Speech Interface for a Mobile Audio Application

JOHAN SVERIN

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LCN 2005-17

Speech Interface for a Mobile Audio Application

Johan Sverin
Royal Institute of Technology
Stockholm, Sweden

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

Advisor: Prof. Gerald Q. Maguire
Examiner: Prof. Gerald Q. Maguire

This version was last updated July 1, 2005

Department of Microelectronics and Information Technology (IMIT)
Royal Institute of Technology, Stockholm, Sweden

Abstract

Today almost everyone owns a mobile phone, adults along with teenagers and kids. Even laptops and other wearable devices such as personal digital assistants (PDA's) are become more common. We want constant connectivity to networks and the Internet, which in turn makes us more and more available.

Context-awareness will play a bigger role for these devices in the future. Aware of its surroundings, a portable device can adapt and communicate with different devices and objects, hiding complexity from the user. This enables a simpler user interface and reduces user interaction.

This master thesis builds partially upon the prior work done by Maria José Parajón Dominguez. To realize the concept of "context-awareness" HP's iPAQ Pocket PC h5500 was used together with a server/client application developed as part of this thesis project. Questions that were addressed; what are the effects on the traffic to and from the mobile device of having a personal voice interface; what are the effect on the traffic to and from the mobile device of having significant local storage; and is it possible to exchanging personal CODECs to reduce bandwidth.

With this background in mind, this thesis focuses on audio for mobile users in a quest to create more useful devices by exploiting context awareness.

Sammanfattning

Idag äger nästan alla en mobiltelefon, vuxna, tonåringar så väl som barn. Även bärbara datorer och andra bärbara apparater så som ”personal digital assistans” (PDA:er) blir vanligare. Vi strävar efter konstant dataåtkomst, vilket i sin tur gör oss mer och mer tillgängliga för andra.

”Context-awareness” kommer att spela en större roll för dessa apparater i framtiden. Medveten om sin omgivning, så kan en portabel apparat anpassa sig och kommunicera med andra, utan att göra det komplext för användaren. Detta medför ett enklare gränssnitt för användaren och minskar användarens samspel.

Detta examensarbete bygger delvis på ett tidigare arbete av Maria José Parajón Dominguez. För att realisera begreppet “context-awareness” användes HP:s iPAQ Pocket PC h5500 tillsammans med en utvecklad server/klient programvara. Frågor som man försökte besvara var; vilken effekt trafiken har till och från PDA:n vid användning av ett röstgränssnitt; vilken effekt trafiken har till och från PDA:n vid lagring av mycket lokalt utrymme; och om det är möjligt att utväxla personliga algoritmer, så kallade CODECs.

Med detta i tanke, så försöker detta examensarbete att fokusera på ljud för mobila användare i ett försök att skapa mer användbara apparater genom att utnyttja ”context-awareness”.

Acknowledgements

I would like to thank everybody who has contributed to this project, sharing their knowledge and devoting some of their time to carry out this challenging task. I would especially thank the following people:

- Professor Gerald Q. Maguire Jr., for his quick answers and great patience. For sharing his experience and encourage me in times when needed.

- My family and all friends needed to be mentioned here, those who supported me during this project. Especially my grandmother who sadly passed away in cancer this February, who always believed in me.

- And last but not least, I would like to thank with all my heart the support of my girlfriend Nina, for encouraging me in bad moments, making me feel that I was able to carry out this project.

Table of Contents

List of Figures, Tables and Acronyms

Figures

Tables

Acronyms

API	Application Programmers Interface
BGA	Ball Grid Array
CLR	Common Language Runtime
COM	Component Object Model
DLL	Dynamic Link Library
GPRS	General Packet Radio Service
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hyper-Text Mark-up Language
IETF	Internet Engineering Task Force
JIT	Just In Time
MSIL	Microsoft Intermediate Language
NAT	Network Address Translation
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal Digital Assistants
PSTN	Public Switched Telephone Network
RIFF	Resource Interchange File Format
RSSI	Received Signal Strength Indication
RTP	Real-time Transport Protocol
RTSP	Real-time Streaming Protocol
SAPI	Speech Application Programmers Interface
SGML	Standard Generalised Mark-up Language
UDP	User Datagram Protocol
USB	Universal Serial Bus
VoIP	Voice over IP
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network
XML	eXtensible Mark-up Language

1. Introduction

1.1 Overview of the Problem Area

Almost everyone owns a wearable device of some kind. It could be a regular mobile phone, a laptop or a personal digital assistant (PDA). We use them every day and they are playing a bigger part of our life than even a few years ago. People take them everywhere and use them in various environments and different situations.

However, as described in [1], none of these devices are aware of the environment that surrounds the user and none of them takes advantage of knowing the user's state, i.e. whether they are busy, available, at work or at home etc. Context-awareness can make these devices adapt depending on the environment. Local speakers, screens, or another portable device nearby could be used, without prior knowledge. From a prior interaction a user's device can learn how to handle a certain situation and act. Although, users of mobile phones and PDA's are not specialists, they require more and more advanced features, so it is important that these added functions do not compromise the ease of use of the applications.

A user has to be able to move around between different networks without losing their identity while communicating with different devices. An attractive feature is to make the connectivity transparent to the user [2]. The user has no interest of knowing when they change between GPRS, WLAN, Ethernet, etc. For this to work, these networks have to be self-configuring. Local services should be automatically detected and configured without the user needing any prior knowledge of the communication environment. This hides the complexity from the user and should lead to "better" services and simpler user interfaces.

Another important aspect to look into when developing new services is the use of a voice interface. With a textual or graphical user interface, the user is forced to focus on typing or selecting an option. This is not efficient because you loose some seconds every time. With a voice interface time can be better utilized, by giving these commands and selecting the options through a microphone. For this to work, both speaker and speech recognition have to be implemented. The first helps provide a certain level of security and the second enables interpretation of the commands dictated by the user.

The desire for constant connectivity could be useful, but also very expensive because constant connectivity will consume resources. A solution to this could try to take advantage of the local storage of the mobile device. If the local storage system could provide a certain amount of data, the connection could be lost for some time, even while the user's activities continue. It is also important to try to reduce the bandwidth used. Exchanging personal CODECs (in the extreme case exchanging voice synthesis modules) could be a solution to that.

This thesis builds on Maria José Parajón Dominguez earlier work that is presented in "Audio for Nomadic Radio" [1]. "Nomadic Radio" is defined as: "… a wearable computing platform that provides a unified audio-only interface to remote services and messages such as email, voice mail, hourly news broadcast, and personal calendar events…" [8]. The development of "Nomadic Radio" builds upon speaker and speech recognition, text-to-speech synthesis, and spatial audio. Sensors to detect the user and this environment, prioritization of incoming information, and a suitable wireless network infrastructure are also necessary.

To realize the test environment in Maria's thesis, a client/server application was designed using the UDP protocol. This application consists of a server-manager and several clients. The manager builds and modifies a list of audio content, which determines what to output to the user as audio and when it should be output. The manager maintains and manipulates this list, and it can be dynamically modified. Several enhancements, such as providing context information to the application, were proposed at the end of her thesis.

1.2 Problem Specification

This master thesis aims to address some of the problems presented above concerning context-awareness in a wearable device. By understanding how to use and exploit the audio interface of a mobile device and implementing the changes and improvements suggested in [1] we hope to explore the potentials of this emerging field.

HP's iPAQ Pocket PC H5500 series was considered a suitable platform for a possible approach to realize the concept of "context-awareness". It has integrated wireless LAN (802.11b) and Bluetooth (v1.1) giving the capability of communicating in different ways in various environments. It is small and powered by batteries, so mobility is supported. A 400 MHz Intel XScale technology based processor and relatively large amounts of memory allow rather substantial applications to run on the iPAQ.

To realize our test environment, we designed a new client/server application that could be run on the Pocket PC operating system found in the device. The application consists of a server-manager and several clients. The speech recognition client waits for audio commands and parameters from the device's input.

We studied and compared the network traffic in two different situations. On one hand, when the audio is streamed from the laptop to the PDA. On the other hand, when the audio is downloaded locally and then played later. Another point we will look at is the effect of communication errors in the two different situations. A brief view of user's opinion will also be included in the study. Finally will the thesis examine the advantages and disadvantages of having a voice interface. [This was the main focus of my work.]

2. Previous and related work

2.1 Background

Context awareness becomes more and more interesting due to smaller sizes, lower power consumption, increasing performance, and wireless communication in our mobile phones. Other wearable devices show the same development. Utilizing the context information in wearable devices and networks can produce a higher degree of intelligent. The main idea is to simplify, or even better eliminate, some of the interaction with the user, resulting in simpler user interfaces and better services.

With this background in mind, this thesis focuses on audio for mobile users in a quest to create more useful devices with context awareness. To understand the rest of this thesis some knowledge in several areas are needed, this is presented in the following sections.

2.1.1 Wearable Devices

A description given in [7] states that a device needs to be portable, enable hands-free use, possess a wide array of environmental sensors, and always be proactively acting on its user's behalf. This description is of a quite powerful and flexible device. However, this description is not sufficient. It fails as a more general description as it excludes the devices that are considered wearable today. Another description in [7] says a wearable computer is any device that offers some kind of computing, that is worn or is carried on one's person habitually, and whose primary interaction is with the person wearing, or carrying, the device. This description better fits today's laptops, mobile phones, and personal digital assistants (PDA's) and is the definition we will use in this document.

A measurement of the performance of a wearable device looks at transparency and efficacy. This thesis will not go deeper into the details of the performance of a wearable device, but will assume that our user's wearable device has constant connectivity and the details of it are hidden from the user.

2.1.2 Wireless Local Area Network (WLAN)

A wireless LAN (WLAN) is a local area network that operates by transferring data by radio or infrared transmission. It offers the possibility to maintain the connectivity while moving around and allows multiple users to share the same network. A user only requires a wireless card and authorization to use the network. The data are sent between users (and access points) with electromagnetic waves through the air. The access point interconnects the wired and wireless networks, enabling the wireless device to communicate with devices attached to the wired network.

KTH's campus in Kista, Sweden, has installed WLAN almost everywhere. The low-cost and ease of installation has lead to installation of wireless LAN systems in classrooms and other places where existing LAN ports are not already in place.

An access point has coverage radii of 150 meters for indoor and 300 meters for outdoors. Although, with specially designed antennas and the use of repeaters and other devices it is possible to enlarge the wireless cell's area. These attributes and the increasing number of

wearable devices have made WLAN very poplar. Today a wireless LAN interface is built into most laptops and PDA's, and external wireless cards are also available.

2.1.3 Voice over IP (VoIP)

Voice over IP (VoIP) is based on a set of protocols to carrying voice information and call setup over the IP network. This means sending the voice information in digital form in packets rather than in the traditional circuit-switched protocols of the public switched telephone network (PSTN).

Today VoIP is a growing market and will probably replace the old phone system in the near future. Very few offices and even fever homes have a pure VoIP infrastructure, but telecommunication providers routinely use VoIP.

The Real-time Transport Protocol (RTP) is used to transport the traffic through the network. It defines a standardized packet format for delivering audio and video over the Internet (see RFC-1889 [16]). Originally, RTP was designed as a multicast protocol, but has since been applied to many unicast applications.

For signaling there are several alternative protocols. The most widely used ones are H.323 and SIP. H.323 defines protocols to provide audio-visual communication sessions on any packet networks [17]. However, Session Initiation Protocol (SIP) has gained popularity. Although many other VoIP signaling protocols exist, its roots in the IP community rather than the telecom industry characterize SIP. Unlike H.323, which is a complete protocol suite, SIP is a single protocol, but it has been designed to interwork well with existing Internet applications.

SIP is a proposed standard from the Internet Engineering Task Force (IETF) to setup a session between one or more clients [18]. SIP can establish two-party sessions (ordinary telephone calls), 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 or whiteboard applications. Media can also be added to (and removed from) an existing session. SIP handles only setup, management, and termination of sessions. Other protocols, such as RTP/RTCP (described above), are used for data transport. SIP is an application-layer protocol and can run over UDP or TCP.

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 to allow SIP to interwork with existing Internet applications. The six methods defined by the core specification are:

Table 1, SIP methods

Method	Description
INVITE	Request initiation of a session
ACK	Confirm that a session have been initiated
BYE	Request termination of a session
OPTIONS	Query a host about its capabilities

CANCEL	Cancel a pending request
REGISTER	Inform a redirection server about the user's current location

For more information about VoIP and specifically SIP, have a look at Carlos Marco Arranz thesis [41].

2.1.4 Connectionless Transport: UDP

The User Datagram Protocol (UDP) is a simple connectionless protocol. It provides a procedure for applications to send messages to other program with a minimum of protocol mechanism [15]. In contrast to TCP, UDP does not require any connection, nor does it guarantee any delivery or have duplicate protection. An application that uses UDP must deal directly with end-to-end communication problems such as retransmission for reliable delivery, packetization and reassembly, flow control, and congestion avoidance.

UDP operates as a transport protocol as follows. After receiving a message from the application process, the source and destination port number fields are attached for the multiplexing and demultiplexing. The resulting segment is sent to the network layer where it is encapsulated in an IP datagram. The packet is set to the receiving host with hope for delivery. Upon delivery, the receiving host uses the port numbers and the IP source and destination addresses to deliver the data in the packet to the right application process.

Although one might consider that the transport control protocol (TCP) is always preferable to UDP since it provides reliable data transfer, there are many applications that are better suited for UDP. Developers often use UDP in applications where the speed and performance requirements outweigh the reliability, for example, video streaming [20].

A server using UDP can support many more active clients than if the same application where run over TCP [1]. This is possible because UDP does not maintain connection state and hence does not track parameters such as receive and send buffer occupancy, congestion control parameters, or sequence and acknowledgement numbers. UDP also features lower head than TCP allowing more useful information to be transmitted over a given link.

One disadvantage is that UDP does not mix well with network address translation (NAT), since incoming UDP traffic will usually be rejected. TCP traffic on the other hand can return as long as the application on the inside of the NAT that created the connection.

To realize the advantages stated above, the transport protocol to be used in this master thesis is UDP. It will be used for communication between the server and several clients.

2.1.5 Playlists

A playlist can be described as a metafile that contains the required information for playing a set of pre-selected tracks. Different players use different formats. Some examples of possible formats are: .asx, .m3u, .wvx, .wmx and the most generic one, .xml. We say "the most generic one" because most of the possible extensions used for building playlists, are, as stated above, proprietary ones or dependent on the 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 the necessary player.

2.1.6 XML – eXtensible Mark-up Language

eXtensible Mark-up Language (XML) is a mark-up language for documents containing structured documents [9]. Almost all documents have some structure. Structured information contains both content and some indication of what role that content plays. A mark-up language is a way to identify structures in a document. The specification of XML defines a standard to a mark-up to documents.

XML differs from HTML. In HTML all the tag semantics and tag set are fixed. While the World Wide Web Consortium (W3C) and the WWW community constantly try to extend the definition of HTML, it is unlikely that browser vendors have implemented all the extensions. Therefore, there is often a delay and some differences between the specifications and implementations. In contrast, XML specifies neither semantics nor a tag set. Since there is not a predefined tag set, there cannot be preconceived semantics. All semantics of an XML document will either be defined by the application that process them or by stylesheets [9].

XML is defined as an application profile of SGML. SGML is the Standard Generalized Mark-up Language defined by ISO 8879 [10]. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade, but it is not well suited to serving documents over the web. XML is a restricted form of SGML.

XML was created so that richly structured documents could be used over the web. Some of the goals are that it should be straightforward to use over the Internet, it should support a wide variety of applications, it should be compatible with SGML, it should be easy to write programs that process XML documents. For more information about the goals, see [9].

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 database, and generic applications can be used to display the data.

2.1.7 Speech and Speaker Recognition

Lately speech and speaker recognition have moved from concept to common for the telecommunications industry. The growing market for mobiles and handheld devices has led to a need for new services with a simpler user interface by exploiting speech and speaker recognition. One might think that the two are the same, but there is an important difference between them.

Speech recognition is the process by which a computer maps a speech signal to text. Speech recognition is often used as commands in applications. A user can go through a menu or tell the application to execute a command, which normally have to be done with a mouse, keyboard or other manual interaction.

Speaker recognition on the other hand, is the process by which a computer identifies and verifies who is speaking on the basis of individual information included in speech waves. This technique makes it possible to use the speaker's voice to verify their identity and control

access to services. It can also be used to select a specific user profile based on who is speaking.

2.1.7.1 Speaker recognition

Four modules compose a speaker recognition system.

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

The speaker modeling performs a reduction of feature data by modeling (typically clustering) the distributions of the feature vectors, while the speaker database stores the speaker models.

Decision logic is a module that makes the final decision about the identity of the speaker by comparing unknown feature vectors to all models in the database and selecting the best matching model.

2.1.7.2 Speech Recognition

Speech recognition refers to the process of translating spoken phrases into their equivalent text strings. Approximation of this process is described in the following figure:

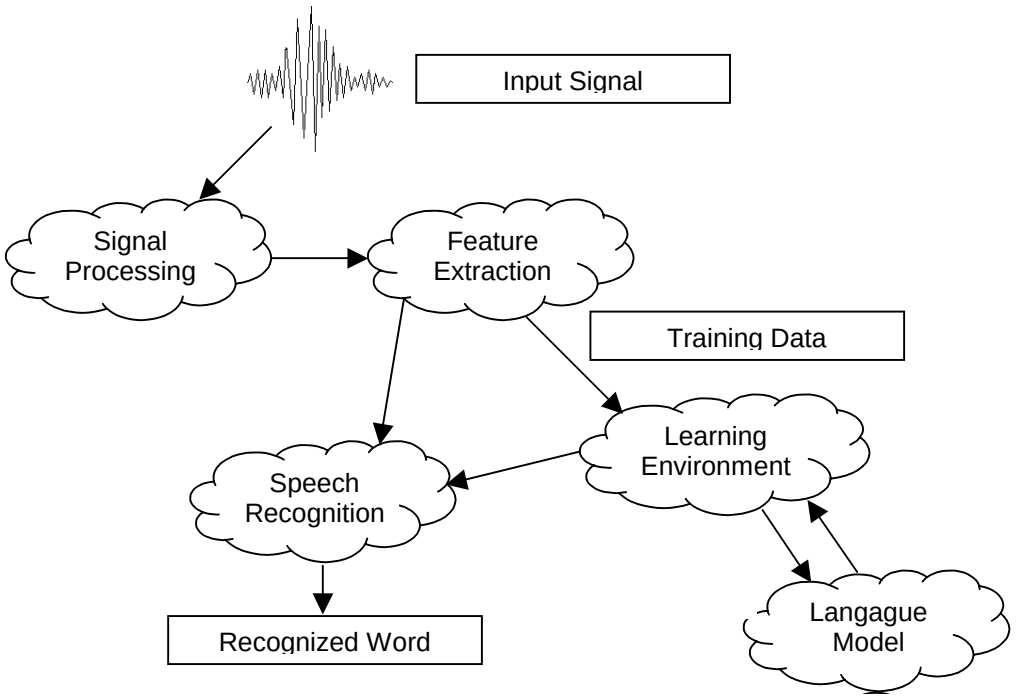


Figure 1. Speech recognition modules

Having this scheme in mind, the process can be described in detail.

1. Preparing the signal for processing
   After the microphone has captured the audio, the first step is to prepare the signal for recognition process. The most important steps is to detect the presence of speech in the signal, thus discarding those parts that of the signal corresponding to silences. This allows the recognition system to differentiate the words that are spoken.

2. Signal modeling
   This step consist consists of representing the spoken signal into its equivalent sequence of bits and extracting parameters from it that will be useful for subsequent statistical analysis.

3. Vector quantizations
   Vector Quantization (VQ) is the process where a continuous signal is approximated by a digital representation (quantization) considering a set of parameters to model a complete data pattern (vector).

4. Phone estimations
   A phone is the acoustical representation of a phoneme. In this sense, 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 high-level recognizers (word recognizers). In this phase, the distance between trained vectors and test frames is obtained to produce a pattern-matching hypothesis.

5. Word recognition
   Word recognition is the last step in the process and now the most probable word obtained during all the process is returned as output.

2.1.8 Microsoft's Speech SDK 5.1 (SAPI 5.1)

Microsoft's Speech SDK (SAPI 5.1) provides a high-level interface between the application we want to build and the speech engines. 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; those are text-to-speech (TTS) systems and speech recognizers (SR). 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.

2.1.8.1 API for Text-To-Speech

The component responsible for controlling text-to-speech is the ISpVoice Component Object Model (COM) interface. With a call to ISpVoice.Speak, speech is easy generated from some text data. The interface also provides several methods to change the voice and synthesis properties, such as speaking rate (ISpVoice.SetRate), output volume (ISpVoice.SetVolume), and changing the current speaker voice (ISpVoice.SetVoice).

2.1.8.2 API for Speech Recognition

The equivalent to ISpVoice in the speech recognition engine is the ISpRecoContext interface.

A recognizer can be created in two ways. The application can create an in-process (InProc) ISpRecognizer object. In this case, SAPI will create the SR engine COM object from the object token representing an engine. Alternatively, an application can create a shared recognizer. In this case, SAPI will create the SR engine in a separate process (named sapisvr.exe) and all applications will share this recognizer.

