running on a PDA. Both were connected using an ad-hoc WLAN link. The IP address for the laptop was 192.168.0.4 and 192.168.0.3 in the case of the PDA.

The most extreme error that we can encounter when using this configuration is loss of the connectivity.

All the cases of lost of service will be simulated while streaming the same song in order to be able to compare effects under the same conditions. The track was “Vertigo” from “U2”. Some interesting features of this song include:

- BitRate: 372 Kbps.
- Duration: 3’16’’
- File size: 3.39 Mb

The error of loosing the network connection will be simulated by disabling the PDA’s WLAN interface for a certain amount of time (this will be described later) and it will depend on how critical is the error that we want to simulate.

The reason of choosing this way of generating the error is because we think it can simulate the most exteme error that we can have while using our mobile device (i.e., on the street). When disabling the WLAN feature the connection is completely lost. Upon enabling the WLAN interface again, first it starts the WLAN adapter and then the PDA is configured to connect automatically to the ad-hoc WLAN link that is established between the laptop computer and the PDA. We are considering the case that it is always possible to reconnect to this existing WLAN link between the laptop computer and the PDA. Were this not be possible, then the effect of this error is pretty obvious, the server will not be able to reach the client any more so the audio streaming will be stopped and this system will **not** be able to work until the connection is recovered.

We have decided to divide this study into three parts simulating a loss of connectivity of less than 10 seconds, between 10 and 15 seconds and more than 15 seconds. The results obtained are described in the following sections.

4.2.1.1 Loss of connectivity of [0, 10] seconds

To simulate this case we are going to disable the WLAN interface of the PDA and then re-enable it. For the evaluation of how much time it takes to recover the connectivity in this case we have made the following observations:

Observation Number	*Time to recover the connectivity (sec)*
1	9,31
2	9,95
3	9,57
4	9,29
5	9,18
6	9,81
7	9,47
8	9,14
9	9,59
10	9,49
Average	9,48

Table 14: Recovering connectivity after [0, 10] seconds loss of connectivity

Taking into account these data we can simulate now a loss of connectivity of [0, 10] seconds when using our system. The evaluation of its effects is shown in the following table:

Observation Number	*Pause in sound (sec)*
1	4,85
2	5,35
3	4,02
4	4,1
5	4,81
6	5,63
7	4,17
8	4,05
9	5,3
10	3,97
Average	4,63

Table 15: Pause in sound observed with loss of connectivity [0, 10] seconds

4.2.1.2 Loss of connectivity of [10, 15] seconds

To simulate this case we are going to disable the WLAN interface of the PDA and enable it again after 5 seconds. For the evaluation of how much time it takes to recover the connectivity in this case we have made the following observations:

Observation Number	*Time to recover the connectivity (sec)*
1	14,22
2	14,05
3	13,42
4	14,23
5	14,32
6	14,67
7	14,09
8	14,17
9	14,42
10	14,16
Average	14,18

Table 16: Recovering connectivity after [10, 15] seconds

Given these data we can simulate now a loss of connectivity of [10, 15] seconds when using our system. The evaluation of its effects is shown in table 17.

Observation Number	*Pause in sound (sec)*
1	8,65
2	8,44
3	8,54
4	8,59
5	*
6	*
7	8,75
8	8,82
9	*
10	8,78
Average	8,65

Table 17: Pause in sound observed with loss of connectivity [10, 15] seconds

The symbol "*" indicates that in the current observation the system was **not** able to recover and a fatal error was shown to the user. As it can be seen at the table, this case occurred in 30% of these observations. The connectivity is restablished atomatically in all other cases but in those which the system is not able to recover on its own, then the user has to restart the program to continue listening to the audio content once the connectivity is restored.

4.2.1.3 Loss of connectivity > 15 seconds

If after 15 seconds the connection is not recovered, then the link between the Server and the Client application is lost in 100% of the cases and the application has to be restarted when network connectivity is available again in order to be able to continue listening to the audio content.