After an ISpRecognizer and an ISpRecoContext has been created its time to setup the audio input stream. Once the input for the recognizer is set, the next step is to define the events that are of interest. The recognizer can subscribe to many different events but the most important is "RECOGNITION". This event will be raised each time that recognition takes place and in its event handler the code that should be executed is placed. Finally, a speech application must create, load and activate an ISpRecoGrammar, which essentially indicates what types of utterances to recognize, i.e. dictation or command and control grammar.

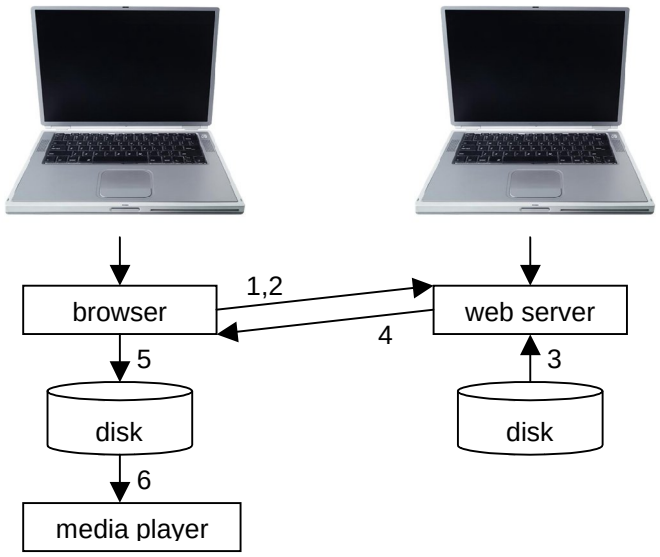
A shared recognizer is recommended for most speech applications, mainly those with a microphone as input [44]. For large server applications that would run alone on a system, and for which performance is key, an in-process speech recognition engine is more appropriate.

The implemented voice interface, used in this thesis, uses an in-process recognizer.

2.1.9 Streaming Audio

Streaming audio has become a very widely used way to listen to music over Internet. People use it without knowing how it really works. When you click a song on a web page the computer has to establish a TCP connection to the web server where the song is stored. Then it sends a HTTP GET request to request the song. The song, which might be encoded as mp3 or another format, is fetched from the server and sent back to the requesting computer. If the file is bigger than the server's memory, it can be fetched and sent in blocks.

There are many different audio players on the market, such as RealOne Player [30], Microsoft's Windows Media Player [31], or Winamp [32]. These players are associated with different types of files, even the same types. These applications are called helper applications, because it is a helper to the browser. 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 scratch file on the disk. Then it will start the media player and pass it the name of the scratch file. Last, the media player fetches the content and plays the music, block by block.



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

Figure 2. Process of obtaining an audio file from Internet

This approach is not good if you have a slow connection, such as 56 kbps, especially when the song file is over 4 MB (a typical file size of and mp3 encoded song). Since the song can only played when the entire file is downloaded, it would take approximately ten minutes before the song started.

To overcome this problem a new scheme has been developed. The link from the page is now not actually a link to the audio file, but a link to a metafile, a very short file that simply identifies the music. A typical metafile might only be one line of an ASCII text and look like:

rtsp://my-audio-server/song-003.mp3

When the browser gets this one line file, it writes it to a scratch file and starts the media player that is used as a helper. When the audio player reads the scratch file and discover that it contain a URL it contacts the server and request the content to be streamed directly to the player, without the involvement of the browser.

In most cases, the server in the metafile is not the same as the web server. In fact, it is not generally a HTTP server, but rather it is a specialized media server. In the example above the protocol used to stream the audio is the Real-time Streaming Protocol (RTSP) [42].

A media player has four major tasks. The first is to provide a user interface; the second to handle error transmissions; the third to decompress and decode the audio; and the fourth to eliminate (or at least try to) jitter.

As noted, the second task is to deal with error transmissions. Real-time application rarely uses TCP as transport protocol. Because a TCP connection utilizes retransmissions, error could cause a long delay. The actual transmission is usually done with a protocol such as RTP, see section 2.1.3. As with many real-time protocols, RTP is layered over UDP, so packets may be lost, but it is up to the *player* to deal with it.

All real-time systems want to eliminate jitter or hide. A solution is to buffer the audio, normally 10 to 15 seconds worth before starting the audio playout.

2.1.10 Wave Format

The WAVE file format is a subset of Microsoft's Resource Interchange File Format (RIFF) specification [24]. A WAVE file is often just a RIFF file with a single "WAVE" chunk, which consists of two sub-chunks. The first sub-chunk is "fmt" that specifies the data format, while the other one is the "data" chunk that contains the actual data samples.

The default byte ordering assumed for WAVE data files is little-endian. Files written using the big-endian byte-ordering scheme have the identifier RIFX instead of RIFF.

2.1.11 HP iPAQ Pocket PC h5500 Series

HP's iPAQ Pocket PC h5500 series is a powerful and flexible handheld device. It has integrated Bluetooth and WLAN 801.11b that allows one to access Internet, email, and corporate data either via an access point or indirectly via a cellular phone. It supports security solutions such as unique biometric fingerprints, virtual private networks (VPN), and 64-bit and 128-bit wired equivalent privacy (WEP) for the WLAN interface. It also includes a removable battery, transflective display, integrated Secure Digital slot, increased memory and Microsoft Windows Pocket PC 2003 Premium Edition.

Table 2. HP iPAQ Pocket PC h5500 series specifications [43]

Integrated wireless	Bluetooth v1.1 and WLAN 802.11b
Operating System	Microsoft Windows Pocket PC 2003 Premium
Processor	400-MHz Intel XScale Technology-based processor
Display	16-bit touch-sensitive TFT liquid crystal display (LCD), 64K color Viewable image size: 96 mm diagonal
Memory	48-MB Flash ROM, 128-MB SDRAM
Dimensions	138 x 84 x 15.9 mm (H x W x D)
Weight	206.8 g

We considered that this iPAQ suited our needs for a possible approach to the implementation of the software and trials that we have planned.

2.1.12 .NET Framework

The .NET Framework is made up of four parts: the Common Language Runtime (CLR), 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 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 on mobile devices as well. Last, the framework was designed to provide a single set of libraries that would work with multiple languages.

One of the most important features of the .NET Framework is the portability of the code generated. Using Visual Studio .NET, the code that is output by the compiler is written in Microsoft Intermediate Language (MSIL). MSIL is made up of a specific set of instructions that specify how the code should be executed. MSIL is not a specific instruction set for a

physical CPU. The “just-in-time” (JIT) compilation translates the MSIL code into CPU specific machine code.

We decided to implement our application with the .NET Framework and C# programming language. To be able to implement our application on the Pocket PC we had to use the Compact .NET Framework, that is a smaller version designed for mobile devices.

2.1.13 Windows Mobile Developer Power Toys

Windows Mobile Developer Power Toys [34] are a set of tools whose main purpose is to allow the developer to test the mobile applications that are being built. The most relevant ones are:

- **CeCopy**: a small application that copies files from a stationary computer or laptop to a wearable device such as a PDA.

  Usage: CeCopy [options] <Source_FileSpec> <Destination>

- **CmdShell**: a shell on the wearable device for executing commands.

2.2 Related work

Since the foundation for context-awareness and modern handheld devices was laid in the end of 1980s [6], a rapid evolution in the research has taken place. There is lots of information to find about the subject thanks to different project groups. Here are some short summaries of related work.

2.2.1 Audio for Nomadic Audio

Audio for Nomadic Audio is a master thesis done by María José Parajón Domínguez in 2003 [1]. The aim of her thesis was to solve the problems of having multiple wearable devices by introducing a new one, capable of combine all of them and offering an audio interface.

SmartBadge 4 was used as a wearable device and a laptop was needed to complete the system. Both were running Linux operative system. To test the environment she developed a client/server application using the UDP protocol and C language.

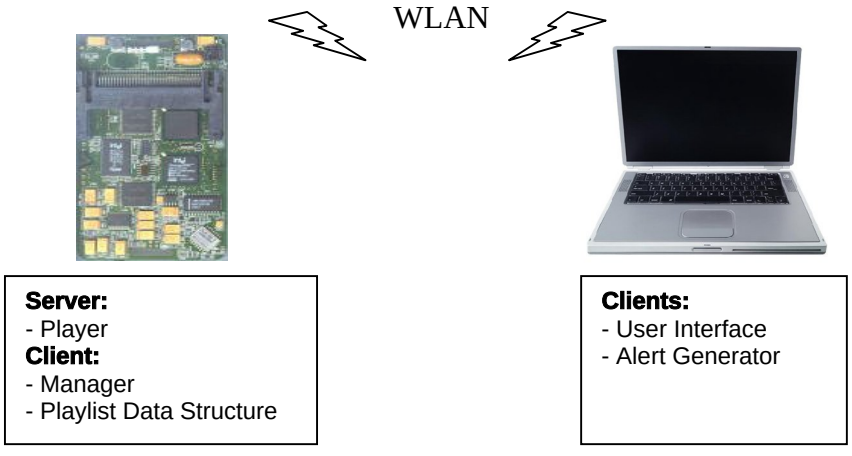


Figure 3. System used in Audio for Nomadic Audio

The following components were developed:

- ***Manager:*** represents the server part of the application. It is running on the SmartBadge and its main function is to create and maintain a playlist by processing the client requests.

- ***Player:*** this client is also running on the SmartBadge. Its main function is to ask to the Manager for the first element of the playlist and invoke a suitable player to reproduce the content of this element.

- ***User Interface:*** in this case, this client is running on the laptop and its main functionality is to accept commands from the user and transmit them to the Manager.

- ***Alert Generator:*** this client is also running on the laptop and accepts text input transforming it into audio alerts. María José Parajón used and modified a client developed before by Sean Wong [19].

2.2.2 SmartBadge 4

This is fourth version of SmartBadge, a prototype for future smart cards. It has been developed at Hewlett-Packard Laboratories together with researchers at the Royal Institute of Technology (KTH). Running Linux, this version of the badge was operational on February 2001. This version is a 12 layer printed circuit board with ball grid array (BGA) mounted SA1110 processor and SA1111 companion chip [11].

The SmartBadge is equipped with several sensors such as a 3-axis accelerator, temperature sensors, humidity sensors, and light level sensors. It also supports infrared, PCMCIA, USB, and compact flash interfaces. This gives the badge a wide diversity in its connectivity and communication.

2.2.3 Active Badge

Active Badge is used to locate a person in a building. The device repeatedly transmits a unique infrared signal every ten seconds, identifying it. Networked sensors installed within offices and rooms in the building then receive the signal. The sensors provide the system with information about the location of the badges [12].

2.2.4 Festival-Lite

Festival-Lite, also known as FLite, is a small, fast run-time synthesis engine developed at Carnegie Mellon University (CMU). It has mainly been designed to fit small embedded machines like PDA's, as well as to large servers. FLite is written in ANSI C and offers text to speech synthesis in a small and efficient binary. The engine is very portable and can be used on most platforms. The synthesis library can be linked into other programs and includes two simple voices, a small diphone voice along with a limited domain voice [13]. The result of the text to speech synthesis is an ordinary wave file that can be played in an audio player.

2.2.5 MyCampus

MyCampus is an agent-based environment for context-aware mobile services, developed at Carnegie Mellon University (CMU). A user access personalized context-aware agents from their PDA's over the campus's wireless LAN. The different agents can suggest different

restaurants based on the user's location, schedule and expected weather. MyCampus users can download new task-specific agents to the PDA in order to access the services they are interested in [14].

2.2.6 Pocket Streamer

Pocket Streamer [25] is a small application written in C# that allows you to browse a music library on a desktop from your PDA. It allows you to select an artist, album, and track or radio station. The music is streamed from your computer over the network and is played in your PDA. This application helped us to compare two ways of playing audio. It streams audio managed on the laptop while our solution uses local storage on the PDA. It consists of two parts, one client used on the PDA and a server, that manage the music library, on the desktop. The application uses Windows Media Player and Encoder 9 [26], [27], [28], [29].

When you start the server it will appear as a system tray icon on your desktop. From the client on the PDA the user can obtain the content in the Media Library on the desktop. When the play button is pressed a broadcast session is setup and the audio is streamed to the PDA.

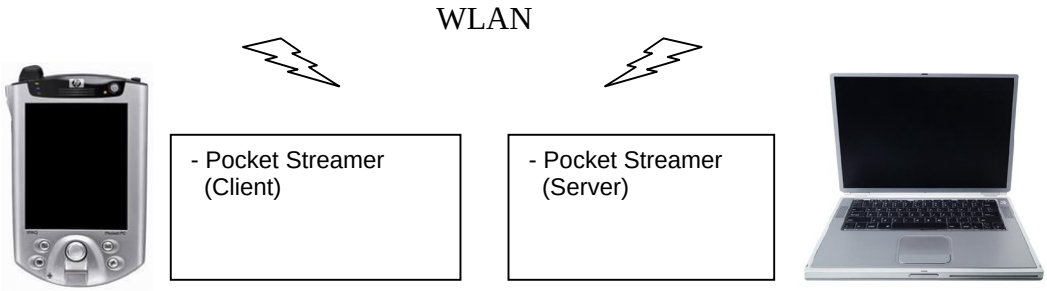


Figure 4. Pocket Streamer

2.2.7 Microsoft Portrait

Microsoft Portrait is a research prototype for mobile video communication [33]. It supports .NET Messenger Service, Session Initiation Protocol (SIP), and Internet Locator Service on PC's, Pocket PC's, Handheld PC's, and Smartphones. It runs on local area networks, dialup networks, and even wireless networks with bandwidths as low as 9.6 kilobits/second. Microsoft Portrait delivers portrait-like video if users are connected with low bandwidth and displays full-color video if users have a broadband connection.

If you do not have a camera, you can still see others who do send video, or talk with others via a robust voice codec working at as low as 2.4 Kbps bandwidth.

2.3 Prerequisites

In order to fully understand this thesis the reader needs to have some previous knowledge and understand the basic concepts and fundamentals of data and computer communication, including wireless communication (specifically Wireless Local Area Network (WLAN)), and the principles and functions of communication protocols.

3. Design

3.1 Overview

Our system is utilizes for two platforms: Microsoft® Windows® Pocket PC 2003 Premium is used on the PDA, while the desktop is running Microsoft® Windows® 2000. Microsoft's Active Sync 3.7 was also installed on the desktop.

Below is a graphical overview of the system; detailed descriptions can be found in the following sections.

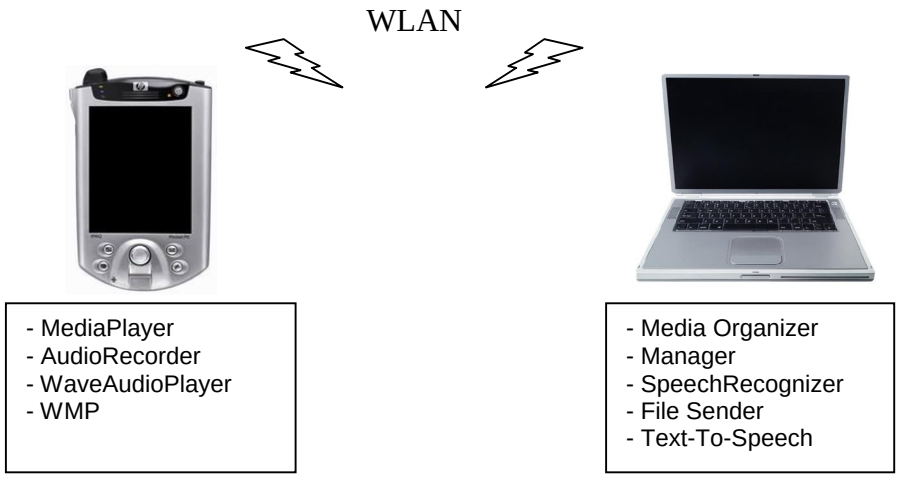


Figure 5. Design Overview of our system

The system consists of many different small applications that work together.

On the PDA the main application is the MediaPlayer, which handles playlists and invokes a media player. Mp3 and wmv files are played in the background by Windows Media Player, while regular wave files uses the WaveAudioPlayer. The AudioRecorder is the basis for the voice interface on the PDA.

The most imported application on the laptop is the Manager that handles all messaging between the different applications. The FileSender transfer playlists and files to the PDA. TextToSpeech converts alerts, i.e., textual messages, requests into wave files and transfers them to the PDA, while the SpeechRecognizer receives a real-time audio stream from the AudioRecorder. When a command is recognized a message is sent via the Manager to the MediaPlayer.

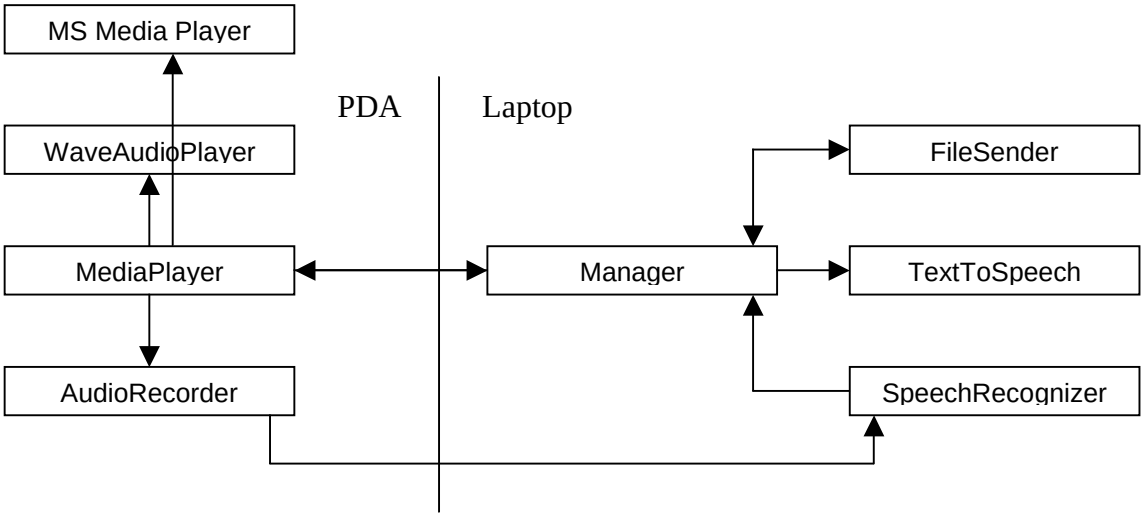


Figure 6. Flow of execution of the system

Table 3. Available commands in the system

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

3.2 Methodology

In the first part of our study we compared Pocket Streamer, which streams the audio from the laptop to the PDA, with our developed application that stores the music locally and only use the network when needed [38]. We did the comparison with respect to the following:

- 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 available.

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

- Briefly compare, from the user's point of view, 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?

In the following sections two example scenarios are described. From this moment "System 1" will refer to the Pocket Streamer, where audio content is streamed from the laptop; while

"System 2" will refer to our application, where audio is stored locally on the PDA. These are the same scenarios that Inmaculada Rangel Vacas uses in her thesis [38].

3.2.1 Scenario for System 1

Eva loves listening to music. As a present she received 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 downloads and installs 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 favorite songs and adds them to the Media Library and closes Windows Media Player.

Before she leaves to visit her friend Susana, she decides to take her new PDA so she could to listen to music on the way to Susana's house. She starts the Pocket Streamer Server on the laptop and the 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 favorite songs.

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

3.2.2 Scenario for 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. After downloading and installing the application she has some new applications: MediaPlayer, WaveAudioPlayer, and AudioRecorder on the PDA; and SpeechRecognizer, Media Organizer, TextToSpeech, and FileSender on the laptop.

Following the instructions for this new system, she decides to start the Media Organizer and select her favorite 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 later. She starts the MediaPlayer on the PDA and chose new content. She writes the name of the playlist and press OK.

A message is sent to the manager who sends a request to the FileSender about the file. FileSender finds the file, verifies that it exists and that it is in the valid format and starts sending the audio content to the PDA while Eva is studying. This audio content will be stored in a new 512Mb 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 that is 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 records 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”. 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 on the screen the “Next” button.

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”, a dialog is opened asking for a playlist. She selects one and request that information about this file is sent to the application Manager at the laptop. A response is sent back to the PDA and processed according to the current context information. As the current conditions are favorable for the transmission (she is in a WLAN hotspot), the transmission starts and when it will be finished, a message box will tell Eva that her additional tunes are ready to be used. In the mean time, she continues to listen to the local audio content.

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

3.3 Implementation

In the following section a deeper look at the implementation is presented.

3.3.1 Playlist Representation

A playlist could be considered as a metafile, containing information about a set of audio content to be played at some later time. There are also several formats for a playlist.

To represent a playlist in our system we use XML. We chose a XML playlist because we didn’t wanted it to be restricted to our player. Another player could easily substitute it.

The elements and attributes we use in our representation are:

- playListBase / playListBaseID : the full name and location of the XML file
- playListAuthor / playListAuthorID : the author of the playlist.
- track
  - title (titleID): .....the title of the track.
  - author (authored): .....the group or soloist 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).

A possible example of a playlist is shown below. However, only one track has been added in order to simplify the example.

```
<?xml version="1.0" encoding="utf-8" ?>
<playList>
  <playListBase playListBaseID="D:\Music\playlist.xml" />
  <playListAuthor playListAuthorID="Johan Sverin" />
  <track>
    <title titleID="Vertigo" />
    <author authorID="U2" />
    <bitRate bitRateID="372,76" />
    <duration durationID="3,28" />
    <fileSize fileSizeID="3,39" />
    <fileType fileTypeID="mp3" />
    <sourceURL sourceURLID="D:\Music\U2 - Vertigo.mp3" />
    <sourcePDA sourcePDAID="\Storage Card" />
    <fileName fileNameID="U2 - Vertigo.mp3" />
  </track>
 </playList>
```

Figure 7. Playlist example

3.3.2 AudioRecorder

The recorder was built using Microsoft's Visual Studio .NET 2003 as a development environment, C# as the programming language, and uses Platform Invoke (P/Invoke) to access required external functions. P/Invoke allows managed code to invoke unmanaged functions residing in Dynamic Link Libraries (DLL's) [22]. The recorder builds on the recorder in the Smart Device Framework from OpenNETCF [23], but modifications have been made to fit our needs. The biggest change was to enable the recorder to record constantly and not simply for a short period of time.

Note that the AudioRecorder was needed as there were no available speech recognizers that would run on the iPAQ under the Pocket PC operating system. Thus we chose to split the functionality between the PDA and a laptop.

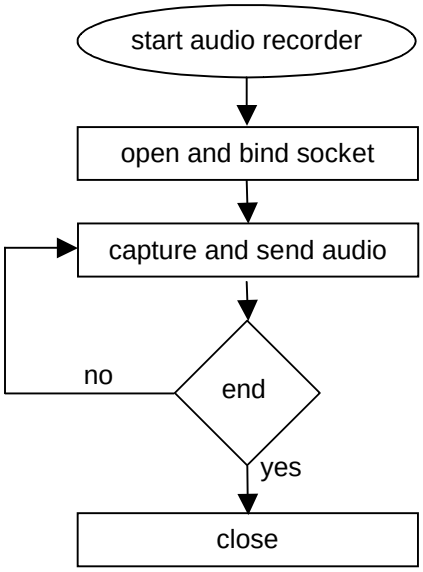


Figure 8. Flowchart of Audio Recorder

The audio recorder has been created to enable a remote voice interface for the speech recognition based application. As the flowchart shows, the AudioRecorder opens and binds a socket for communication with the SpeechRecognizer at the laptop when it starts.

The main feature is to constantly record audio at the PDA and send this real-time stream to the laptop. A Pocket PC window message is received when each audio buffer is full; the buffer is stored into a byte array before it is emptied and reused again. The data stored into the byte array is then put into a RTP packet and sent to the remote computer.

Using different audio codec's, such as GSM [48], or lower audio quality could reduce the network traffic. However, it is much harder to recognize speech after it has been encoded since lots of information has been thrown away. Hence using a codec such as the GSM codec did **not** suit the project.

3.3.2.1 Silence Detection

At first, silence detection was thought to be a means to reduce the amount of bandwidth needed; i.e. only feeding the recognizer with the necessary audio data.

However, performing silence detection locally or on the remote side makes the word selection harder for the recognizer, because all necessary audio may **not** be available; due to clipping out inter word silence. So a decision was made not to use silence detection, but rather feeding the recognizer with the complete real-time audio stream.

3.3.3 MediaPlayer

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.

A screen capture of the program at the start of its execution is shown below.

Figure 9. Screen capture of MediaPlayer

The MediaPlayer has been extended since Inmaculada Rangel Vacas thesis. A new button has been added to enable or disable the speech recognition.

Upon start, this button is disabled. It is enabled as soon as a playlist has been loaded into the application. When this button is pressed, a message is sent to the remote computer, to start the SpeechRecognizer. Then the AudioRecorder is started in the background to send real-time audio to the remote SpeechRecognizer for analysis. The recognizer in turn sends the recognized commands to the Manager.

3.3.4 WaveAudioPlayer

This application is a simple console application built using Microsoft Visual Studio .NET 2003 as environment and C# as development language.

The application is running on the PDA and its main purpose is to play a wave file. It receives as input a string containing the file name (including the full path to the file). The application checks if the file exists and if it is valid and then starts the process of playing the file. It uses Microsoft's Waveform Audio interface [35] to do that.

3.3.5 FileSender

This program is a simple console application built using Microsoft Visual Studio .NET 2003 as environment and C# as development language. The FileSender is responsible for sending all the audio tracks contained in the playlist.

After validity checks of the file, the application starts a new process of the tool "CeCopy" that doing the actual copying of the files. Each audio track in the xml playlist is read and given to CeCopy that copies the file to the PDA. Before sending the file, the state of the network is examined. If the link quality or the Received Signal Strength Indication (RSSI) drops below a certain threshold (here 50), a timeout occurs. After ten seconds the application checks the RSSI again. If the parameters have changed to favorable, the transmission will continue.

3.3.6 SpeechRecognizer

This application was built using C# under the environment of Microsoft's Visual Studio .NET 2003. This program runs on the laptop. A flowchart of the Speech Recognition program is shown at figure 10.

The speech recognizer builds upon Microsoft's Speech SDK, SAPI 5.1. Here we utilize the speech recognizer (SR) engines.

Microsoft's speech recognition engine supports context free grammars, these allows us to specify a command list that it recognizes from. This makes it easier for the engine to determine what word to translate into. Alternatively in the case of dictation, the engine has to look up the potential word in a large vocabulary. The recognizer has also been trained to my profile; collecting necessary data to build up an internal data model of my voice, see section 3.3.6.1. This makes it even easier for the engine to make a correct recognition decision.

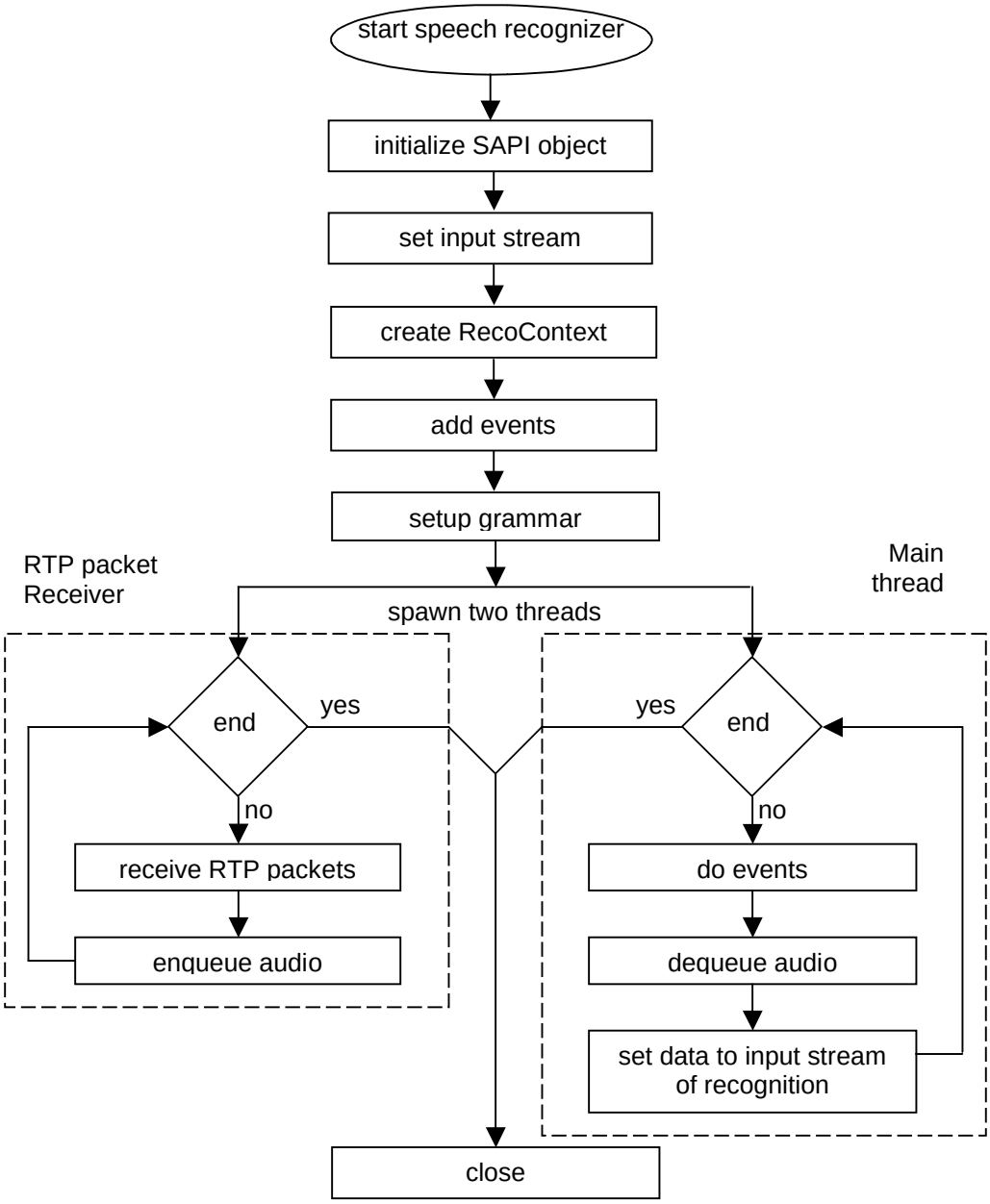


Figure 10. Flowchart of SpeechRecognizer

The SAPI provides a Recognizer interface called *ISpRecognizer*, which provides the application with different functions to control the properties of the ASR engine. Each *ISpRecognizer* represents a single speech engine. In the initialization phase *ISpRecognizer* is used to setup the input stream.

The main interface to the application is the Recognition Context *(ISpRecoContext).* The application informs the Recognition Context about all the events it is interested in. In our implementation the events are: SPEI_RECOGNITION (the recognized event) and SPEI_SR_END_STREAM (which indicates the end of a stream).

The grammar file is loaded from an xml file containing all the commands we intended to use in the application.

According to [37], the preferred wave file format for Microsoft's ASR engine is 22 kHz 16 Bit mono, and tests have showed that the best confidence is given using this format. Sample rates ranging from 11 kHz to 44.1 kHz have been tested as well as 8 Bit and 16 Bit samples (see section 4.4.1).

The audio is received in a separate thread and the data are extracted from the RTP packets and put into a queue. In the main thread the recognizer works on the queue doing recognition. The engine stops and waits if there is congestion in the network. As soon as there are at least two packets, each 1440 bytes of audio data, in the input queue it proceeds.

When a command from the grammar is recognized with sufficient confidence a message is sent to the MediaPlayer indicating the command.

3.3.6.1 Voice Training

To get better results from the speech recognizer it has to be trained to your voice pattern and pitch. Training consists of reading texts shown on your screen into your microphone. As more text is read, the speech input engine learns more about your particular voice. Five hours of training is recommended on average [40]. It will also work with minimal training, but more training improves the accuracy. A quality headset with noise reduction improves the results as well. The results may also vary from person to person because some people speak very clearly with a consistent voice whereas some people speak in variable tones and at times even mumble.

Use the attribute "–VoiceTraining" when you start SpeechRecognizer, to be able to train and create a profile of your own voice. An example is shown below.

"SpeechRecognizer.exe –VoiceTraining"

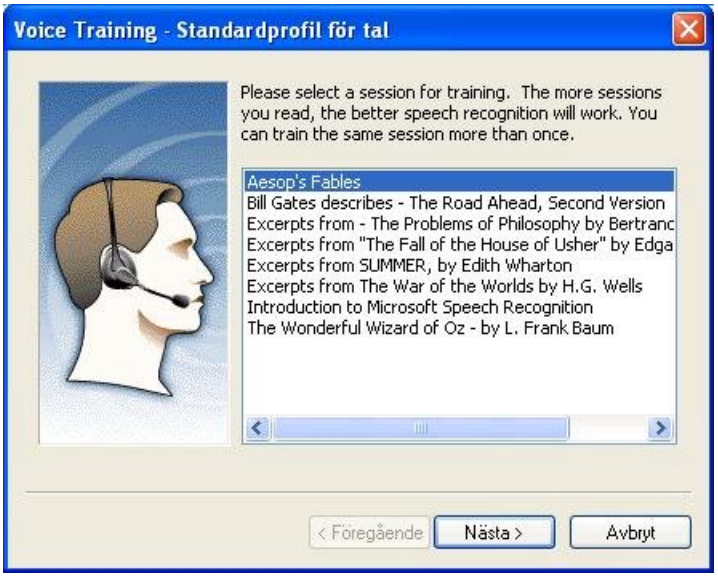


Figure 11. Voice Training for speech engine.

Figure 11 shows the startup screen for voice training. Here the user can choose from eight different short sessions. Once a session is picked, the user reads the text on the screen into the

microphone. The engine collects the necessary data and updates the users profile after each session.

I trained the recognizer, for approximately 2 hours, directly on the laptop using a headset with noise reduction. Later, during the test I use the build in microphone on the PDA. The use of different microphones should affect the recognition process. So a higher confidence level could be reached if the same microphone were used during the training and regular use. Unfortunately, my headset did not work on the PDA.

3.3.7 TextToSpeech

This program is a simple console application built using Microsoft Visual Studio .NET 2003 as environment and C# as development language.

Sometimes it may be desirable to convert a text string into speech. This application does just that. As input it takes a simple text string and produces as output a wave file. The process of converting the text into speech is done by Microsoft's Text-To-Speech SAPI engine. After the conversion is done the file is copied to the PDA using CeCopy.

3.3.8 Manager

This program is a simple console application built using Microsoft Visual Studio .NET 2003 as environment and C# as development language. This program runs on the laptop. Its flowchart is shown in figure 11.

The manager act as a server and handles all the messaging between the different applications. There are eight requests and five acknowledgements that the manager handles. These messages are listed in table 4.

The PDA can request information about the playlist to be downloaded. It can also request a start of the file transfer, as well as request an audioalert and terminate (close) applications. The two available acknowledgements from the PDA are either OK or WAIT.

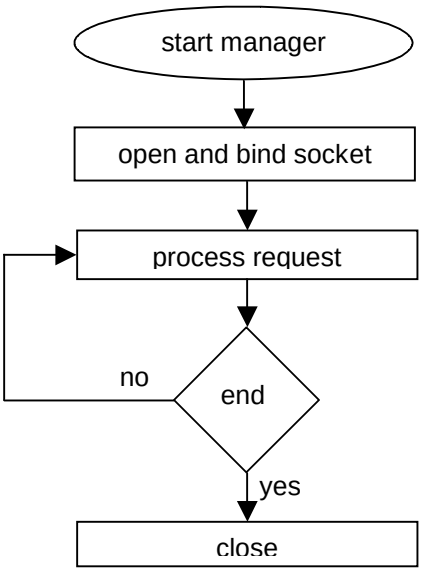


Figure 12. Flowchart of Manager

The FileSender handles only one request, returning the state of the network. However, it sends three different acknowledgements to the PDA: the validity of the requested playlist, the information about the requested playlist and a message indicating that the filetransfer have finished.

The SpeechRecognizer can request commands to be executed at the MediaPlayer. See table 3 for available commands.

Table 4, List of messages handled by Manager

REQ-00-	Close application
REQ-01-<playlist>-	Information about the playlist
REQ-02-<playlist>-	Start FileSender
REQ-03-	Network state
REQ-04-<command>-	Command
REQ-05-<time>-<message>-	Audio alert
ACK-06-	No such file
ACK-07-	No XML file
ACK-08-<size(mb)>-<duration(min)>-	File information
ACK-09-	Network state OK
ACK-10-	Network state WAIT
ACK-11-	Filetransfer finieshed
REQ-12-	Start SpeechRecognizer
REQ-13-	Stop SpeechRecognizer

4 Design Evaluation

In the following section we will evaluate our design from the point of view described in section 3.

4.1 Amount of traffic

In this section follows an evaluation of the amount of traffic used by the two different systems. When using System 1 constant connectivity is clearly required. Conversely, System 2 does not need constant connectivity, since it most of the time is able to play tracks it has stored locally. The results obtained for this study could be viewed in detail in Inmaculada Rangel Vacas's master thesis [38].

4.2 Effect of communication error

When using the voice interface, constant connectivity is required since the real-time audio is sent to the laptop at all times. A total loss of connectivity would cause the recorder to malfunction. So when not using the voice interface, or while no connection is available, it should be turn off.

A better solution would be to implement the recognizer locally at the PDA.

Other effects of communication errors can be viewed in Inmaculada Rangel Vacas's master thesis [38].

4.3 Users opinion

To gauge the significance of this study we asked 15 users (selected from our fellow co-students) about their preferences. We described the two systems and then asked them some questions. The question relevant to this thesis is how they felt about using a voice interface.

The result, shown in figure 12, shows that slightly more than half of these students preferred using a voice interface, while the other half did not. Note that there is no statistical confidence that the true preference is for or against the use of a voice interface.

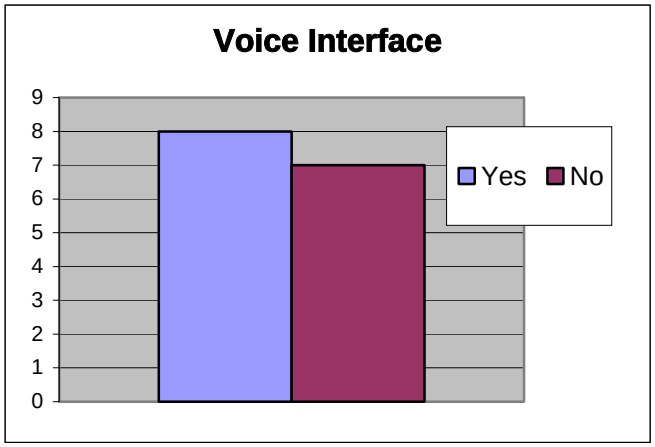


Figure 13. Students who preferred using a voice interface

Main reason for those who answered yes was:

"Having a voice interface could be very useful for handicapped. It is also more comfortable than typing, and in case of long delays, the user can always change to the typing mode."

Main reason for those who answered no was:

"It gives you no privacy, because everybody can hear your command. It causes greater delays in the command that being executed.

More user opinions can be found in Inmaculada Rangel Vacas's master thesis [38]. However, given the level of interest expressed in having a voice interface, I implemented and evaluated this alternative.

4.4 Voice Interface

As previous described, Microsoft's SAPI prefers a wave stream with sample settings 22 kHz 16 Bit Mono [37]. Some tests were done to see how our voice interface responded to different sampling rates and encodings. In the following section a description of the evaluation is given. The profile was trained for approximately 2 hours before the tests.

4.4.1 Evaluation of sampling rates and encodings

To evaluate the voice interface, some tests with different settings were made. Every command in the grammar was tested five times. The SREngineConfidence was printed out to be able to draw some conclusions. SAPI defines SREngineConfidence to be a positive value, with zero indicating the lowest confidence [44]. A very high confidence level has a value over 30,000, while a good confidence level is approximately 20,000. SpeechEngineConfidence could also be used. However, this results only in three values: low (-1), medium (0), and high (1); and does not give us as much information.

Using stereo samples is unnecessary, since the microphone on the PDA is mono and results in equal left and right samples, and simply doubles the bandwidth used.

Two cases were constructed. In *Case 1*, the distance from the user to the PDA was 50 cm. In *Case 2*, the distance was closer, about 5-10 cm. In both cases the audio output (music) was output to a headset; while the PDA's build in microphone was used for audio input.

4.4.1.1 Case 1

First 8 bit mono sound was tested with sample rates at 11, 22, 44 kHz respectively. The results were so bad that it could not be used in our application. The best confidence had a sampling rate at 22 kHz, but it missed twenty commands that were given. The table below shows the result.

Table 5, Confidence results with 8 bit mono, 50 cm

	avg.	# misses
11 kHz	524	26
22 kHz	2206	20
44 kHz	1217	22

With 8-bit mono, the speech recognizer had problems especially with three words, “start”, “next”, and “exit”. They always had the confidence level zero in all tests using 8 bit sound.

Then the same tests were made on 16-bit mono instead, and the results improved. 22 kHz gave the best results if you look at the total number of misses. But 44 kHz gave the best average level of confidence. Here 11 kHz gave the worst result with a total of 19 misses. The results are given in the tables below.

Table 6. Confidence level with 11 kHz 16 bit mono, 50 cm

Trial	1	2	3	4	5	avg.
start	3109	2823	6770	5187	6650	4908
stop	0	0	0	0	0	0
close	229	155	643	0	222	250
play	0	0	0	0	0	0
previous	2628	0	925	0	3658	1442
next	485	2644	3410	4431	0	2194
exit	0	0	0	0	0	0

Table 7. Confidence level with 22 kHz 16 bit mono, 50 cm

Trial	1	2	3	4	5	avg.
start	20584	10725	8158	16482	11045	13399
stop	9760	12738	1668	16774	3716	8931
close	4417	15275	10915	6208	12525	9868
play	5772	5016	0	6982	9856	5525
previous	22106	8052	27040	12665	26863	19345
next	0	6631	0	0	0	1326
exit	12296	11867	562	1124	6122	6394

Table 8. Confidence level with 44 kHz 16 bit mono, 50 cm

Trial	1	2	3	4	5	avg.
start	23576	17884	10643	20036	0	14428
stop	0	10487	15326	10431	0	7249
close	10973	31262	25106	13510	13388	18848
play	9038	9064	4987	1485	702	5055
previous	10209	34352	20091	16385	7532	17714
next	0	0	5955	0	27783	1748
exit	0	0	357	0	11866	2445

Table 9. Average confidence level and # misses, 16 bit mono, 50 cm

	avg.	# misses
11 kHz	1256	19
22 kHz	9256	5
44 kHz	9641	8

4.4.1.2 Case 2

In Case 2, the distance to the PDA is shorten to nearly 5-10 cm, the same distance of a normal headset. Tests on 8 bit mono sound were skipped since the first results were so poor.