4.2.2 Effect of errors when using System 2

When using System 2, the user doesn’t depend on 100% connectivity to the WLAN. In this case the user can be affected by a transmission error only when using this WLAN connection that could be:

- downloading extra content,
- creating a new audio alert, or
- using the voice interface.

4.2.2.1 Downloading extra content

System 2 offers the possibility for the user to decide to have more audio content stored locally. To be able to obtain extra content, the user has to start the Player program (if not started yet) and press the “New content” button. The user will be asked for the name of an XML file containing the new audio content. When the user enters the name of the file, the process of validating the name of the file and the context information starts. More information about this process can be found at sections 3.5 and 3.8.

As was explained in previous sections, System 2 uses File Sender (section 3.5) to transfer extra audio content to the PDA. It was also stated at section 3.5 that File Sender uses the “CeCopy” tool and this in turn uses Active Sync to copy files remotely. The main problem we encountered when dealing with File Sender was the case of loosing the WLAN connection in the middle of a transmission. The solution we found for this problem was to check the network state after each song is sent. If at that time we realize that we are *disconnected* then it is possible that the last track was not delivered properly and we will try to send it again. The “CeCopy” tool will check if the file already exists, if it was properly delivered it will not send it again.

Taking all this information into consideration, we can now answer the question of what would happen if lost WLAN connectivity while downloading extra content? If this happens we can differentiate between two cases:

- Best case: the last track was delivered properly. In this case File Sender doesn’t have to send the last track again, but it will have to wait until the connection is recovered again to continue sending.

- Worst case: the last track was almost completely delivered. Then the File Sender will have to wait until the connection is established again and then proceed to send the last track again before continuing with the rest of the tracks.

4.2.2.2 Creating a new audio alert

This case refers to the situation in which the file containing the audio alert is not able to arrive properly to the PDA due to, for example, a loss of connectivity or a problem while delivering the file.

To overcome this problem in some way, the Player program provides an alternative mechanism. When the user enters the text for a new audio alert, the Player program saves it in a .txt file at the PDA. When the time to play an audio alert arrives, the Player program will check first if the .wav file arrived, if an error occurred and the file is not present, then a message box will announce to the user that the audio alert didn't arrive due to a communication problem and it will also show the user the text from the audio alert (.txt) file stored previously. It is not as nice reading the "text alert" as listening to the "audio alert" but at least and in some way, the user *reads* the same message that it was supposed to *listen*. Of course this needs to be an audio alert telling the user to look at the screen but this can be pre-recorded.

4.2.2.3 Using the voice interface

This section and all the studies related with the effects of using the voice interface implemented for this study will be explained in detail at Johan Sverin's Masters Thesis, Context Aware and Adaptive Mobile Audio [54].

4.3 Users opinion

To perform this study we have asked 15 users (selected from our fellow students) about their preferences. First we explained to them how System 1 and System 2 work and then asked them which one would they prefer to use and why. The results obtained are shown at the following graph:

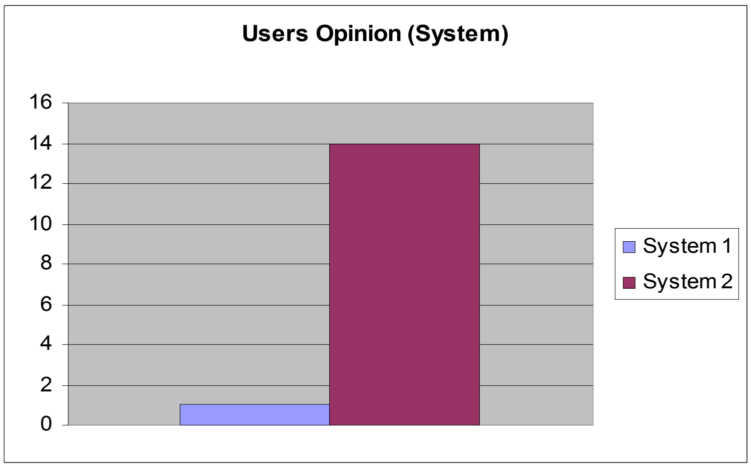


Figure 23: Users Opinion (System)

As it can be seen, the majority chose System 2 and the main reasons they gave where:

- Better to have less total storage but certain local storage.
- The effect of loosing the connection in System 1 and resulting pause in the sound can be very annoying.
- In case of System 2 you can also download additional content when the WLAN connection is available.