Otherwise the same tests were done. All commands were repeated five times and the confidence of the recognition was studied.

Shortening the distance improved the results significantly. This time 44 kHz gave the best result when it comes to number of misses. But 22 kHz gave better confidence with an average level of 22440. Again 11 kHz gave the worst result with an average confidence level at 2061 and a total of twenty misses.

However, 22 kHz and 44 kHz are very closely in both confidence level and total number of misses, so the best one to use would probably be 22 kHz since it uses less bandwidth.

Table 10. Confidence results with 11 kHz 16 bit mono, 5-10 cm

Trial	**1**	**2**	**3**	**4**	**5**	**avg.**
start	7976	0	0	10861	0	3767
stop	0	0	0	0	0	0
close	2898	524	6709	0	2300	2486
play	0	2440	0	0	0	488
previous	3840	8410	9613	2284	5191	5868
next	0	1160	4160	3778	0	1820
exit	0	0	0	0	0	0

Table 11. Confidence results with 22 kHz 16 bit mono, 5-10 cm

Trial	**1**	**2**	**3**	**4**	**5**	**avg.**
start	16547	41393	35225	30943	28366	30495
stop	0	38129	9403	10876	19383	15558
close	17575	29870	27413	21905	18663	23085
play	0	15794	10547	16442	12484	11053
previous	37679	28497	34255	33219	23903	31511
next	24839	14373	30701	30707	20673	26259
exit	36339	13251	19487	14059	14059	19210

Table 12. Confidence results with 44 kHz 16 bit mono, 5-10 cm

Trial	**1**	**2**	**3**	**4**	**5**	**avg.**
start	15472	32214	8834	23754	34880	23031
stop	17560	13815	20986	14986	18919	17253
close	11005	13209	16778	15592	20788	15474
play	0	5448	9683	6925	15360	7483
previous	18437	20759	24303	21780	22352	21526
next	29384	31661	29029	27775	34059	30382
exit	10938	24630	7790	22931	2488	13755

Table 13. Average confidence level and # misses, 16 bit mono, 5-10 cm

	avg.	**# misses**
11 kHz	2061	20
22 kHz	22440	2
44 kHz	18415	1

As the results show, it is not necessarily better to use 22 kHz 16-bit mono rather than 44 kHz as Dalys Sebastian states in [37].

Note that this evaluation test does not hold as a statistical proof. More tests have to be performed to be able to really establish these results.

4.4.2 Bandwidth used

The different sampling rates require different bandwidths to transfer the real-time audio stream from the PDA to the laptop. As we skipped 8-bit mono, due to bad results, we only show measurements of 16-bit mono.

Ethereal was used to capture the RTP packets from the PDA over a period of one minute. Then the transfer rate was calculated.

Table 14, Bandwidth used (kilobytes / second), 16-bit mono

	calculated	measured
11 kHz	21,5	21,5
22 kHz	43,0	42,9
44 kHz	86,1	85,9

Taking the entire test into account using 22 kHz seems to be the best sampling rate. It gives a higher confidence compared to 11 kHz, and is close enough to 44 kHz, at the same time that it uses half the bandwidth compared to 44 kHz. Even better would be if the voice interface could be implemented locally on the PDA, as this would remove the need to send raw samples to the laptop.

4.4.3 Response Time Measurement

The response time is of significant value. If it takes too long to recognize a command and execute it, the user probably would use typing instead. To measure the response time, I measured the time from when a command was said until it was executed. Ethereal was used during this test to calculate the time from the first audio packet entering the network until the response message left the network.

During the evaluation 22 kHz 16 bit mono audio were used. A packet size of 1440 bytes gives us 32.6 ms of audio in each packet. In the worst case it takes approximately 32 ms before the audio buffer is full and is sent out to the network interface, causing some delay.

Each command was executed three times and an average response time was calculated. The results are shown in table 15.

Table 15. Response time measurements

	duration (ms)	delay (ms)	first to last (ms)
start	559	592	1151
stop	229	388	687
close	405	436	842
play	375	484	859
previous	495	459	954
next	380	406	786
exit	282	543	825
average	399	473	872

As can be seen in table 15, the response time for a voice command is in average 500 ms if you add the worst case for the audio buffer to be filled. This is the time calculated from the end of the command until it executed.

For me, as a user, this is acceptable. It takes longer to reach for the PDA and press the right button. To better understand, I did a test. I had my PDA on a table playing a song. Then I measured the time it took me to reach for it, pick out the pointer and change the song. The results was approximately 5 seconds, which is far from 500 ms (0,5 seconds). The time will be even longer when you have to reach down to a pocket.

Here, I only measured the response time for each command three times. Looking at every packet, using Ethereal, trying to figure out exactly where the audio began and ended, took a lot of time. So, this measurement does not hold as statistical proof.

4.4.4 Other issues

Having a voice interface could be both good and bad. It certainly simplifies the interaction with the user if it works. But speech recognition is still in a developing stage and problems such as background noise and audio quality are still issues. We live in a hectic world with cars, construction sites, and cafés, to mention some sources of background noise. However, headsets with noise cancellation exist (using two or more microphones). To be able to understand each other at nightclubs or cafés, people tends to speak louder. Unfortunately speaking louder does not help when it comes to speech recognition. On the contrary it distorts the voice and hinders the speech recognition process. This phenomenon is known as “Lombard Speech” [39]. The best result is given in a quiet environment, speaking clearly in a normal tone.

If a user is able to use a voice interface, a lot of time and energy could be saved. Reaching down to the Pocket PC to change a song on some sort of media player could be substituted with a voice command such as “next”.

The implemented speech recognizer runs on a remote computer. This has some drawbacks. Executing it locally would probably give better response times. Additionally the bandwidth used could be decreased drastically. But as there were no engines for the Pocket PC that suited the project, an alternative method was chosen. For the speech recognizer to work properly constant connectivity is needed also, something the overall project tries to avoid.

4.5 Obstacles

The development of the voice interface faced many obstacles. First I had to make it record constantly reusing the audio buffers. The version from OpenNETCF could only record for a shorter period of time and wrote the audio content out to a wave file. It was necessary to learn how the audio interface used the buffers and how to reuse them.

A timer controlled how long the recorder was recording and by disable it, I could make it record constantly. But since the original recorder wrote the audio buffers to a wave file, it had to be changed so it passed the raw audio data to an event handler.

First declare the recorder that is supposed to be used. Then create it by setting the input device, -1 indicating the default device.

```
Recorder m_Recorder = new Recorder (-1);
```

But before the recorder is started, the application has to create a new thread that handles the actual sending of the RTP packets.

```
Thread sender = new Thread(new ThreadStart(BufferHandling));
sender.Start();
```

To start the audio recorder use the method Record(). As input it takes the SoundFormat as well as a reference to a BufferDoneEventHandler.

```
m_Recorder.Record(SoundFormats.Mono16bit22kHz, new
   BufferDoneEventHandler(DataArrived));

do
{
   Application.DoEvents();
   Thread.Sleep(8);
} while(m_Recorder.Recording);
```

The BufferDoneEventHandler receives the data from the audio buffer and enqueues it. The other thread then works on the queue and put the raw audio into a RTP packet before sending it out on the network.

```
public static void DataArrived(byte[] buffer)
{
  Monitor.Enter(m_Queue.SyncRoot);
  m_Queue.Enqueue(buffer);
  Monitor.Exit(m_Queue.SyncRoot);

  m_PacketEvent.Set();
}
```

The thread-code for sending the raw audio packet, encapsulated in a RTP packet looks like:

```
public static void BufferHandling()
{
  while(m_PacketEvent.WaitOne())
  {
    while(m_Queue.Count > 0)
    {
      Monitor.Enter(m_Queue.SyncRoot);
      byte[] data = (byte[])m_Queue.Dequeue();
      Monitor.Exit(m_Queue.SyncRoot);

      m_RtpPacket.Payload = new BufferChunk(data);
      m_RtpPacket.Sequence = seq++;
      m_RtpPacket.TimeStamp += number_of_samples_packet;

      m_UdpSocket.SendTo((BufferChunk)m_RtpPacket);
    }

    if(!m_Recorder.Recording)
      break;
  }

  m_UdpSocket.Dispose();
}
```

But the real problems started when I tried to process the audio content and put it into RTP packets. The audio buffers seemed to be consumed faster than they could be emptied. So after a short period of time the application stopped recording, i.e. there were no audio buffers left. A fast solution would be to create more audio buffers, but that does not solve the problem if the application runs a longer time. After a lot of debugging I found out that a do-while loop (the one above) unnecessarily consumed a lot of CPU power, and that issue could be solved. Time could have been saved if I had understood how multithreaded applications affected the CPU.

For the complete source code of the AudioRecorder, see appendix A.1.

At first we implemented silence detection on the PDA as well. However, as noted in section 3.3.2.1 a decision was later made to use a real-time audio stream instead, thus this was removed. Unfortunately a lot of time had been spent on silence detection. This could have been avoided if I, in the design stage, earlier had realized that silence detection made the word selection harder for the speech recognizer. Going through necessary components had saved me time.

When it comes to the SpeechRecognizer, problems occurred because the documentation was hard to follow and there were few of previous examples written in C#. The hardest part was how to setup the audio input stream to be accepted as a real-time audio stream. The final implementation uses a custom audio object of type SpAudioPlug, which is specific and unique to this application. It is a DLL (simpleaudio.dll) that is loaded during the SAPI install.

To be able to use the object it has to be loaded as a reference in Visual Studio .NET 2003. This is done by right-clicking the reference line, and choose “Add Reference”, in the Solution Explorer. Then browse for the DLL file. It should be located in the “bin” folder where Microsoft Speech SDK 5.1 is installed. Now reference to it in the code by typing:

```
using SIMPLEAUDIOLib;
```

To use the custom audio object it has to be declared, then it has to be created. Before it could be used it has to be initialized, setting the input audio format type. Then it could be added to the recognizer as an real-time audio input stream.

```
private static SpAudioPlug AudioPlugIn = null;
. . .
AudioPlugIn = new SpAudioPlug();
AudioPlugIn.Init(false, SpeechAudioFormatType.SAFT22kHz16BitMono);
MyRecognizer.AudioInputStream = (ISpeechBaseStream)AudioPlugIn;
. . .
AudioPlugIn.SetData ((object) data);
```

The data can be added by using the method SetData(object). But, since the speech recognizer take an byte stream as input, the data has to be type-casted as an object.

For the complete source code of the SpeechRecognizer, see appendix A.2.

5 Conclusions

The main objectives of this thesis were to understand:

- What are the advantages and disadvantages of having voice commands versus typing on the screen of the PDA?

- What are the effects on the traffic to and from the mobile device of having a personal voice interface?

To be able to evaluate these points a voice interface, used in our system, was developed. The system was formed by several small applications running both on a PDA and a laptop. The voice interface consists of two applications: the AudioRecorder and the SpeechRecognizer. For further information about the system see section 3.3.6.

The voice interface could be seen as very useful to the user. It enabled a lot of functions without any mechanical interaction from the user, apart from the voice command. However, a voice profile of the user has to be created to get good results, something that will take some time initially.

The bandwidth used by the voice interface depends of the quality of the raw audio data. Lower quality comes with lower bandwidth used, but it also results in worse results from the speech recognizer. A sampling rate at 22 kHz with 16 bit mono, results in good response from the recognizer, and uses less bandwidth than when sampling at 44 kHz.

However, it would be better to do speech recognition locally to be able to completely avoid the need for constant connectivity, and reduce the delay that occurs.

Note that if the user is simultaneously listening to entertainment audio, then the sampling rate should be set to this same rate for the audio input since the hardware codec only allows matched rates for input and output. See also the thesis of Ignácio Sanchez Pardo [47] for why this sampling rate should actually be 48 kHz.

6 Open issues and future work

The biggest challenge during the development of our system was the voice interface. Our initial idea and goal was to implement the voice recognizer locally in the PDA, but no speech engines suitable for our handheld device could be found. However, there is a portable version of Sphinx [36] for Linux operating systems running on a PDA. Our solution was to develop a recorder running on a PDA, constantly sending RTP packets with the real-time audio to the speech recognizer running on a laptop.

However, this solution adds some delay to the system so future approaches could be to either:

- find a way of performing the speech recognition locally on the PDA, or
- establishing a SIP session between the audio recorder and the speech recognizer[1]. This could solve all the problems with buffering and sending the audio data.

Future implementation should add context information about the user's location to enable context awareness. This could enable the application to better guess what should be done for the user and to consequently act appropriately. Further improvements to the system regarding context information could also be done in the future.

The voice interface should be improved by implement the recognizer locally on the PDA. This would reduce the bandwidth used by our system. Then connectivity is only needed when transferring new audio content to the PDA. Additionally, this would enable a speech-to-text then text-to-speech communication system. Some elements of this are the topics of another thesis project [46].

[1] Another thesis project [45] is examining the porting of a SIP client to this PDA under Pocket PC.

References

[1] Maria José Parajón Dominguez, *"Audio for Nomadic Users"*,
Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Master of Science Thesis performed at KTH. (2003)

[2] Andreas Wennlund, *"Context-aware Wearable Device for Reconfigurable Application Networks"*, Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Master of Science Thesis performed at KTH. (2003)

[3] O. Conlan, et al., *"Next Generation Context Aware Services"*,
Knowledge and Data Engineering Group (KDEG), Trinity College, Dublin (2003)
Last accessed: 2004-09-22 http://www.m-zones.org/deliverables/d234_1/papers/conlan-adaptive-services.pdf

[4] A. Schmidt, et al.,*"Advanced Interaction in Context"*,
TecO, University of Karlsruhe, Germany, (1999)
Last accessed: 2004-09-22 http://www.m-zones.org/deliverables/d234_1/papers/conlan-adaptive-services.pdf

[5] Anind K Dey and Gregory D. Abowd,
"Towards a Better Understanding of Context and Context-Awareness",
Graphics, Visualization and Usability Center and College of Computing,
Georgia Institute of Technology, Atlanta, GA, USA, (1999)
Last accessed: 2004-09-22 http://www.it.usyd.edu.au/~bob/IE/99-22.pdf

[6] E. Tuulari, *"Context aware hand-held devices"*,
VTT Technical Research Centre of Finland, ESPOO, (2000)
Last accessed: 2004-09-27 http://www.inf.vtt.fi/pdf/publications/2000/P412.pdf

[7] J. Klein and A. Toney, *"What is a wearable Computer?"*,
Metrics for Assessing Wearable Devices
Last accessed: 2004-09-30 http://gro.hhhh.org/~joeboy/publications/toney_klein_how_wearable.pdf

[8] N. Sawhney and C. Schmandt, *"Nomadic Radio: Scaleable and Contextual Notification for Wearable Audio Messaging"*, Proceedings of the Conference on Human Factors and Computing Systems (CHI)´99, May 15-20, (1999)
Last accessed: 2004-09-30 http://citeseer.nj.nec.com/sawhney99nomadic.hml

[9] Norman Walsh, *"What is XML?"*,
O'Reilly XML.com – xml from inside to out, (1998)
Last accessed: 2004-10-08 http://www.xml.com/pub/a/98/10/guide1.html#AEN58

[10] Charles F. Goldfarb (Project Editor) *"ISO 8879 TC2"*, (1997)
Last accessed: 2004-10-08 http://www.y12.doe.gov/sgml/wg8/document/1955.htm

[11] Mark T. Smith and Gerald Q. Maguire Jr., *"SmartBadge/BadgePad version 4"*,
HP Labs and Royal Institute of Technology (KTH),
Last accessed: 2004-10-10 http://www.it.kth.se/~maguire/badge4.html

[12] *"The Active Badge System"*, AT&T Laboratories, Cambridge
Last accessed: 2004-10-10 http://www.uk.research.att.com/ab.html

[13] *"Festival Lite"*, Carneige Mellon University, Pittsburgh, PA, USA
Last accessed: 2004-10-10 http://www.speech.cs.cmu.edu/flite/

[14] Norman M. Sadeh, et al., *"MyCampus: An Agent-Based Environment for Context-Aware Mobile Services"*, Carneige Mellon University, Pittsburgh, PA, USA, 2002
Last accessed: 2004-10-10 http://autonomousagents.org/ubiquitousagents/2002/papers/papers/29.pdf

[15] RFC 768 – User Datagram Protocol
Last accessed: 2004-10-12 http://www.faqs.org/rfcs/rfc768.html

[16] RFC 1889 – Real-time Transport Protocol
Last accessed: 2004-10-13 http://www.ietf.org/rfc/rfc1889.txt

[17] "H.323", From Wikipedia, the free encyclopedia
Last accessed: 2004-10-13 http://en.wikipedia.org/wiki/H.323

[18] RFC-3261 - Session Initiation Protocol,
Last accessed: 2004-10-13 http://www.ietf.org/rfc/rfc3261.txt

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

[20] UDP (User Datagram Protocol), Tom Sheldon's Linktionary.com
Last accessed: 2004-10-15 http://www.linktionary.com/u/udp.html

[21] Waveform – Webopedia Computer Dictionary
Last accessed 2005-01-13 http://www.webopedia.com/TERM/W/waveform.html

[22] An introduction to P/Invoke of Microsoft .NET Compact Framework
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/netcfintrointerp.asp
Last accessed: 2005-01-13

[23] OpenNETCF.org – Smart Device Framework
Last accessed: 2005-01-13 http://www.opennetcf.org

[24] Microsoft WAVE Soundfile Format
Last accessed: 2005-02-21 http://ccrma.stanford.edu/CCRMA/Courses/422/projects/WaveFormat/

[25] Pocket Streamer
Last accessed: 2005-02-06 http://www.thecodeproject.com/netcf/PocketStreamer.asp

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

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

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

[29] Windows Media Encoder 9 SDK

Last accessed: 2005-02-06 http://www.microsoft.com/downloads/details.aspx?FamilyID=000a16f5-d62b-4303-bb22-f0c0861be25b&DisplayLang=en

[30] RealOne Player
Last accessed 2005-02-22 http://www.real.com/player/

[31] Microsoft's Windows Media Player
Last accessed 2005-02-22 http://www.microsoft.com/windows/windowsmedia/default.aspx

[32] Winamp
Last accessed 2005-02-22 http://www.winamp.com/

[33] Microsoft Portrait
Last accessed 2005-03-01 http://research.microsoft.com/~jiangli/portrait/

[34] Windows Mobile Developer Power Toys
Last accessed 2005-03-01 http://www.microsoft.com/downloads/details.aspx?familyid=74473fd6-1dcc-47aa-ab28-6a2b006edfe9&displaylang=en

[35] Recording and Playing sound with the Waveform Audio Interface
Last accessed 2005-05-02
http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnnetcomp/html/WaveInOut.asp

[36] The CMU Sphinx Group Open Source Speech Recognition Engines
Last accessed 2005-05-05 http://cmusphinx.sourceforge.net/html/cmusphinx.php

[37] Dalys Sebastian, Field-Deployable Voice-Controlled Ultrasound Scanner System
Last accessed 2005-05-24
http://www.wpi.edu/Pubs/ETD/Available/etd-0625104-170951/unrestricted/sebastian.pdf

[38] Inmaculada Rangel Vacas, *"Context Awareness and Adaptive Mobile Audio"*
Master of Science Thesis, Department of Microelectronics and Information Technology (IMIT), Royal institute of Technology (KTH), April 2005.

[39] Kimberlee A. Kemble, *"An introduction to Speech Recognition"*
Last accessed 2005-05-30 http://www.voicexmlreview.org/Mar2001/features/recognition2.html

[40] *"Speech Recognition"*, DesktopMates.com
Last accessed 2005-06-15 http://desktopmates.com/speech.html

[41] Carlos Marco Arranz, *"IP Telephony, peer-to-peer versus SIP"*
Master of Science Thesis, Department of Microelectronics and Information Technology (IMIT), Royal institute of Technology (KTH), June 2005.

[42] RFC2326 - Real Time Streaming Protocol, RTSP
Last accessed 2005-06-15 http://www.ietf.org/rfc/rfc2326.txt

[43] HP iPAQ Pocket PC h5500 series specification
Last accessed 2005-06-15 http://h18002.www1.hp.com/products/quickspec/11646_div/11646_div.html

[44] SR Engine Vendor Porting Guide
Last accessed 2005-06-16
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/SAPI51sr/html/Struct_SPPHRASERULE.asp

[45] Andreas Ångström, *"Peer-to-peer versus SIP"*
Master of Science Thesis, Department of Microelectronics and Information Technology (IMIT), Royal institute of Technology (KTH).