When asking the users for their opinion when using the voice interface, the results were more mixed as can be appreciated in the following graph:

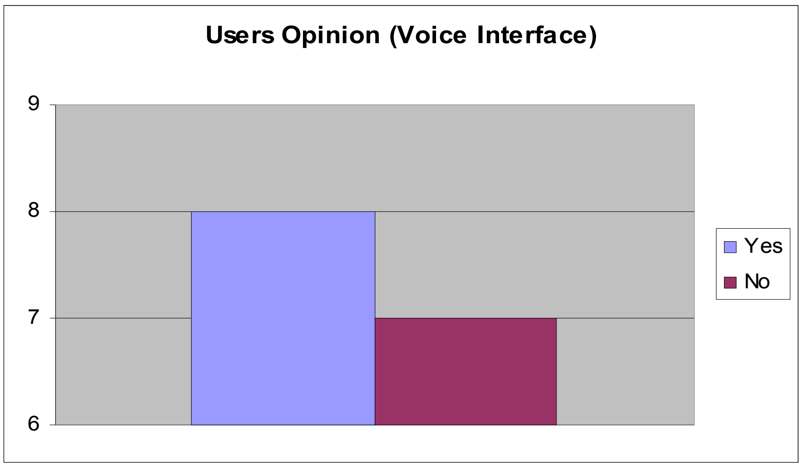


Figure 24: Users Opinion (Voice Interface)

Main reasons for those who chose Yes:

- Very useful for handicapped people.
- More comfortable than typing.
- In case of experiencing big delay, the user can always change to the typing mode.

Main reasons for those who chose No:

- No privacy, everybody can hear what you command.
- Greater delay in the command being executed.

4.4 Voice interface

This section and all the studies related with the effects of using the voice interface implemented for this study will be explained in detail at Johan Sverin's Masters Thesis, Context Aware and Adaptive Mobile Audio [54].

5 Conclusions

The main objectives of this masters thesis were:

- Compare the amount of traffic which needs to be sent in peak period via high cost network connection versus the possibility of being able to send traffic only when we have a large amount of low cost bandwidth.

- The effects of errors in the case of streaming audio versus the case in which we are caching and have cached data.

- Brief comparison, from the user's point of view, of both systems. What do users like and dislike about having cached files based on a playlist versus only streamed content.

- Regarding the voice interface, what are the advantages and disadvantages of having voice commands versus typing on the screen of the PDA.

To evaluate all these points we presented two different systems. System 1 was already built and its main feature was that that it allowed the user to stream audio constantly from the laptop computer to the PDA. Further information about this system can be found at section 2.1.2. Then we needed a system that took profit of having local storage, so we built System 2. This last system was formed of several application running at the laptop computer and at the PDA. Further information about this system and its applications can be found at section 3. Now that we had these two systems we could perform some studies to be able to answer to the previous stated points according to each system and extract some conclusions. The experiments performed and the results obtained are show in section 4.

One thing that I would like to highlight of this project is the utility founded when using context information. Our system was provided with context information such us, network state, battery state and storage state. These items resulted very useful when developing our applications in order to make them aware of the user's and device's state and act consequently with this situation. They also contributed on performing more effective transmissions and minimize the probability of transmission errors because transmissions were only advisable when network state was favourable. Further information about the context awareness used in our studies can be found at section 3.2.