[46] Allessandro Sacchi, "A Motivation for Text on RTP"
Master of Science Thesis, Department of Microelectronics and Information Technology (IMIT), Royal institute of Technology (KTH).

[47] Ignácio Sanchez Pardo, *"Spatial Audio for the Mobile User"*
Master of Science Thesis, Department of Microelectronics and Information Technology (IMIT), Royal institute of Technology (KTH), June 2005.

[48] Communication Research Group, University of Southhampton, *"The GSM Codec"*
Last accessed 2005-06-21 http://www-mobile.ecs.soton.ac.uk/speech_codecs/standards/gsm.html

Appendix A – Application's Source Code

A.1 AudioRecorder

A.1.1 MainApplication.cs

```
//=========================================================================================
//
//   AdaptiveAudio.AudioRecorder
//   Copyright (c) 2005, Johan Sverin
//
//   Version: 1.5
//   Date   : 19/05/2005
//
//   Description:
//   n audio recorder, that constantly records audio on a wearable device.
//   Each audio buffer is send as an RTP packet over the network.
//
//=========================================================================================

using System;
using System.Data;
using System.Threading;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using System.Net;

using Network.RtpLibrary;
using Network.NetworkingBasics;
using Network.Sockets;

using OpenNETCF.Multimedia.Audio;

namespace AdaptiveAudio.AudioRecorder
{

  class MainApplication
  {
    private static Recorder m_Recorder = null;
    private static RtpPacket m_RtpPacket = null;
    private static UDPSocket m_UdpSocket = null;

    private static Random rnd = new Random();
    private static ushort seq = 0;
    private static uint ts = 0;
    private static uint ssrc = 0;

    private static Queue m_Queue = new Queue(10);
    private static ManualResetEvent m_PacketEvent = new ManualResetEvent(false);

    private const int PACKET_SIZE = Recorder.BUFFER_SIZE;
    private const int number_of_samples_packet = (PACKET_SIZE) / 4;

    static void Main(string[] args)
    {
      IPHostEntry r_HostInfo = Dns.Resolve("unnamed");
      IPAddress r_Address = r_HostInfo.AddressList[0];

      IPHostEntry h_HostInfo = Dns.Resolve(Dns.GetHostName());
      IPAddress h_Address = h_HostInfo.AddressList[0];

      try
      {
```

```
      m_UdpSocket = new UDPSocket(r_Address, 5003, 5004);
      m_UdpSocket.DelayBetweenPackets = 8;

      // Create the recorder, with default audio device -1;
      m_Recorder = new Recorder(-1);

      // Initilize the RtpPacket
      InitializeRtp();

      Thread PrintInfo = new Thread(new ThreadStart(PrintScreen));
      Thread PacketSender = new Thread(new ThreadStart(BufferHandling));

      PacketSender.Start();

      // Start recording with given format
      m_Recorder.Record(SoundFormats.Mono16bit22kHz,
        new BufferDoneEventHandler(DataArrived));

      do
      {
        Application.DoEvents();
        Thread.Sleep(8);
      }while(m_Recorder.Recording);
    }
    catch(Exception e)
    {
      Console.WriteLine("An error occured, exit...");
      Console.WriteLine(e.ToString());
    }

  }

  /// <summary>
  /// Receives the data from the buffer and enqueues it
  /// </summary>
  /// <param name="buffer"></param>
  public static void DataArrived(byte[] buffer)
  {
    Monitor.Enter(m_Queue.SyncRoot);
    m_Queue.Enqueue(buffer);
    Monitor.Exit(m_Queue.SyncRoot);

    m_PacketEvent.Set();
  }

  /// <summary>
  /// Dequeues the buffer and sends the packets
  /// </summary>
  public static void BufferHandling()
  {
    while(m_PacketEvent.WaitOne())
    {
      while(m_Queue.Count > 0)
      {
        Monitor.Enter(m_Queue.SyncRoot);
        byte[] data = (byte[])m_Queue.Dequeue();
        Monitor.Exit(m_Queue.SyncRoot);

        m_RtpPacket.Payload = new BufferChunk(data);
        m_RtpPacket.Sequence = seq++;
        m_RtpPacket.TimeStamp += number_of_samples_packet;

        m_UdpSocket.SendTo((BufferChunk)m_RtpPacket);
      }

      if(!m_Recorder.Recording)
        break;
```

```
      }

      m_UdpSocket.Dispose();
    }

    /// <summary>
    /// Initialize the RtpPacket
    /// </summary>
    public static void InitializeRtp()
    {
      // Create a new RtpPacket with given payload size
      m_RtpPacket = new RtpPacket(PACKET_SIZE +
        RtpPacket.RTP_HEADER_SIZE +
        RtpPacket.HEADER_EXTENSIONS_SIZE);

      // Set PayloadType
      m_RtpPacket.PayloadType = PayloadType.L16;

      // Set sequence number, randomize value
      seq = (ushort) rnd.Next(1, int.MaxValue);
      m_RtpPacket.Sequence = seq;

      // Set timestamp, randomize value
      ts = (uint) rnd.Next(1, int.MaxValue);
      m_RtpPacket.TimeStamp = ts;

      // Set SSRC, randomize value
      ssrc = (uint) rnd.Next(1, int.MaxValue);
      m_RtpPacket.SSRC = ssrc;
    }
  }
}
```

A.1.2 Recorder.cs

```
//=============================================================================
//
//   OpenNETCF.Multimedia.Audio.Recorder
//   Copyright (c) 2003, OpenNETCF.org
//
//   This library is free software; you can redistribute it and/or modify it under
//   the terms of the OpenNETCF.org Shared Source License.
//
//   This library is distributed in the hope that it will be useful, but
//   WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//   FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//   for more details.
//
//   You should have received a copy of the OpenNETCF.org Shared Source License
//   along with this library; if not, email licensing@opennetcf.org to get a copy.
//
//   For general enquiries, email enquiries@opennetcf.org or visit our website at:
//   http://www.opennetcf.org
//
//=============================================================================
//
//   This file has been modyfied to better fit our needs of the recorder.
//
//   Name: Johan Sverin
//   Date: May 19, 2005
//
//=============================================================================

using System;
using System.IO;
using System.Threading;
using System.Collections;

using OpenNETCF.Win32;

namespace OpenNETCF.Multimedia.Audio
{
  /// <summary>
  /// Recorder class. Wraps low-level WAVE API for recording purposes.
  /// </summary>
  public class Recorder : Audio
  {
    /// <summary>
    /// Handles the event that is fired when wave device is successfully opened.
    /// </summary>
    public event WaveOpenHandler WaveOpen;

    /// <summary>
    /// Handles the event that is fired when wave device is successfully closed.
    /// </summary>
    public event WaveCloseHandler WaveClose;

    /// <summary>
    /// Handles the event that is fired when recording is stopped
    /// (on timer or by calling <see cref="Recorder.Stop">Stop</see> method.
    /// </summary>
    public event WaveFinishedHandler DoneRecording;

    /// <summary>
    /// Hardware interface instance for this recording.
    /// </summary>
    private IntPtr m_hWaveIn = IntPtr.Zero;
```

```
/// <summary>
/// SoundMessageWindow used to receive messages from the audio system.
/// </summary>
private SoundMessageWindow m_recmw;

/// <summary>
/// Specifies whether the device is recording.
/// </summary>
private bool recording = false;

/// <summary>
/// Specifies whether the recording has finished.
/// </summary>
private bool recordingFinished = false;

/// <summary>
/// Size of each record buffer.
/// </summary>
private int m_recBufferSize;

/// <summary>
/// Instance of the WaveFormatEx header for this file.
/// </summary>
private WaveFormatEx m_recformat;

/// <summary>
/// Handles the event that is fired when the buffer is full.
/// </summary>
private event BufferDoneEventHandler m_DoneProc;

/// <summary>
/// Whether the Recorder is presently recording
/// </summary>
public bool Recording { get { return recording; } }

/// <summary>
/// Creates Recorder object and attaches it to the default wave device
/// </summary>
public Recorder()
{
  m_qBuffers = new Queue(MaxBuffers);
  m_HandleMap = new Hashtable(MaxBuffers);
}

/// <summary>
/// Creates Recorder object and attaches it to the given wave device
/// </summary>
/// <param name="AudioDeviceID">Wave device ID</param>
public Recorder(int AudioDeviceID) : this()
{
  m_deviceID = AudioDeviceID;
}

/// <summary>
/// Number of wave input devices in the system
/// </summary>
public static int NumDevices { get { return Core.waveInGetNumDevs(); } }

/// <summary>
/// Stop recording operation currently in progress.
/// Throws an error if no recording operation is in progress
/// </summary>
public void Stop()
{
  if (!recordingFinished)
  {
    Thread.Sleep(1000);
```

```
      CheckWaveError( Core.waveInReset(m_hWaveIn) );
      recordingFinished = true;
    }
  }

  /// <summary>
  /// Record sound data at 11025 sps and 1 channel
  /// </summary>
  /// <param name="DoneProc">Event handler that takes care of the data</param>
  public void Record(BufferDoneEventHandler DoneProc)
  {
    Record(SoundFormats.Stereo16bit44kHz, DoneProc);
  }

  /// <summary>
  /// Record sound data using given wave format
  /// </summary>
  /// <param name="SoundFormat">Sound format to record in.</param>
  /// <param name="DoneProc">Event handler that takes care of the data</param>
  public void Record(SoundFormats SoundFormat, BufferDoneEventHandler DoneProc)
  {
    m_DoneProc = DoneProc;
    m_hWaveIn = IntPtr.Zero;

    // Only allow 1 recording session at a time
    if(recording) { throw new InvalidOperationException("Already recording"); }

    // Set our global flag
    recording = true;

    if ( m_qBuffers == null )
      m_qBuffers = new Queue(MaxBuffers);
    if ( m_HandleMap  == null )
      m_HandleMap = new Hashtable(MaxBuffers);

    m_recformat = new WaveFormatEx();

    m_recmw = new SoundMessageWindow();
    m_recmw.WaveDoneMessage +=new WaveDoneHandler(mw_WaveDoneMessage);
    m_recmw.WaveCloseMessage +=new WaveCloseHandler(mw_WaveCloseMessage);
    m_recmw.WaveOpenMessage += new WaveOpenHandler(mw_WaveOpenMessage);

    switch(SoundFormat)
    {
      case SoundFormats.Mono16bit11kHz:
        m_recformat.Channels = 1;
        m_recformat.SamplesPerSec = 11025;
        m_recformat.BitsPerSample = 16;
        break;
      case SoundFormats.Mono16bit22kHz:
        m_recformat.Channels = 1;
        m_recformat.SamplesPerSec = 22050;
        m_recformat.BitsPerSample = 16;
        break;
      case SoundFormats.Mono16bit44kHz:
        m_recformat.Channels = 1;
        m_recformat.SamplesPerSec = 44100;
        m_recformat.BitsPerSample = 16;
        break;
      case SoundFormats.Mono8bit11kHz:
        m_recformat.Channels = 1;
        m_recformat.SamplesPerSec = 11025;
        m_recformat.BitsPerSample = 8;
        break;
      case SoundFormats.Mono8bit22kHz:
        m_recformat.Channels = 1;
        m_recformat.SamplesPerSec = 22050;
```

```
      m_recformat.BitsPerSample = 8;
      break;
    case SoundFormats.Mono8bit44kHz:
      m_recformat.Channels = 1;
      m_recformat.SamplesPerSec = 44100;
      m_recformat.BitsPerSample = 8;
      break;
    case SoundFormats.Stereo16bit11kHz:
      m_recformat.Channels = 2;
      m_recformat.SamplesPerSec = 11025;
      m_recformat.BitsPerSample = 16;
      break;
    case SoundFormats.Stereo16bit22kHz:
      m_recformat.Channels = 2;
      m_recformat.SamplesPerSec = 22050;
      m_recformat.BitsPerSample = 16;
      break;
    case SoundFormats.Stereo16bit44kHz:
      m_recformat.Channels = 2;
      m_recformat.SamplesPerSec = 44100;
      m_recformat.BitsPerSample = 16;
      break;
    case SoundFormats.Stereo8bit11kHz:
      m_recformat.Channels = 2;
      m_recformat.SamplesPerSec = 11025;
      m_recformat.BitsPerSample = 8;
      break;
    case SoundFormats.Stereo8bit22kHz:
      m_recformat.Channels = 2;
      m_recformat.SamplesPerSec = 22050;
      m_recformat.BitsPerSample = 8;
      break;
    case SoundFormats.Stereo8bit44kHz:
      m_recformat.Channels = 2;
      m_recformat.SamplesPerSec = 44100;
      m_recformat.BitsPerSample = 8;
      break;
  }

  m_recformat.FormatTag = WAVE_FORMAT_PCM;
  m_recformat.AvgBytesPerSec = m_recformat.SamplesPerSec * m_recformat.Channels;
  m_recformat.BlockAlign = (short)((m_recformat.Channels *
    m_recformat.BitsPerSample) / 8);
  m_recformat.Size = 0;

  //Console.WriteLine("  Channels:    {0}", m_recformat.Channels);
  //Console.WriteLine("  Bits/Sample: {0}", m_recformat.BitsPerSample);
  //Console.WriteLine("  Samples/Sec: {0}", m_recformat.SamplesPerSec);
  //Console.WriteLine("  Bytes/Sec:   {0}\n", m_recformat.AvgBytesPerSec);

  // Check for support of selected format
  CheckWaveError(OpenNETCF.Win32.Core.waveInOpen(out m_hWaveIn, WAVE_MAPPER,
    m_recformat, IntPtr.Zero, 0, WAVE_FORMAT_QUERY));

  // Open wave device
  CheckWaveError(OpenNETCF.Win32.Core.waveInOpen(out m_hWaveIn,
    (uint)m_deviceID, m_recformat, m_recmw.Hwnd, 0, CALLBACK_WINDOW));

  // Set buffersize (orignal: m_recformat.SamplesPerSec * m_recformat.Channels)
  m_recBufferSize = BUFFER_SIZE;

  for ( int i = 0; i < 100; i ++ )
  {
    WaveHeader hdr = GetNewRecordBuffer( m_recBufferSize );

    // Send the buffer to the device
    CheckWaveError(Core.waveInAddBuffer(m_hWaveIn, hdr.Header,
```

```
        hdr.HeaderLength));
    }

    // Begin recording
    CheckWaveError(Core.waveInStart(m_hWaveIn));
    recordingFinished = false;

    //Console.WriteLine("Recording started!");

  }

  /// <summary>
  /// Creates a recording buffer
  /// </summary>
  /// <param name="dwBufferSize"></param>
  /// <returns>new buffer as WaveHeader</returns>
  private WaveHeader GetNewRecordBuffer(int dwBufferSize)
  {
    WaveHeader hdr = new WaveHeader(dwBufferSize);
    Monitor.Enter(m_HandleMap.SyncRoot);
    m_HandleMap.Add(hdr.Header.ToInt32(), hdr);
    Monitor.Exit(m_HandleMap.SyncRoot);

    // Prepare the header
    CheckWaveError(Core.waveInPrepareHeader(m_hWaveIn, hdr.Header,
      hdr.HeaderLength));
    return hdr;
  }

  /// <summary>
  /// Retrieves the data from the buffer that have been filled.
  /// </summary>
  private void DumpRecordBuffers()
  {
    while( m_qBuffers.Count > 0 )
    {
      if(m_qBuffers.Count > 1)
        Console.WriteLine(m_qBuffers.Count);

      Monitor.Enter(m_qBuffers.SyncRoot);
      WaveHeader hdr = (WaveHeader)m_qBuffers.Dequeue();
      Monitor.Exit(m_qBuffers.SyncRoot);

      m_DoneProc(hdr.GetData());
      hdr.Dispose();
    }

    Monitor.Exit(m_qBuffers.SyncRoot);
  }

  private void mw_WaveOpenMessage(object sender)
  {
    if(WaveOpen != null) { WaveOpen(this); }
  }

  private void mw_WaveCloseMessage(object sender)
  {
    if(WaveClose != null) {        WaveClose(this); }
  }

  private void mw_WaveDoneMessage(object sender, IntPtr wParam, IntPtr lParam)
  {

    // Retrieve Waveheader object by the lpHeader pointer
    Monitor.Enter(m_HandleMap.SyncRoot);
    WaveHeader hdr = m_HandleMap[lParam.ToInt32()] as WaveHeader;
    m_HandleMap.Remove(hdr.Header.ToInt32());
```

```
      Monitor.Exit(m_HandleMap.SyncRoot);

      // Unprepare the header
      CheckWaveError(Core.waveInUnprepareHeader(m_hWaveIn, hdr.Header,
        hdr.HeaderLength));
      hdr.RetrieveHeader();
      m_qBuffers.Enqueue(hdr);

      if ( recordingFinished )
      {
        DumpRecordBuffers();
        CheckWaveError(Core.waveInClose(m_hWaveIn));

        // Clean up the messageWindow
        m_recmw.Dispose();

        // Reset the global flag
        recording = false;

        // Set our event
        if(DoneRecording != null) {   DoneRecording(); }

        foreach( WaveHeader whdr in m_HandleMap.Values )
          whdr.Dispose();

        m_HandleMap.Clear();
      }
      else
      {
        hdr = GetNewRecordBuffer(m_recBufferSize);
        CheckWaveError(Core.waveInAddBuffer(m_hWaveIn, hdr.Header,
          hdr.HeaderLength));
        DumpRecordBuffers();
      }
    }
  }
}
```

A.1.3 SoundMessageWindow.cs

```
//=========================================================================================
//
//   OpenNETCF.Multimedia.Audio.Recorder
//   Copyright (c) 2003, OpenNETCF.org
//
//   This library is free software; you can redistribute it and/or modify it under
//   the terms of the OpenNETCF.org Shared Source License.
//
//   This library is distributed in the hope that it will be useful, but
//   WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//   FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//   for more details.
//
//   You should have received a copy of the OpenNETCF.org Shared Source License
//   along with this library; if not, email licensing@opennetcf.org to get a copy.
//
//   For general enquiries, email enquiries@opennetcf.org or visit our website at:
//   http://www.opennetcf.org
//
//=========================================================================================
//
//   This file has been modyfied to better fit our needs of the recorder.
//
//   Name: Johan Sverin
//   Date: May 19, 2005
//
//   Original filename: Recorder.cs
//
//=========================================================================================

using System;
using Microsoft.WindowsCE.Forms;

namespace OpenNETCF.Multimedia.Audio
{
  internal class SoundMessageWindow : MessageWindow
  {
    public event WaveOpenHandler  WaveOpenMessage;
    public event WaveCloseHandler WaveCloseMessage;
    public event WaveDoneHandler  WaveDoneMessage;

    public const int WM_WOM_OPEN  = 0x03BB;
    public const int WM_WOM_CLOSE = 0x03BC;
    public const int WM_WOM_DONE  = 0x03BD;
    public const int MM_WIM_OPEN  = 0x03BE;
    public const int MM_WIM_CLOSE = 0x03BF;
    public const int MM_WIM_DATA  = 0x03C0;

    public SoundMessageWindow()
    {
    }

    protected override void WndProc(ref Message msg)
    {
      switch(msg.Msg)
      {
        case WM_WOM_CLOSE:
        case MM_WIM_CLOSE:
          if(WaveCloseMessage != null)
          {
            WaveCloseMessage(this);
          }
          break;
        case WM_WOM_OPEN:
```

```
      case MM_WIM_OPEN:
        if(WaveOpenMessage != null)
        {
          WaveOpenMessage(this);
        }
        break;
      case MM_WIM_DATA:
      case WM_WOM_DONE:
        if(WaveDoneMessage != null)
        {
          WaveDoneMessage(this, msg.WParam, msg.LParam);
        }
        break;
    }


    // Call the base class WndProc for default message handling
    base.WndProc(ref msg);
  }
 }
}
```

A.1.4 Core.cs

```
//=========================================================================================
//
//   OpenNETCF.Multimedia.Audio.Recorder
//   Copyright (c) 2003, OpenNETCF.org
//
//   This library is free software; you can redistribute it and/or modify it under
//   the terms of the OpenNETCF.org Shared Source License.
//
//   This library is distributed in the hope that it will be useful, but
//   WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//   FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//   for more details.
//
//   You should have received a copy of the OpenNETCF.org Shared Source License
//   along with this library; if not, email licensing@opennetcf.org to get a copy.
//
//   For general enquiries, email enquiries@opennetcf.org or visit our website at:
//   http://www.opennetcf.org
//
//=========================================================================================
//
//   This file has been modyfied to better fit our needs of the recorder.
//
//   Name: Johan Sverin
//   Date: May 19, 2005
//
//=========================================================================================

using System;
using System.Text;
using System.Runtime.InteropServices;

using OpenNETCF.Multimedia;

namespace OpenNETCF.Win32
{

  /// <summary>
  /// OpenNETCF Win32 API Wrapper Class for CoreDLL.dll
  /// </summary>
  public class Core
  {
    /// <summary>
    /// Class for getting audio device capabilities
    /// </summary>
    public class WaveInCaps
    {
      private const int MAXPNAMELEN = 32;

      private const int wMIDOffset          = 0;
      private const int wPIDOffset          = wMIDOffset + 2;
      private const int vDriverVersionOffset = wPIDOffset + 2;
      private const int szPnameOffset       = vDriverVersionOffset + 4;
      private const int dwFormatsOffset     = szPnameOffset + MAXPNAMELEN * 2;
      private const int wChannelsOffset     = dwFormatsOffset + 4;
      private const int wReserved1Offset    = wChannelsOffset + 2;

      private byte[] flatStruct = new byte[2 + 2 + 4 + MAXPNAMELEN * 2 + 4 + 2 + 2];

      public byte[] ToByteArray() { return flatStruct; }
      public static implicit operator byte[]( WaveInCaps wic )
        { return wic.flatStruct; }

      public WaveInCaps() { Array.Clear(flatStruct, 0, flatStruct.Length); }
```

```
  public WaveInCaps( byte[] bytes ) : this( bytes, 0 ){ }
  public WaveInCaps( byte[] bytes, int offset )
    { Buffer.BlockCopy( bytes, offset, flatStruct, 0, flatStruct.Length ); }

  public short MID {get{return BitConverter.ToInt16(flatStruct, wMIDOffset);}}
  public short PID {get{return BitConverter.ToInt16(flatStruct, wPIDOffset);}}

  public int DriverVersion {
    get{return BitConverter.ToInt32(flatStruct, vDriverVersionOffset);}
  }

  public string szPname {get{return Encoding.Unicode.GetString(
    flatStruct, szPnameOffset, MAXPNAMELEN * 2).Trim('\0'); } }

  public int Formats
  {
    get
    {
      return BitConverter.ToInt32(flatStruct, dwFormatsOffset);
    }
    set
    {
      byte[] bytes = BitConverter.GetBytes( value );
      Buffer.BlockCopy(
        bytes, 0, flatStruct, dwFormatsOffset, Marshal.SizeOf(value)
      );
    }
  }

  public short Channels
  {
    get
    {
      return BitConverter.ToInt16(flatStruct, wChannelsOffset);
    }
    set
    {
      byte[] bytes = BitConverter.GetBytes( value );
      Buffer.BlockCopy(
        bytes, 0, flatStruct, wChannelsOffset, Marshal.SizeOf(value)
      );
    }
  }

  public short wReserved1 {
    get { return BitConverter.ToInt16(flatStruct, wReserved1Offset); } }
}

[DllImport ("coredll.dll", EntryPoint="waveInGetDevCaps", SetLastError=true)]
public static extern int waveInGetDevCaps(int uDeviceID, byte[] pwic,int cbwic);

[DllImport ("coredll.dll", EntryPoint="waveInGetNumDevs", SetLastError=true)]
public static extern int waveInGetNumDevs();

[DllImport ("coredll.dll", EntryPoint="waveInReset", SetLastError=true)]
public static extern int waveInReset(IntPtr hwi);

[DllImport ("coredll.dll", EntryPoint="waveInAddBuffer", SetLastError=true)]
public static extern int waveInAddBuffer(IntPtr hwi, byte[] pwh, int cbwh);

[DllImport ("coredll.dll", EntryPoint="waveInAddBuffer", SetLastError=true)]
public static extern int waveInAddBuffer(IntPtr hwi, IntPtr lpHdr, int cbwh);


[DllImport ("coredll.dll", EntryPoint="waveInOpen", SetLastError=true)]
internal static extern int waveInOpen(out IntPtr t, uint id,
  WaveFormatEx pwfx, IntPtr dwCallback, int dwInstance, int fdwOpen);
```

```
    [DllImport ("coredll.dll", EntryPoint="waveInStart", SetLastError=true)]
    public static extern int waveInStart(IntPtr hwi);

    [DllImport ("coredll.dll", EntryPoint="waveInPrepareHeader", SetLastError=true)]
    public static extern int waveInPrepareHeader(IntPtr hwi, byte[] pwh, int cbwh);

    [DllImport ("coredll.dll", EntryPoint="waveInPrepareHeader", SetLastError=true)]
    public static extern int waveInPrepareHeader(IntPtr hwi, IntPtr lpHdr,int cbwh);

    [DllImport ("coredll.dll", EntryPoint="waveInClose", SetLastError=true)]
    public static extern int waveInClose(IntPtr hDev);

    [DllImport ("coredll.dll",EntryPoint="waveInUnprepareHeader",SetLastError=true)]
    public static extern int waveInUnprepareHeader(IntPtr hwi, byte[] pwh,int cbwh);

    [DllImport ("coredll.dll",EntryPoint="waveInUnprepareHeader",SetLastError=true)]
    public static extern int waveInUnprepareHeader(
      IntPtr hwi, IntPtr lpHdr, int cbwh);

    /// <summary>
    /// LocalAlloc flags
    /// </summary>
    public enum MemoryAllocFlags : int
    {
      /// <summary>
      /// Allocates fixed memory.
      /// The return value is a pointer to the memory object.
      /// </summary>
      GMEM_FIXED = 0x0000,
      /// <summary>
      /// Initializes memory contents to zero.
      /// </summary>
      LMEM_ZEROINIT  = 0x0040,
      /// <summary>
      /// Combines the Fixed and ZeroInit flags.
      /// </summary>
      LPTR = (GMEM_FIXED | LMEM_ZEROINIT)
    }

    /// <summary>
    /// Frees memory previously allocated from unmanaged memory.
    /// </summary>
    public static void LocalFree(IntPtr hMem)
    {
      LocalFreeCE(hMem);
    }

    /// <summary>
    /// Allocates unmanaged memory.
    /// </summary>
    /// <param name="uFlags"></param>
    /// <param name="uBytes"></param>
    /// <returns></returns>
    public static IntPtr LocalAlloc(MemoryAllocFlags uFlags, int uBytes)
    {
      return LocalAllocCE((uint)uFlags, (uint)uBytes);
    }

    [DllImport("coredll.dll",EntryPoint="LocalAlloc",SetLastError=true)]
    public static extern IntPtr LocalAllocCE(uint uFlags, uint Bytes);

    [DllImport("coredll.dll",EntryPoint="LocalFree",SetLastError=true)]
    public static extern IntPtr LocalFreeCE(IntPtr hMem);

  }
}
```

A.1.5 WaveHeader.cs

```
//=========================================================================================
//
//   OpenNETCF.Multimedia.Audio.WaveHeader
//   Copyright (c) 2003, OpenNETCF.org
//
//   This library is free software; you can redistribute it and/or modify it under
//   the terms of the OpenNETCF.org Shared Source License.
//
//   This library is distributed in the hope that it will be useful, but
//   WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//   FITNESS FOR A PARTICULAR PURPOSE. See the OpenNETCF.org Shared Source License
//   for more details.
//
//   You should have received a copy of the OpenNETCF.org Shared Source License
//   along with this library; if not, email licensing@opennetcf.org to get a copy.
//
//   For general enquiries, email enquiries@opennetcf.org or visit our website at:
//   http://www.opennetcf.org
//
//=========================================================================================
//
//   This file has been modyfied to better fit our needs of the recorder.
//
//   Name: Johan Sverin
//   Date: May 19, 2005
//
//=========================================================================================

using System;
using OpenNETCF.Win32;
using System.Runtime.InteropServices;

namespace OpenNETCF.Multimedia.Audio
{
  /// <summary>
  /// Internal wrapper around WAVEHDR
  /// Facilitates asynchronous operations
  /// </summary>
  internal class WaveHeader: IDisposable
  {
    private WaveHdr m_hdr;
    private IntPtr m_lpData;
    private int m_cbdata;
    private int m_cbHeader;
    private IntPtr m_lpHeader;


    public WaveHeader(byte[] data)
    {
      InitFromData(data, data.Length);
    }

    /// <summary>
    /// Creates WaveHeader and fills it with wave data
    /// <see cref="WaveHdr"/>
    /// </summary>
    /// <param name="data">wave data bytes</param>
    /// <param name="datalength">length of Wave data</param>
    public WaveHeader(byte[] data, int datalength)
    {
      InitFromData(data, datalength);
    }
```

```
    /// <summary>
    /// Constructor for WaveHeader class
    /// Allocates a buffer of required size
    /// </summary>
    /// <param name="BufferSize"></param>
    public WaveHeader(int BufferSize)
    {
      InitFromData(null, BufferSize);
    }

    internal void InitFromData(byte[] data, int datalength)
    {
      m_cbdata = datalength;
      m_lpData = Core.LocalAlloc(Core.MemoryAllocFlags.LMEM_ZEROINIT, m_cbdata);

      if ( data != null )
        Marshal.Copy(data, 0, m_lpData, m_cbdata);

      m_hdr = new WaveHdr((int)m_lpData.ToInt32(), m_cbdata);
      m_cbHeader = m_hdr.ToByteArray().Length;
      m_lpHeader = Core.LocalAlloc(Core.MemoryAllocFlags.LMEM_ZEROINIT, m_cbHeader);
      byte[] hdrbits = m_hdr.ToByteArray();
      Marshal.Copy(hdrbits, 0, m_lpHeader, m_cbHeader);
    }

    ///<summary>Ptr to WAVEHDR in the unmanaged memory</summary>
    public IntPtr Header { get { return m_lpHeader; } }

    ///<summary>Ptr to wave data in the unmanaged memory</summary>
    public IntPtr Data { get { return m_lpData; } }

    ///<summary>Wave data size</summary>
    public int DataLength { get { return m_cbdata; } }
    public int HeaderLength { get { return m_cbHeader; } }
    public WaveHdr waveHdr { get { return m_hdr; } }
    public byte[] GetData()
    {
      byte [] data = new byte[m_cbdata];
      Marshal.Copy(m_lpData, data, 0, m_cbdata);
      return data;
    }
    public void RetrieveHeader()
    {
      byte[] headerBits = new byte[m_cbHeader];
      Marshal.Copy(m_lpHeader, headerBits, 0, m_cbHeader);
      m_hdr = new WaveHdr(headerBits);
    }

    public void Dispose()
    {
      Core.LocalFree(m_lpData);
      Core.LocalFree(m_lpHeader);
    }
  }
}
```

A.2 SpeechRecognizer

A.2.1 SpeechRecognizer.cs

```
//============================================================================
//
//   AdaptiveAudio.SpeechRecognizer
//   Copyright (c) 2005, Johan Sverin
//
//   Version: 1.1
//   Date   : 19/05/2005
//
//   Description:
//   A speech recognizer, that receives RTP packet from the network and
//   do speech recognition on the audio. It uses Command & Control grammar
//   to match the words. A correct command results in a messages that are
//   send to the MediaPlayer.
//
//============================================================================

using System;
using System.IO;
using System.Net;
using System.Collections;
using System.Threading;
using System.Text;
using System.Windows.Forms;

using SpeechLib;
using SIMPLEAUDIOLib;

using Network.Sockets;
using Network.RtpLibrary;
using Network.NetworkingBasics;

namespace AdaptiveAudio.SpeechRecognizer
{

  class SpeechRecognizer
  {
    private static SpAudioPlug AudioPlugIn = null;
    private static SpInprocRecognizer MyRecognizer = null;
    private static SpInProcRecoContext RecoContext = null;
    private static ISpeechRecoGrammar Grammar = null;

    private static SpSharedRecoContext objTrain = new SpSharedRecoContext();

    private const int PAYLOAD_SIZE = 1440;
    private const int HEADER = 16;
    private const int SLEEPTIME = 15;

    private static UDPSocket mySocket = null;
    private static UDPSocket managerSocket = null;

    private static Queue myQueue = new Queue();
    private static bool EndOfStream = false;

    private static ManualResetEvent PacketReceived = new ManualResetEvent(false);

    static void Main(string[] args)
    {
      if (args[0] == "-VoiceTraining")
      {
        TrainUser trainerU = new TrainUser();
        trainerU.OpenTrainer(objTrain);
```

```
    }
    else
    {

    Console.WriteLine("\n  Speech Recognizer v0.1\n");

    try
    {
      IPHostEntry r_HostInfo = Dns.Resolve("PDA");
      IPAddress r_Address = r_HostInfo.AddressList[0];

      IPHostEntry h_HostInfo = Dns.Resolve(Dns.GetHostName());
      IPAddress h_Address = h_HostInfo.AddressList[0];

      mySocket = new UDPSocket(r_Address, 5004, 5003);
      managerSocket = new UDPSocket(h_Address, 50565, 50564);

      Console.WriteLine("  ================================\n");

      Console.WriteLine("  Host IP  :   {0}", h_Address.ToString());
      Console.WriteLine("  Remote IP:   {0}\n", r_Address.ToString());

      Console.WriteLine("  ================================\n");

      Console.WriteLine("  Initializing SAPI reco context object...");
        MyRecognizer = new SpInprocRecognizer();

      Console.WriteLine("  Set input stream to AudioPlugIn...");
        AudioPlugIn = new SpAudioPlug();
        AudioPlugIn.Init( false,
          SIMPLEAUDIOLib.SpeechAudioFormatType.SAFT22kHz16BitMono);
        MyRecognizer.AudioInputStream = (ISpeechBaseStream)AudioPlugIn;

      Console.WriteLine("  Create recocontext...");
        RecoContext = (SpInProcRecoContext)MyRecognizer.CreateRecoContext();

      Console.WriteLine("  Adding events...");
        RecoContext.Recognition += new
        _ISpeechRecoContextEvents_RecognitionEventHandler(RecoContext_Recognition);
        RecoContext.EndStream += new
        _ISpeechRecoContextEvents_EndStreamEventHandler(RecoContext_EndStream);
        RecoContext.EventInterests =
          SpeechLib.SpeechRecoEvents.SRERecognition |
          SpeechLib.SpeechRecoEvents.SREStreamEnd;

      Console.WriteLine("  Setup grammar (Command & Control)...");
        Grammar = RecoContext.CreateGrammar(0);
        Grammar.CmdLoadFromFile("grammar.xml", SpeechLoadOption.SLOStatic);

      Console.WriteLine("\n  ================================\n");

      Thread receiver = new Thread(new ThreadStart(Receiver));
      Thread exit = new Thread(new ThreadStart(Exit));
      receiver.Start();
      exit.Start();

      Console.WriteLine("  Waiting for packets...");
      PacketReceived.WaitOne();

      Recognizer();
    }
    catch(Exception e)
    {
      Console.WriteLine("Error while creating UDP socket");
      Console.WriteLine(e.ToString());
    }
  }
```

```
}

/// <summary>
/// Receives the audio data from the network and
/// adds it to a queue processed by the recognizer
/// </summary>
private static void Receiver()
{
  BufferChunk chunk = null;
  RtpPacket packet = null;

  while(!EndOfStream)
  {
    // Received the audio from the network
    chunk = new BufferChunk(PAYLOAD_SIZE+HEADER);
    mySocket.ReceiveFrom(chunk);
    packet = new RtpPacket(chunk);

    // Enqueue the audio from the network
    Monitor.Enter(myQueue.SyncRoot);
      myQueue.Enqueue( (byte[])packet.Payload );
    Monitor.Exit(myQueue.SyncRoot);

    PacketReceived.Set();

    Thread.Sleep(SLEEPTIME);
  }

}

/// <summary>
/// Works on the queue processing the audio data,
/// to do recognition.
/// </summary>
private static void Recognizer()
{
  byte[] data = new byte[PAYLOAD_SIZE];

  Console.WriteLine("  SpeechRecognition Start\n");

  Grammar.CmdSetRuleIdState(0, SpeechRuleState.SGDSActive);

  do
  {
    if( myQueue.Count < 2 )
    PacketReceived.WaitOne();

    Application.DoEvents();

    Monitor.Enter(myQueue.SyncRoot);
      data = (byte[])myQueue.Dequeue();
    Monitor.Exit(myQueue.SyncRoot);

    AudioPlugIn.SetData( (object)data );

    Thread.Sleep(SLEEPTIME);

    PacketReceived.Reset();

  }while(!EndOfStream);
}


/// <summary>
/// SR engine has reached the end of an input stream.
/// </summary>
/// <param name="StreamNumber">Number of the stream</param>
```

```
    /// <param name="StreamPosition">Position in the stream</param>
    /// <param name="var"></param>
    private static void RecoContext_EndStream(int StreamNumber, object
StreamPosition, bool var)
    {
      Grammar.DictationSetState(SpeechRuleState.SGDSInactive);
      Console.WriteLine("--- END OF STREAM ---");
      EndOfStream = true;
    }


    /// <summary>
    /// SR engine's best hypothesis for the audio data.
    /// </summary>
    /// <param name="StreamNumber">Number of the stream</param>
    /// <param name="StreamPosition">Position in the stream</param>
    /// <param name="RecognitionType">Type of recognition</param>
    /// <param name="e">The result</param>
    private static void RecoContext_Recognition(int StreamNumber, object
StreamPosition, SpeechRecognitionType RecognitionType, ISpeechRecoResult e)
    {
      int actual_confidence = (int)e.PhraseInfo.Elements.Item(0).ActualConfidence;
      float confidence = (float)e.PhraseInfo.Elements.Item(0).EngineConfidence;
      string text = e.PhraseInfo.GetText(0, 1, true);

      if( text.Equals("previous") )
        Console.WriteLine("COMMAND: {0}\t- LEVEL: {1} - CONFIDENCE: {2}", text,
actual_confidence, confidence);
      else
        Console.WriteLine("COMMAND: {0}\t\t- LEVEL: {1} - CONFIDENCE: {2}", text,
actual_confidence, confidence);

      if(confidence >= 0)
      {
        string message = "REQ-04-"+e.PhraseInfo.GetText(0, 1, true)+"-";
        byte[] data = Encoding.ASCII.GetBytes(message);
        managerSocket.SendTo(data);
      }
    }

    private static void Exit()
    {
      Console.ReadLine();
      Application.Exit();
    }

  }
}
```

A.2.2 TrainUser.cs

```
using System;
using SpeechLib;

namespace AdaptiveAudio.SpeechRecognizer
{
  /// <summary>
  /// Summary description for TrainUser.
  /// This class displays the UI for training the learner.
  /// </summary>
  public class TrainUser
  {

    public TrainUser()
    {
    }

    public void OpenTrainer(SpeechLib.SpSharedRecoContext oTrain)
    {

      object str1 = "";
      AuxForm aForm = new AuxForm();

      if (oTrain.Recognizer.IsUISupported("UserTraining", ref str1)==true)
      {
        oTrain.Recognizer.DisplayUI((int)aForm.Handle,
          "SpeechRecognizer","UserTraining", ref str1);
      }
      else
        Console.WriteLine("User Training wizard not supported");
    }
  }
}
```

A.2.3 Aux.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace AdaptiveAudio.SpeechRecognizer
{
  /// <summary>
  /// Summary description for AuxForm.
  /// </summary>
  public class AuxForm : System.Windows.Forms.Form
  {
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.Container components = null;

    public AuxForm()
    {
      //
      // Required for Windows Form Designer support
      //
      InitializeComponent();

      //
      // TODO: Add any constructor code after InitializeComponent call
      //
```

```
    }

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    protected override void Dispose( bool disposing )
    {
      if( disposing )
      {
        if(components != null)
        {
          components.Dispose();
        }
      }
      base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
      this.components = new System.ComponentModel.Container();
      this.Size = new System.Drawing.Size(300,300);
      this.Text = "AuxForm";
    }
    #endregion
  }
}
```

A.3 Common

A.3.1 UdpSocket.cs

```
//============================================================================
//
//   Network.Sockets
//   Copyright (c) 2005, Johan Sverin
//
//   Version: 1.4
//   Date   : 19/05/2005
//
//   Description:
//   Implements a asynchronous UDP socket.
//   Use Receive and Send, as well as ReceiveFrom and SendTo.
//
//============================================================================

using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Text;

using Network.NetworkingBasics;

namespace Network.Sockets
{

  public class UDPSocket : IDisposable
  {
    private Socket m_Socket = null;

    private int localport = 0;
    private int remoteport = 0;
    private IPAddress address = null;

    /// <summary>
    /// Millisecond delay to add between packet sends, used to govern network
    /// throughput on limited networks such as 802.11b
    /// </summary>
    private short delayBetweenPackets = 0;

    /// <summary>
    /// Initializes a new instance of the UDPSocket class.
    /// It creates a socket with at a given address and port.
    /// </summary>
    /// <param name="address"></param>
    /// <param name="port"></param>
    public UDPSocket(IPAddress address, int localport, int remoteport)
    {
      this.address = address;
      this.localport = localport;
      this.remoteport = remoteport;

      EndPoint EP = new IPEndPoint(address, localport);

      m_Socket = new Socket(EP.AddressFamily, SocketType.Dgram, ProtocolType.Udp);
      m_Socket.Bind( new IPEndPoint(IPAddress.Any, localport) );
    }


    /// <summary>
    /// This method closes and releases all the resources associated to the socket.
    /// </summary>
```

```
public void Dispose()
{
  m_Socket.Close();
  m_Socket = null;
}

/// <summary>
///
/// </summary>
/// <param name="data"></param>
public void SendTo(BufferChunk data)
{
  try
  {
    EndPoint myEndPoint = new IPEndPoint(address, remoteport);

    m_Socket.SendTo(data.Buffer, data.Index, data.Length,
      SocketFlags.None, myEndPoint);
    if(delayBetweenPackets != 0)
    {
      // To control bandwidth we add some sleep
      Thread.Sleep(delayBetweenPackets);
    }
  }
  catch(SocketException e) { Console.WriteLine(e.Message); }
  catch(Exception e) { Console.WriteLine(e.Message); }
}