6 Open issues and future work

The biggest problem we have encountered in this masters thesis was with developing the voice interface. Our idea was to perform the speech recognition process locally at the PDA but there were no speech engines accepted for our current configuration of the handheld device. There is a portable version of Sphinx [55] for Linux operating systems running at the PDA but that was not our case. The solution we encountered for that was to develop our own speech recognition application to run at the laptop computer and to build another application at the PDA to be able to capture and send the audio and feed the recognizer with it.

With this schema, of course, we are adding some delay to the performance of our system so two possible future approaches could be taken into account:

- Find the way of performing the speech recognition process locally at the PDA

- Establishing a SIP session (further information in section 2.2.1.3.2) between the program that captures the audio and the speech recognition application. With this solution we could minimize the delay and also solve all the problems we encountered with buffering and sending the audio from the PDA to the laptop.

It was highlighted before the utility that we found when using context information but maybe in future approaches some other items could be added. Perhaps some context information about the user's location could be of interest in order to take it into account and be able to *guess* in some way what could be the needs of the user in each moment and act consequently.

Regarding also the use of context information, some improvements can be done in future approaches of this thesis. In our study, we have obtained the status of the battery before starting the transmissions but not at the end of them. By checking the battery status when a transmission is finished, we could obtain the amount of battery consumed during the process of downloading audio. In order to save battery consumption, new ideas could be added to our system. As we are taking advantage of having local storage and we are not using the wireless interface 100% of the time, it could be useful to disable this interface during the periodes that we are not going to use it and enable it again when necessary.

Another thing that could be also improved is the way of sending the audio tracks from the laptop computer to the PDA. In our system we check the network state after sending each audio track and if by any chance we discover that we are disconnected, then we attempt to send it again. Before sending each track, we check if the file already exists at the PDA, if not, we will send it again. In the case that the file already exists we have two alternatives, if the file at the PDA is *identical* to the one existing at the laptop computer, then there is not need on sending the file again, but if both files differ, then we send the hall file again. This approach can be improved by dividing the files in blocks and send block by block. In this way if we discover that we are disconnected after sending one track, then we could just have a look at which is the last block that was completely delivered (by comparing the blocks existing at the track at the laptop and the ones of the track stored at the PDA) and simply send the blocks that are left to send. With this approach we can save from sending a large amount of useless data.

7 References

[1] María José Parajón Domínguez, *"Audio for Nomadic Users"*, Department of Microelectronics and Information Technology (IMIT), Royal institute of Technology (KTH), Master of Science Thesis performed at KTH, Stockholm, December 2003.

[2] Sean Wong, *"Context-Aware Support for Opportunistic Mobile Communication"*, Aberdeen, Scotland, BEng (Hons) EE, Thesis performed at KTH, Stockholm, December 2003.

[3] Andrew S. Tanenbaum, *"Computer Networks"*, 4th Edition, Prentice Hall, March 17, 2003. ISBN: 0-13-066102-3.

[4] Homepage of the project *"Vector Quantization in Speaker Recognition"*, University of Joensuu, Department of Computer Science, http://www.cs.joensuu.fi/pages/tkinnu/research/, last accessed 2004-10-19.

[5] Joseph P. Campbell, Jr., *"Speaker Recognition: A Tutorial"*, http://www.ee.columbia.edu/~patricia/papers/tutorials/tutorial.pdf, last accessed 2004-10-19.

[6] Jeff Ferguson, Brian Patterson, Jason Beres, Pierre Boutquin, and Meeta Gupta, *"C# Bible"*, Wiley Publishing, Inc. ISBN: 0-7645-4834-4.

[7] Frequently Asked Questions (FAQ), http://msdn.microsoft.com/mobility/netcf/faq/default.aspx, last accessed 2004-10-19.

[8] Introduction to Development Tools for Windows Mobile-based Pocket PCs and Smartphones, http://msdn.microsoft.com/mobility/gettingstarted/windowsmobile/default.aspx?pull=/library/en-us/dnppcgen/html/devtoolsmobileapps.asp, last accessed 2004-10-19.