/// <summary>
///
/// </summary>
/// <param name="data"></param>
public void SendTo(byte[] data)
{
  try
  {
    EndPoint myEndPoint = new IPEndPoint(address, remoteport);

    m_Socket.SendTo(data, 0, data.Length, SocketFlags.None, myEndPoint);
    if(delayBetweenPackets != 0)
    {
      // To control bandwidth we add some sleep
      Thread.Sleep(delayBetweenPackets);
    }
  }
  catch(SocketException e) { Console.WriteLine(e.Message); }
  catch(Exception e) { Console.WriteLine(e.Message); }
}

/// <summary>
///
/// </summary>
/// <param name="buffer"></param>
/// <param name="n_EndPoint"></param>
public void ReceiveFrom(BufferChunk buffer)
{
  try
  {
    EndPoint myEndPoint = new IPEndPoint(IPAddress.Any, remoteport);
    buffer.Length = m_Socket.ReceiveFrom(buffer.Buffer, 0,
      buffer.Buffer.Length, SocketFlags.None, ref myEndPoint);
  }
  catch(SocketException e) { Console.WriteLine(e.Message); }
  catch(Exception e) { Console.WriteLine(e.Message); }
}

/// <summary>
```

```
    ///
    /// </summary>
    /// <param name="buffer"></param>
    /// <param name="n_EndPoint"></param>
    public void ReceiveFrom(byte[] buffer)
    {
      try
      {
        EndPoint myEndPoint = new IPEndPoint(IPAddress.Any, remoteport);
        m_Socket.ReceiveFrom(buffer, SocketFlags.None, ref myEndPoint);
      }
      catch(SocketException e) { Console.WriteLine(e.Message); }
      catch(Exception e) { Console.WriteLine(e.Message); }
    }


    /// <summary>
    /// Sets or gets the delay between packets
    /// </summary>
    public short DelayBetweenPackets
    {
      get
      {
        return delayBetweenPackets;
      }
      set
      {
        if ( delayBetweenPackets < 0 || delayBetweenPackets > 30 )
        {
          throw new ArgumentException("Must be in the range of 0 to 30");
        }

        delayBetweenPackets = value;
      }
    }

  }
}
```

A.3.2 RtpPacket.cs

```
using System;

using Network.NetworkingBasics;

namespace Network.RtpLibrary
{

  public enum PayloadType : byte
  {
    PCMU = 0, PT1016, G721, GSM,
    DVI4 = 5,
    LPC = 7, PCMA, G722, L16,
    MPA = 14, G728,
    CelB = 25, JPEG,
    nv = 28,
    H261 = 31, MPV, MP2T,
    // 96-127 are intended for dynamic assignment
    xApplication1 = 96,
    xApplication2, xApplication3, xApplication4, xApplication5, xApplication6,
    xApplication7, xApplication8, xApplication9, xApplication10,
    Venue1 = 106,
    Venue2, Venue3, Venue4, Venue5, Venue6, Venue7, Venue8, Venue9, Venue10,
    Reserved1 = 116,
    Reserved2, Reserved3, Reserved4, Reserved5,
    RTDocument = 121,
    PipecleanerSignal = 122,
    Reserved6 = 123,
    FEC = 124,
    dynamicPresentation = 125,
    dynamicVideo = 126,
    dynamicAudio = 127
  }

  public class RtpPacketBase
  {
    #region Statics

    internal const int RTP_HEADER_SIZE = SSRC_INDEX + SSRC_SIZE;

    internal const int VERSION = 2;

    private const int VPXCC_SIZE = 1;
    private const int MPT_SIZE = 1;
    private const int SEQ_SIZE = 2;
    private const int TS_SIZE = 4;
    protected const int SSRC_SIZE = 4;

    private const int VPXCC_INDEX = 0;
    private const int MPT_INDEX = VPXCC_INDEX + VPXCC_SIZE;
    private const int SEQ_INDEX = MPT_INDEX + MPT_SIZE;
    protected const int TS_INDEX = SEQ_INDEX + SEQ_SIZE;
    protected const int SSRC_INDEX = TS_INDEX + TS_SIZE;

    private const int MTU = 1500;
    private const int IP_Header = 20;
    private const int UDP_Header = 4;
    public const int MAX_PACKET_SIZE = MTU - IP_Header - UDP_Header;

    /// <summary>
    ///  Cast operator for forming a BufferChunk from an RtpPacketBase.
    /// </summary>
    public static explicit operator BufferChunk(RtpPacketBase packet)
    {
      return packet.buffer;
```

```
}

/// <summary>
///  Cast operator for forming a BufferChunk from an RtpPacketBase.
/// </summary>
public static explicit operator RtpPacketBase(BufferChunk buffer)
{
  return new RtpPacketBase(buffer);
}

/// <summary>
///  Buffer to contain the raw data
/// </summary>
public BufferChunk buffer;

/// <summary>
/// Creates a max size packet
/// </summary>
internal RtpPacketBase() : this(MAX_PACKET_SIZE) {}

/// <summary>
/// Creates a packet of the given size
/// </summary>
internal RtpPacketBase(int packetSize)
{
  buffer = new BufferChunk(new byte[packetSize]);
  Reset();
}

/// <summary>
/// Create a packet from an existing buffer
/// </summary>
/// <param name="buffer"></param>
internal RtpPacketBase(BufferChunk buffer)
{
  ValidateBuffer(buffer);

  this.buffer = buffer;
}

/// <summary>
/// Create a packet from an existing packet
/// </summary>
/// <param name="packet"></param>
internal RtpPacketBase(RtpPacketBase packet)
{
  buffer = packet.buffer;
}

/// <summary>
/// Marker reserved for payload/protocol specific information.
/// </summary>
internal bool Marker
{
  get{return ((buffer[MPT_INDEX] & 128) == 128);}

  set
  {
    if(value)
    {
      // Set it
      buffer[MPT_INDEX] |= (byte)(128);
    }
    else
    {
      // Clear the bit
      buffer[MPT_INDEX] ^= (byte)(buffer[MPT_INDEX] & 128);
```

```
    }
  }
}

/// <summary>
/// The type of data contained in the packet
/// </summary>
internal PayloadType PayloadType
{
  get{return (PayloadType)(buffer[MPT_INDEX] & 127);}

  set
  {
    if ((int)value > 127)
    {
      throw new ArgumentOutOfRangeException("PayloadType" +
        " is a seven bit structure, and can hold values between 0 and 127");
    }

    // Preserve most significant bit
    buffer[MPT_INDEX] = (byte)(buffer[MPT_INDEX] & 128);
    buffer[MPT_INDEX] += (byte)value;
  }
}

/// <summary>
/// Sequence number of the packet, used to keep track of the
/// order packets were sent in
///</summary>
public ushort Sequence
{
  get{return buffer.GetUInt16(SEQ_INDEX);}
  set{buffer.SetUInt16(SEQ_INDEX, value);}
}

/// <summary>
/// In our implementation, it is an incrementing counter
/// used to group packets into a frame
/// </summary>
internal virtual uint TimeStamp
{
  get{return buffer.GetUInt32(TS_INDEX);}
  set{buffer.SetUInt32(TS_INDEX, value);}
}

/// <summary>
/// Synchronization source used to identify streams within a session
/// </summary>
public uint SSRC
{
  get{return buffer.GetUInt32(SSRC_INDEX);}
  set{buffer.SetUInt32(SSRC_INDEX, value);}
}

/// <summary>
/// Payload data of the RtpPacket
/// </summary>
internal BufferChunk Payload
{
  set
  {
    // Make sure they haven't tried to add more data than we can handle
    if(value.Length > MaxPayloadSize)
    {
      throw new ArgumentOutOfRangeException("The maximum " +
        "payload for this packet is: " + MaxPayloadSize);
    }
```

```
      // Reset Buffer to just after the header because packets are re-used and so
      // that operator+ works properly when copying the payload
      buffer.Reset(0, HeaderSize);
      buffer += value;
    }
    get
    {
      return buffer.Peek(HeaderSize, PayloadSize);
    }
  }

  /// <summary>
  /// How much payload data can this packet accept
  /// </summary>
  internal int MaxPayloadSize
  {
    get{return buffer.Buffer.Length - HeaderSize;}
  }

  /// <summary>
  /// Release the BufferChunk held by this packet so it can be
  /// reused outside the scope of this packet.
  /// </summary>
  internal BufferChunk ReleaseBuffer()
  {
    BufferChunk ret = buffer;
    buffer = null;

    return ret;
  }

  internal virtual int HeaderSize
  {
    get{return RTP_HEADER_SIZE;}
  }

  internal BufferChunk Buffer
  {
    get{return buffer;}
  }

  internal virtual int PayloadSize
  {
    get
    {
      int size = buffer.Length - HeaderSize;

      return size;
    }
    set
    {
      buffer.Reset(0, HeaderSize + value);
    }
  }

  /// <summary>
  ///
  /// </summary>
  internal virtual void Reset()
  {
    buffer.Reset(0, HeaderSize);
    buffer.Clear();

    // Initialize the first byte: V==2, P==0, X==0, CC==0
    buffer[VPXCC_INDEX] = (byte)(VERSION << 6);
  }
```

```
  /// <summary>
  ///
  /// </summary>
  private void ValidateBuffer(BufferChunk buffer)
  {
    int version = buffer[VPXCC_INDEX] >> 6;

    if (version != VERSION)
      Console.WriteLine(string.Format(
        "Invalid version: {0}, current: {1}", version, VERSION));
  }

}

public class RtpPacket : RtpPacketBase
{

  #region Statics

  /// <summary>
  /// We use a fixed size header extension
  /// </summary>
  public const int HEADER_EXTENSIONS_SIZE =
    PACKETS_IN_FRAME_SIZE + FRAME_INDEX_SIZE;

  private const int PACKETS_IN_FRAME_SIZE = 2;
  private const int FRAME_INDEX_SIZE = 2;

  internal RtpPacket() : base() {}

  internal RtpPacket(int packetSize) : base(packetSize){}

  internal RtpPacket(BufferChunk buffer) : base(buffer){}

  internal RtpPacket(RtpPacketBase packet) : base(packet){}

  internal ushort PacketsInFrame
  {
    get{return buffer.GetUInt16(PacketsInFrame_Index);}
    set{buffer.SetUInt16(PacketsInFrame_Index, value);}
  }

  internal ushort FrameIndex
  {
    get
    {
      return buffer.GetUInt16(FrameIndex_Index);
    }
    set
    {
      buffer.SetUInt16(FrameIndex_Index, value);
    }
  }

  internal override int HeaderSize
  {
    get
    {
      return base.HeaderSize + HEADER_EXTENSIONS_SIZE;
    }
  }

  private int PacketsInFrame_Index
  {
    get{return base.HeaderSize;}
  }
```

```
    private int FrameIndex_Index
    {
      get{return PacketsInFrame_Index + PACKETS_IN_FRAME_SIZE;}
    }

    private int FecIndex_Index
    {
      get{return FrameIndex_Index + FRAME_INDEX_SIZE;}
    }
  }
}
```

A.4 Manager

A.4.1 Manager.cs

```
using System;
using System.Net;
using System.Text;
using System.Threading;
using System.IO;
using System.Xml;
using System.Diagnostics;

using Network.Sockets;

namespace AdaptiveAudio.Manager
{
  class Manager
  {
    private static UDPSocket mainSocket = null;
    private static UDPSocket senderSocket = null;
    private static UDPSocket recognizerSocket = null;
    private static UDPSocket ttsSocket = null;

    private static Process SpeechProcess = null;

    private const int PDA_PORT_R = 50560;
    private const int PDA_PORT_L = 50561;
    private const int SENDER_PORT_L = 50562;
    private const int SENDER_PORT_R = 50563;
    private const int RECOGNIZER_PORT_L = 50564;
    private const int RECOGNIZER_PORT_R = 50565;
    private const int TTS_PORT_L = 50566;
    private const int TTS_PORT_R = 50567;

    private static bool exit = false;

    public static string FilePath = @"D:\Visual\Final Code\Playlists\";
    public static string SpeechPath =
      @"D:\Visual\Final Code\SpeechRecognizer\bin\debug\SpeechRecognizer.exe";

    private enum MESSAGE
    {
      REQ_CLOSE = 0,
      REQ_INFO = 1,
      REQ_FILESENDER = 2,
      REQ_NETWORK_STATE = 3,
      REQ_COMMAND = 4,
      REQ_AUDIOALERT = 5,
      ACK_NO_SUCH_FILE = 6,
      ACK_NO_XML_FILE = 7,
      ACK_FILE_INFO = 8,
      ACK_OK = 9,
      ACK_WAIT = 10,
      ACK_FINISHED = 11,
      REQ_RECOGNIZER_START = 12,
      REQ_RECOGNIZER_STOP = 13
    }

    static void Main(string[] args)
    {
      Console.WriteLine("Manager v1.2\n");

      IPHostEntry r_HostInfo = Dns.Resolve("PDA");
      IPAddress r_address = r_HostInfo.AddressList[0];
```

```
IPHostEntry l_HostInfo = Dns.Resolve("unnamed");
IPAddress l_address = l_HostInfo.AddressList[0];

mainSocket = new UDPSocket(r_address, PDA_PORT_L, PDA_PORT_R);
senderSocket = new UDPSocket(l_address, SENDER_PORT_L, SENDER_PORT_R);
recognizerSocket =
  new UDPSocket(l_address, RECOGNIZER_PORT_L, RECOGNIZER_PORT_R);
ttsSocket = new UDPSocket(l_address, TTS_PORT_L, TTS_PORT_R);

Console.WriteLine("Connected...\n");

Thread sender = new Thread( new ThreadStart(FileSender) );
Thread recognizer = new Thread( new ThreadStart(Recognizer) );
sender.Start();
recognizer.Start();

string text, message = "";
int code, index, stop;

while(!exit)
{
  byte[] data = new byte[128];

  mainSocket.ReceiveFrom(data);

  text = Encoding.ASCII.GetString(data);
  index = text.IndexOf("-", 0)+1;
  code = Convert.ToInt32( text.Substring(index, 2) );

  index = text.IndexOf("-", index)+1;
  stop = Convert.ToInt32( text.IndexOf("-",index) );

  if(stop>0)
    message = text.Substring(index, stop-index);

  switch(code)
  {
    case (int)MESSAGE.REQ_CLOSE:
      Console.WriteLine("REQ: close");
      exit = true;
      break;

    case (int)MESSAGE.REQ_INFO:
      Console.WriteLine("REQ: information of '{0}'", message);
      ProcessFile(message);
      break;

    case (int)MESSAGE.REQ_FILESENDER:
      Console.WriteLine("REQ: start filesender with '{0}'", message);
      senderSocket.SendTo(data);
      break;

    case (int)MESSAGE.REQ_AUDIOALERT:
      Console.WriteLine("REQ: audioalert");
      ttsSocket.SendTo(data);
      break;

    case (int)MESSAGE.ACK_NO_SUCH_FILE:
      Console.WriteLine("ACK: no such file");
      mainSocket.SendTo(data);
      break;

    case (int)MESSAGE.ACK_NO_XML_FILE:
      Console.WriteLine("ACK: no xml file");
      mainSocket.SendTo(data);
      break;
```

```
      case (int)MESSAGE.ACK_FILE_INFO:
        Console.WriteLine("ACK: file information");
        break;

      case (int)MESSAGE.ACK_OK:
        Console.WriteLine("ACK: state of network: OK");
        senderSocket.SendTo(data);
        break;

      case (int)MESSAGE.ACK_WAIT:
        Console.WriteLine("ACK: state of network: WAIT");
        mainSocket.SendTo(data);
        break;

      case (int)MESSAGE.REQ_RECOGNIZER_START:
        Console.WriteLine("REQ: Start SpeechRecognizer");
        SpeechProcess = Process.Start(SpeechPath);
        break;

      case (int)MESSAGE.REQ_RECOGNIZER_STOP:
        Console.WriteLine("REQ: Stop SpeechRecognizer");
        if(SpeechProcess!=null)
        {
          SpeechProcess.Close();
          SpeechProcess = null;
        }
        break;

      default:
        Console.WriteLine("ERROR IN MESSAGE");
        break;
    }

  }
}

private static void FileSender()
{
  while(!exit)
  {
    byte[] data = new byte[128];

    senderSocket.ReceiveFrom(data);

    string message = Encoding.ASCII.GetString(data);

    int index = message.IndexOf("-", 0)+1;
    int code = Convert.ToInt32( message.Substring(index,2) );

    switch(code)
    {
      case (int)MESSAGE.REQ_NETWORK_STATE:
        Console.WriteLine("REQ: state of network");
        mainSocket.SendTo(data);
        break;

      case (int)MESSAGE.ACK_FINISHED:
        Console.WriteLine("ACK: finished filetransfer");
        mainSocket.SendTo(data);
        break;

      case (int)MESSAGE.REQ_CLOSE:
        Console.WriteLine("REQ: close");
        mainSocket.SendTo(data);
        exit = true;
        break;
    }
```

```
  }
}

private static void Recognizer()
{
  while(!exit)
  {
    byte[] data = new byte[128];

    recognizerSocket.ReceiveFrom(data);

    string message = Encoding.ASCII.GetString(data);

    int index = message.IndexOf("-", 0)+1;
    int code = Convert.ToInt32( message.Substring(index,2) );

    index = message.IndexOf("-", index)+1;
    int stop = Convert.ToInt32( message.IndexOf("-",index) );

    if(stop>0)
      message = message.Substring(index, stop-index);

    switch(code)
    {
      case (int)MESSAGE.REQ_COMMAND:
        Console.WriteLine("REQ: command '{0}'", message);
        mainSocket.SendTo(data);
        break;
    }
  }
}

/// <summary>
/// Process an xml file to obtain the total amount of Mb and
/// minutes of audio content listed at the playlist.
/// Then send the information to the Player.
/// </summary>
/// <param name="FileName"></param>
public static void ProcessFile(string FileName)
{
  int index;
  string text, extension;
  byte[] data = new byte[128];

  if (!File.Exists(FilePath+FileName))
  {
    text = "ACK-06"+(int)MESSAGE.ACK_NO_SUCH_FILE+"-";

    Console.WriteLine("ACK: no such file");

    data = Encoding.ASCII.GetBytes(text);
    mainSocket.SendTo(data);
  }
  else
  {
    index = FileName.LastIndexOf(".")+1;

    extension = FileName.Substring(index, FileName.Length - index);

    if (extension != "xml")
    {
      text = "ACK-07"+(int)MESSAGE.ACK_NO_XML_FILE+"-";

      Console.WriteLine("ACK: no xml file");

      data = Encoding.ASCII.GetBytes(text);
      mainSocket.SendTo(data);
```

```
        }
        else
        {
          double mb = 0;
          double min = 0;
          double aux;
          string time = "";
          string size = "";

          XmlTextReader reader = new XmlTextReader(FilePath+FileName);

          try
          {
            while(reader.Read())
            {
              switch (reader.Name)
              {
                case "duration":
                          time = reader.GetAttribute("durationID");
                          aux = Convert.ToDouble(time);
                          min = min + aux;
                          break;

                case "fileSize":
                          size = reader.GetAttribute("fileSizeID");
                          aux = Convert.ToDouble(size);
                          mb = mb + aux;
                          break;
              }
            }

            text = "ACK-0"+(int)MESSAGE.ACK_FILE_INFO+"-"+mb+"-"+min+"-";

            Console.WriteLine("ACK: file information");

            data = Encoding.ASCII.GetBytes(text);
            mainSocket.SendTo(data);
          }
          catch(Exception ex)
          {
            Console.WriteLine(ex.ToString());
          }
        }
      }
    }
  }
}
```

A.5 MediaPlayer

A.5.1 MediaPlayer.cs

```
//=========================================================================================
//
//   AdaptiveAudio.MediaPlayer
//   Copyright (c) 2005, Inmaculada Rangel Vacas
//
//   Version: 1.1
//   Date   : 20/02/2005
//
//   Description:
//   Asks for XML file, reads it and put the content into a listbox. Then you
//   are able to play the tracks, go to the previos one, next one and stop them.
//   Also the user can ask for additional content, in which case the application
//   will ask for a file name, send a request for information about that file to
//   the Manager and process the answer according to the current conte
//   information. If the results are favorable, the new content will be downloaded
//   from the laptop.
//
//=========================================================================================
//
//   Name: Johan Sverin
//   Date: 19/05/2005
//
//   * Code has been optimized and changed to use the new UdpSocket.cs
//   * Added SpeechRecognizer functions
//
//=========================================================================================

using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
using System.Threading;
using System.IO;
using System.Xml;
using System.Text;
using System.Net;
using System.Diagnostics;

using OpenNETCF.IO;
using OpenNETCF.Net;
using OpenNETCF.ToolHelp;

using Network.Sockets;

namespace AdaptiveAudio.MediaPlayer
{
  /// <summary>
  /// Summary description for MainForm.
  /// </summary>
  public class MainForm : System.Windows.Forms.Form
  {
    // Define buttons
    private System.Windows.Forms.Button stopButton;
    private System.Windows.Forms.Button playButton;
    private System.Windows.Forms.Button nextButton;
    private System.Windows.Forms.Button exitButton;
    private System.Windows.Forms.Button prevButton;
    private System.Windows.Forms.Button loadButton;
    private System.Windows.Forms.Button speechButton;
```