[9] Visual C++, http://msdn.microsoft.com/mobility/gettingstarted/windowsmobile/default.aspx?pull=/library/en-us/dnppcgen/html/devtoolsmobileapps.asp, last accessed 2004-10-19.

[10] HP iPAQ Pocket PC h5550 - overview and features, http://h10010.www1.hp.com/wwpc/us/en/sm/WF05a/215348-64929-215381-314903-f66-322916.html, last accessed 2004-10-19.

[11] Getting Started with Visual Studio .NET and the Microsoft .NET Compact Framework, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/netcfgetstarted.asp, last accessed 2004-10-19.

[12] Frequently Asked Questions About Visual C++ .NET, http://msdn.microsoft.com/visualc/productinfo/faq/default.aspx, last accessed 2004-10-19.

[13] SDK for Windows Mobile 2003-based Pocket PCs, http://www.microsoft.com/downloads/details.aspx?FamilyID=9996b314-0364-4623-9ede-0b5fbb133652&displaylang=en, last accessed 2004-10-19.

[14] .NET Compact Framework 1.0 SP2 Redistributable (Re-release), http://www.microsoft.com/downloads/details.aspx?FamilyID=359ea6da-fc5d-41cc-ac04-7bb50a134556&displaylang=en, last accessed 2004-10-19.

[15] Windows Mobile Developer Power Toys, http://www.microsoft.com/downloads/details.aspx?familyid=74473fd6-1dcc-47aa-ab28-6a2b006edfe9&displaylang=en, last accessed 2004-10-19.

[16] Smart Badge 4, Gerald Q. Maguire Jr., http://www.it.kth.se/~maguire/badge4.html, last accessed 2004-10-18.

[17] RealOne Player, http://www.real.com/player/, last accessed 2004-10-18.

[18] Microsoft's Windows Media Player, http://www.microsoft.com/windows/windowsmedia/default.aspx, last accessed 2004-10-18.

[19] Winamp, http://www.winamp.com/, last accessed 2004-10-18.

[20] Real-Time Transport Protocol, Network Working Group Audio-Video Transport Working Group, http://www.ietf.org/rfc/rfc1889.txt?number=1889, last accessed 2004-10-18.

[21] *"Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-Guaranteed Quality of Service."* http://www.iec.org/online/tutorials/h323/topic01.html, last accessed 2004-10-18.

[22] G.711 ITU-T Standard, http://en.wikipedia.org/wiki/G.711, last accessed 2004-10-19.

[23] Florian Maurer, *"Push-2-talk Decentralized"*, Royal institute of Technology (KTH), Semester Project performed during May-August 2004.

[24] Tejaswini Hebalkar, Lee Hotraphinyo, Richard Tseng, "Voice Recognition and Identification System (Text Dependent Speaker Identification)", Final Report, 18-551 Digital Communications and Signal Processing Systems Design, Spring 2000, Group 11, http://www.ece.cmu.edu/~ee551/Final_Reports/Gr11.551.S00.pdf, last accessed 2004-10-19.

[25] Microsoft's Visual Studio .NET 2003, http://msdn.microsoft.com/vstudio/productinfo/, last accessed 2004-10-19.

[26] http://www.mor.itesm.mx/~omayora/Tutorial/tutorial.html , last accessed 2005-01-21.

[27] Microsoft Speech SDK (SAPI 5.1) Documentation. Available after installing Microsoft Speech SDK or from: http://www.microsoft.com/speech/download/sdk51/.

[28] Running ActiveSync Wirelessly, http://www.pocketpcmag.com/forum/topic.asp?TOPIC_ID=173&whichpage=6, last accessed 2005-01-30.

[29] Pocket Streamer, http://www.thecodeproject.com/netcf/PocketStreamer.asp, last accessed 2005-02-06.

[30] Windows Media Player 9 Series, http://www.microsoft.com/windows/windowsmedia/9series/player.aspx, last accessed 2005-02-06.

[31] Windows Media Encoder 9 Series, http://www.microsoft.com/windows/windowsmedia/9series/encoder/default.aspx, last accessed 2005-02-06.

[32] Windows Media Player 9 SDK, http://www.microsoft.com/downloads/details.aspx?FamilyID=e43cbe59-678a-458a-86a7-ff1716fad02f&DisplayLang=en, last accessed 2005-02-06.

[33] Windows Media Encoder 9 SDK, http://www.microsoft.com/downloads/details.aspx?FamilyID=000a16f5-d62b-4303-bb22-f0c0861be25b&DisplayLang=en, last accessed 2005-02-06.

[34] BCL Technologies, VoiceMP3, http://www.bcltechnologies.com/voice/products/voicemp3/voicemp3.htm, last accessed 2005-02-06.

[35] Introduction to Windows Media Metafiles, http://www.meetnewplayers.com/bend/Windows%20Media%20-%20Fun%20with%20asx%20files.htm, last accessed 2005-02-06.

[36] Media Playlists, http://www.phm.lu/documentation/Windows/WMP-Playlists.asp, last accessed 2005-02-06.

[37] Overview of ASX Metafiles, http://cita.rehab.uiuc.edu/mediaplayer/asx-overview.html, last accessed 2005-02-06.

[38] XML Tutorial, http://www.w3schools.com/xml/default.asp, last accessed 2005-02-06.

[39] HTML, http://www.w3.org/MarkUp/, last accessed 2005-02-06.

[40] Recording and Playing sound with the Waveform Audio Interface, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/WaveInOut.asp, last accessed 2005-02-07.

[41] Acronym Finder, http://www.acronymfinder.com/, last accessed 2005-02-08.

[42] Enabling C# Applications for Mobility, http://or1cedar.intel.com/media/training/C_appsMobility/tutorial/index.htm, last accessed 2005-02-20.

[43] Global Memory Status, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/globalmemorystatus.asp, last accessed 2005-02-20.

[44] OpenNETCF.org – Library, http://www.opennetcf.org/library/, last accessed 2005-02-20.

[45] Ethereal, http://www.ethereal.com/, last accessed, 2005-02-21.

[46] ITU Q.931, http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-Q.931, last accessed 2005-02-23.

[47] H.225, http://www.iec.org/online/tutorials/h323/topic08.html, last accessed 2005-02-23.

[48] MIME, http://www.mhonarc.org/~ehood/MIME/, last accessed 2005-02-23.

[49] The Component Object Model: A Technical Overview, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomg/html/msdn_therules.asp, last accessed 2005-02-23.

[50] IEEE 802.11, http://standards.ieee.org/getieee802/802.11.html, last accessed 2005-02-23.

[51] Microsoft's ActiveSync 3.7, http://www.cewindows.net/faqs/activesync3.7.htm, last accessed 2005-02-25.

[52] Pascal Meunier, Sofie Nystrom, Seny Kamara, Scott Yost, Kyle Alexander, Dan Noland, Jared Crane, "ActiveSync, TCP/IP and 802.11b Wireless Vulnerabilities of WinCE-based PDAs", http://www.cs.nmt.edu/~cs553/paper3.pdf, 2005-02-25.

[53] D-Link AirPlus G+ Wireless Router, http://www.dlink.com/products/resource.asp?pid=6&rid=7&sec=0, last accessed 2005-03-06.

[54] Johan Sverin, *"Context Aware and Adaptive Mobile Audio"*, Department of Microelectronics and Information Technology (IMIT), Royal institute of Technology (KTH), Master of Science Thesis performed at KTH, Stockholm.

[55] The CMU Sphinx Group Open Source Speech Recognition Engines, http://cmusphinx.sourceforge.net/html/cmusphinx.php, last accessed 2005-03-08.

IMIT/LCN 2005-06