```
private System.Windows.Forms.OpenFileDialog openFileDialog;
private System.Windows.Forms.MainMenu mainMenu1;

// Define application paths
public static string PDApath = @"\iPAQ File Store\";
public static string progPath =
  @"\Program Files\WaveAudioPlayer\WaveAudioPlayer.exe";

public static string currentTrack;
public static string tmpcurrentTrack;
public static string fileName;
public static string linkQuality;
public static string rssi;

public static double playTime;
public static double sendTime;
public static int playListLenght;

public static string filler = "                                                    ";
private System.Windows.Forms.Timer timerFinished;
private System.Windows.Forms.Button requestButton;
private System.Windows.Forms.SaveFileDialog saveFileDialog;

// Define buffers
public static byte[] receiveBuffer = new byte[128];
public static byte[] sendBuffer = new byte[128];

// Define UDP sockets
public static UDPSocket managerSocket = null;

// Define exit flags
public static bool exitFlag_ProcessAnswer = false;
public static bool exitFlag_NetworkState = false;

// Define battery status
public static Battery.SYSTEM_POWER_STATUS_EX status =
  new Battery.SYSTEM_POWER_STATUS_EX();
public static Battery.SYSTEM_POWER_STATUS_EX2 status2 =
  new Battery.SYSTEM_POWER_STATUS_EX2();

public static AudioAlerts audioAlertForm;

public struct alertStruct
{
  public string time;
  public string fileName;
}

public static ArrayList alerts = new ArrayList();

private static bool downloading = false;

private System.Windows.Forms.ContextMenu contextMenu1;
private System.Windows.Forms.MenuItem menuItem1;
private System.Windows.Forms.MenuItem menuItem2;
private System.Windows.Forms.MenuItem menuItem4;
private System.Windows.Forms.Button alertButton;
private System.Windows.Forms.Timer timer2;
private System.Windows.Forms.Timer timer3;

public static int inProcess = 0;

public class ProcessInfo
{
  public IntPtr hProcess;
  public IntPtr hThread;
```

```
    public Int32 ProcessID;
    public Int32 ThreadId;
  }

  public const int LOCAL_PORT = 50560;
  public const int REMOTE_PORT = 50561;

  private System.Windows.Forms.ListBox listBox;

  #region --- Dll Imports ---

  [DllImport("CoreDll.Dll", SetLastError=true)]
  private extern static int CreateProcess(
    string imageName,
    string cmdLine,
    IntPtr lpProcessAttributes,
    IntPtr lpThreadAttributes,
    Int32 boolInheritHandles,
    Int32 dwCreationFlags,
    IntPtr lpEnvironment,
    IntPtr lpszCurrentDir,
    byte [] si,
    ProcessInfo pi);

  [DllImport("CoreDll.Dll")]
  private extern static Int32 GetLastError();

  [DllImport("CoreDll.Dll")]
  private extern static Int32 WaitForSingleObject(IntPtr Handle, Int32 Wait);

  [DllImport("coredll",EntryPoint="FindWindow")]
  public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);

  [DllImport("coredll",EntryPoint="SetForegroundWindow")]
  public static extern bool SetForegroundWindow(IntPtr hWnd);

  [DllImport("coredll",EntryPoint="DestroyWindow")]
  public static extern bool DestroyWindow(IntPtr hWnd);

  #endregion

  /// <summary>
  /// Creates a process using the WINAPI CreateProcess function.
  /// </summary>
  /// <param name="ExeName">The name of the executable to run</param>
  /// <param name="CmdLine">Any commandline parameters</param>
  /// <param name="pi">Returns a ProcessInfo object with details about the
  /// created process</param>
  /// <returns></returns>
  public static bool CreateProcess(String ExeName, String CmdLine, ProcessInfo pi)
  {
    if (pi == null)
      pi = new ProcessInfo();

    byte [] si = new byte[128];

    CreateProcess(ExeName, CmdLine, IntPtr.Zero, IntPtr.Zero,
      0, 0, IntPtr.Zero, IntPtr.Zero, si, pi);

    return true;
  }

  /// <summary>
  /// Starts the pocket media player. If it is alreday running it
  /// brings it to the foreground.
  /// </summary>
  /// <param name="param"></param>
```

```
private void LaunchPocketWMP()
{
  //Invoke windows media player. First check
  //what version of the Pocket OS is running
  //as the name of the pocket media player exe
  //has changed.
  string progPath;
  OperatingSystem os = System.Environment.OSVersion;
  if (os.Version.Major > 3)
    progPath = "wmplayer.exe";
  else
    progPath = "player.exe";

  ProcessInfo pi = new ProcessInfo();
  CreateProcess(progPath, currentTrack, pi);
}

/// <summary>
/// Starts the audio player. If it is alreday running it
/// brings it to the foreground.
/// </summary>
/// <param name="param"></param>
public static void LaunchWaveAudioPlayer()
{
  //Invoke WaveAudioPlayer.

  ProcessInfo pi = new ProcessInfo();
  currentTrack = '"' + currentTrack + '"';

  CreateProcess(progPath, currentTrack, pi);
}

/// <summary>
/// Exit from the application
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void exitButton_Click(object sender, System.EventArgs e)
{
  string programWM = "Windows Media";
  string programAP = "Command Prompt";

  IntPtr hWndWM, hWndAP;

  exitFlag_ProcessAnswer = true;

  sendBuffer = Encoding.ASCII.GetBytes("REQ-00-");
  managerSocket.SendTo(sendBuffer);

  if (downloading)
  {
    exitFlag_NetworkState = true;
    managerSocket.SendTo(sendBuffer);
  }

  // Find the Windows Media window and destroy it
  hWndWM = FindWindow(null,programWM);
  if(!hWndWM.Equals(IntPtr.Zero))
  {
    DestroyWindow(hWndWM);
  }

  // Find the Command Prompt window and destroy it
  hWndAP = FindWindow(null,programAP);
  if(!hWndAP.Equals(IntPtr.Zero))
  {
    DestroyWindow(hWndAP);
```

```
    }

    Application.Exit();
  }

  /// <summary>
  /// Load XML file and extract the elements of the playList
  /// to show them at the listView.
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void loadButton_Click(object sender, System.EventArgs e)
  {
    string file, extension;
    int index;
    double sec;

    listBox.Items.Clear();
    MessageBox.Show("Please, select a XML file");
    openFileDialog.ShowDialog();
    file = openFileDialog.FileName;

    if (file == "")
    {
      MessageBox.Show("No XML file selected");
    }
    else
    {
      index = file.LastIndexOf(".")+1;
      extension = file.Substring(index, (file.Length-index) );

      if (extension != "xml")
      {
        MessageBox.Show(file + " is not a valid file");
        return;
      }
      else
      {
        try
        {
          XmlTextReader reader = new XmlTextReader(file);

          string track = "";
          string type = "";
          string trackComplete = "";
          string location = "";
          string trackInfo = "";
          string time = "";

          while(reader.Read())
          {
            switch(reader.Name)
            {
              case "title":
                track = reader.GetAttribute("titleID");
                break;

              case "fileType":
                type = reader.GetAttribute("fileTypeID");
                break;

              case "fileName":
                trackComplete = reader.GetAttribute("fileNameID");
                break;

              case "sourcePDA":
                location = reader.GetAttribute("sourcePDAID");
```

```
            break;

          case "duration":
            time = reader.GetAttribute("durationID");
            break;

          default:
            break;
        }

        if ((track != "") && (type != "") && (location != "")
          && (trackComplete != "") && (time !=""))
        {
          // Convert the time to play from minutes to seconds
          sec = Convert.ToDouble(time)*60;

          trackInfo = track + filler + type + "  " + location +
            @"\" + trackComplete + "  " + sec.ToString();

          listBox.Items.Add(trackInfo);
          track = "";
          type = "";
          location = "";
          trackComplete = "";
          time = "";
        }
      }
      listBox.SelectedIndex = 0;
      playButton.Enabled = true;
      speechButton.Enabled = true;
      menuItem1.Enabled = true;
      menuItem2.Enabled = true;
      menuItem4.Enabled = true;
      playListLenght = listBox.Items.Count;
    }

    catch(Exception ex)
    {
      MessageBox.Show(ex.ToString());
    }
  }
 }
}

/// <summary>
/// Play the content of the listBox from the selected item.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void playButton_Click(object sender, System.EventArgs e)
{
 play();
}

private void play()
{
 int selectedIndex, index, index2;
 string selectedTrack, type;

 playButton.Enabled = false;
 stopButton.Enabled = true;
 prevButton.Enabled = true;
 nextButton.Enabled = true;
 loadButton.Enabled = false;
 menuItem1.Enabled = false;
 menuItem2.Enabled = false;
 menuItem4.Enabled = false;
```

```
    listBox.Enabled = false;

    selectedIndex = listBox.SelectedIndex;
    selectedTrack = listBox.SelectedItem.ToString();

    index = selectedTrack.IndexOf(filler)+50;
    index2 = selectedTrack.IndexOf("  ",index);
    type = selectedTrack.Substring(index,index2-index);

    index = index2 + 2;
    index2 = selectedTrack.IndexOf("  ",index);
    currentTrack = selectedTrack.Substring(index,index2-index);

    index = index2 + 2;
    index2 = selectedTrack.Length;
    playTime = Convert.ToDouble(selectedTrack.Substring(index, index2-index));

    timerFinished.Interval = (int)playTime*1000;
    timerFinished.Enabled = true;

    switch(type)
    {
      case "mp3":
        goto case "wma";

      case "wma":
        Thread threadWMA = new Thread(new ThreadStart(LaunchPocketWMP));
        threadWMA.Start();
        break;

      case "wav":
        Thread threadWAV = new Thread(new ThreadStart(LaunchWaveAudioPlayer));
        threadWAV.Start();
        break;
    }

    try
    {
      //Bring application to the front
      IntPtr hWnd = FindWindow(null,this.Text);
      if(!hWnd.Equals(IntPtr.Zero))
        SetForegroundWindow(hWnd);
    }
    catch
    {
      MessageBox.Show("Error while bringing application to front");
    }
  }

  /// <summary>
  /// Timer ticks each time a song finish being played
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void timerFinished_Tick(object sender, System.EventArgs e)
  {
    int selectedIndex;
    int index;
    int index2;
    string selectedTrack;
    string type;

    selectedIndex = listBox.SelectedIndex;

    if (selectedIndex == playListLenght-1)
    {
      timerFinished.Enabled = false;
```

```
      playButton.Enabled = true;
      loadButton.Enabled = true;
      stopButton.Enabled = false;
      prevButton.Enabled = false;
      nextButton.Enabled = false;
    }
    else
    {
      listBox.SelectedIndex = listBox.SelectedIndex + 1;
      selectedTrack = listBox.SelectedItem.ToString();

      index = selectedTrack.IndexOf(filler)+50;
      index2 = selectedTrack.IndexOf("  ",index);
      type = selectedTrack.Substring(index,index2-index);

      index = index2 + 2;
      index2 = selectedTrack.IndexOf("  ",index);
      currentTrack = selectedTrack.Substring(index,index2-index);

      index = index2 + 2;
      index2 = selectedTrack.Length;
      playTime = Convert.ToDouble(selectedTrack.Substring(index, index2-index));

      timerFinished.Interval = (int)playTime*1000;
      timerFinished.Enabled = true;

      switch(type)
      {
        case "mp3":
          goto case "wma";

        case "wma":
          Thread threadWMP = new Thread(new ThreadStart(LaunchPocketWMP));
          threadWMP.Start();
          break;

        case "wav":
          Thread threadWAV = new Thread(new ThreadStart(LaunchWaveAudioPlayer));
          threadWAV.Start();
          break;
      }

      //Bring application to the front
      try
      {
        IntPtr hWnd = FindWindow(null,this.Text);
        if(!hWnd.Equals(IntPtr.Zero))
          SetForegroundWindow(hWnd);
      }
      catch
      {
        MessageBox.Show("Error while bringing application to front");
      }
    }
  }

  /// <summary>
  /// Stop the audio
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void stopButton_Click(object sender, System.EventArgs e)
  {
    stop();
  }

  private void stop()
```

```
{
  string programWM = "Windows Media";
  string programAP = "Command Prompt";

  stopButton.Enabled = false;
  loadButton.Enabled = true;
  playButton.Enabled = true;
  prevButton.Enabled = true;
  nextButton.Enabled = true;
  menuItem1.Enabled = true;
  menuItem2.Enabled = true;
  menuItem4.Enabled = true;
  listBox.Enabled = true;

  IntPtr hWndWM, hWndAP;

  // Find the Windows Media window and destroy it
  hWndWM = FindWindow(null,programWM);
  if(!hWndWM.Equals(IntPtr.Zero))
  {
    DestroyWindow(hWndWM);
  }

  // Find the WaveAudioPlayer window and destroy it
  hWndAP = FindWindow(null,programAP);
  if(!hWndAP.Equals(IntPtr.Zero))
  {
    DestroyWindow(hWndAP);
  }
}

/// <summary>
/// Play the previous track in the playlist.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void prevButton_Click(object sender, System.EventArgs e)
{
  previous();
}

private void previous()
{
  int selectedIndex;

  selectedIndex = listBox.SelectedIndex;

  if (selectedIndex != 0)
  {
    stop();
    selectedIndex = selectedIndex - 1;
    listBox.SelectedIndex = selectedIndex;

    play();
  }
}

/// <summary>
///  Play the next track in the playlist.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void nextButton_Click(object sender, System.EventArgs e)
{
  next();
}
```

```
private void next()
{
  int selectedIndex;
  int total;

  selectedIndex = listBox.SelectedIndex;
  total = listBox.Items.Count;

  if (selectedIndex != (total - 1))
  {
    stop();
    selectedIndex = selectedIndex + 1;
    listBox.SelectedIndex = selectedIndex;
    play();
  }
}

private void speechButton_Click(object sender, System.EventArgs e)
{
  if(this.speechButton.Text=="Enable CnC")
  {
    this.speechButton.Text = "Disable CnC";
    ProcessInfo pi = new ProcessInfo();
    string recorder = @"\Program Files\AudioRecorder\AudioRecorder.exe";

    sendBuffer = Encoding.ASCII.GetBytes("REQ-12-");
    managerSocket.SendTo(sendBuffer);

    CreateProcess( recorder, "", pi);
    Thread speech = new Thread(new ThreadStart(SpeechRecognition));
    speech.Start();
    MessageBox.Show("Speech Recognition Enabled");
  }
  else
  {
    this.speechButton.Text = "Enable CnC";

    ProcessEntry[] pe = ProcessEntry.GetProcesses();

    for(int i=0; i<pe.Length; i++)
    {
      if(pe[i].ToString() == "AudioRecorder.exe")
      {
        pe[i].Kill();
        MessageBox.Show("Speech Recognition Disabled");
        break;
      }
    }

    sendBuffer = Encoding.ASCII.GetBytes("REQ-13-");
    managerSocket.SendTo(sendBuffer);
  }
}

/// <summary>
/// Process the incoming answer.
/// </summary>
/// <returns></returns>
public static bool processAnswer()
{
  bool exit = false;
  string text, message;
  int index, code;

  // Receive packet and extract just the bytes containing useful data
  managerSocket.ReceiveFrom(receiveBuffer);
  message = Encoding.ASCII.GetString(receiveBuffer, 0, receiveBuffer.Length);
```

```
  //Process the received string and extract the code
  index = message.IndexOf("-", 0)+1;

  code = Convert.ToInt32( message.Substring(index, 2) );

  index = message.IndexOf("-", index)+1;

  text = message.Substring(index, 20);


  switch(code)
  {
    case 0: // Request for closing the communication
      exit = true;
      break;

    case 6: // The file doesn't exist
      MessageBox.Show("The file doesn't exist, please select another one");
      exit = true;
      break;

    case 7: // The file is not valid
      MessageBox.Show("The file is not a valid one,
        please select another with xml extension");
      exit = true;
      break;

    case 8:
      processFileInfo(text);
      exit = true;
      break;
  }
  return exit;
}

/// <summary>
/// Check if there is enough space for the incoming playlist.
/// </summary>
/// <param name="megas"></param>
/// <returns></returns>
public static bool storageAwareness(double megas)
{
  string[] scardsList;
  int numScards;
  double totalFree = 0;
  bool result = true;

  scardsList = StorageCard.GetStorageCardNames();
  numScards = scardsList.Length;

  StorageCard.DiskFreeSpace dfs = new StorageCard.DiskFreeSpace();

  for(int i=0;i<numScards;i++)
  {
    dfs = StorageCard.GetDiskFreeSpace(scardsList[i]);
    totalFree = totalFree + dfs.TotalFreeBytes;
  }

  totalFree = totalFree / (1024*1024);
  totalFree = Math.Round(totalFree,2);

  if (totalFree < megas)
  {
    result = false;
  }
```

```
    return result;
  }

  public static bool NetworkMonitor()
  {
    bool state;
    OpenNETCF.Net.IP_ADAPTER_INFO info = new IP_ADAPTER_INFO();
    Adapter a = info.FirstAdapter();

    while(!a.IsWireless)
      a = info.NextAdapter();

    string ip = a.CurrentIpAddress;

    if (ip == "0.0.0.0")
      state = false;
    else
    {
      string st;
      SignalStrength strength = a.SignalStrength;
      st = strength.ToString();

      switch (st)
      {
        case "Excellent":
          state = true;
          break;

        case "Very Good":
          state = true;
          break;

        case "Good":
          state = true;
          break;

        default:
          state = false;
          break;
      }
    }
    return state;
  }

  /// <summary>
  /// Function to execute by the thread which is
  /// answering the request from FileSender
  /// while a transmission of a playlist is
  /// performed.
  /// </summary>
  public static void NetworkState()
  {
    while(!exitFlag_NetworkState)
    {
      exitFlag_NetworkState = processRequestFromLaptop();
    }
  }

  /// <summary>
  /// Answers to the incoming requests from FileSender
  /// about the Network State during a transmission of
  /// a playlist.
  /// </summary>
  /// <returns></returns>
  public static bool processRequestFromLaptop()
  {
    bool exit = false;
```

```
    string message, text;
    int index, code;

    // Receive packet and extract just the bytes containing useful data
    managerSocket.ReceiveFrom(receiveBuffer);
    message = Encoding.ASCII.GetString(receiveBuffer, 0, receiveBuffer.Length);

    //Process the received string and extract the code
    index = message.IndexOf("-", 0)+1;
    code = Convert.ToInt32( message.Substring(index, 2) );

    switch(code)
    {
      case 0: // Close connection
        inProcess = 0;
        exit = true;
        downloading = false;
        break;

      case 11: // Finish sending
        MessageBox.Show("The new audio content is ready to be used");
        inProcess = 0;
        downloading = false;
        exit = true;
        text = "REQ-00-";
        sendBuffer = Encoding.ASCII.GetBytes(text);
        managerSocket.SendTo(sendBuffer);
        break;

      case 3:
        bool networkState = NetworkMonitor();
        if (networkState)
          text = "ACK-09-";

        else
          text = "ACK-10-";

        sendBuffer = Encoding.ASCII.GetBytes(text);
        managerSocket.SendTo(sendBuffer);
        exit = false;
        break;
    }
    return exit;

  }

  /// <summary>
  /// Process the information of the file and decide if
  /// start with the transmission according to the
  /// context information
  /// </summary>
  /// <param name="info"></param>
  public static void processFileInfo(string info)
  {
    int index, index2;
    string sMb, sMin;
    double Mb, Min;
    bool storage, network;

    index = info.IndexOf("-", 0);

    sMb = info.Substring(0, index);

    index += 1;
    index2 = info.IndexOf("-", index);

    sMin = info.Substring(index, index2-index);
```

```
      // Amount of Megabytes contained at the playlist
      sMb = sMb.Replace(",",".");
      Mb = Convert.ToDouble(sMb);

      // Amount of Minutes contained at the playlist
      sMin = sMin.Replace(",",".");
      Min = Convert.ToDouble(sMin);

      // Storage Awareness
      storage = storageAwareness(Mb);

      if (!storage)
      {
        MessageBox.Show("There is not enough space for the selected playlist");
        return;
      }
      else
      {
        // Battery Awareness
        double mainBatteryLife, backUpBatteryLife, totalBatteryLife;

        // Calculate the approximate time that sending the files in the playlist
        // taking into account that in experiments the averge speed was of 390.87Kbps
        sendTime = Math.Round((((Mb * 1024) / 390.87) / 60),2);

        // Obtain the remaining life time in minutes of the main battery
        mainBatteryLife = Battery.getMainBatteryLifeMinutes(status);

        // Obtain the remaining life time in minutes of the backup battery
        backUpBatteryLife = Battery.getBackUpBatteryLifeMinutes(status2);

        totalBatteryLife = mainBatteryLife + backUpBatteryLife;

        if (sendTime > totalBatteryLife)
        {
          MessageBox.Show("There is not enough battery to receive the selected
playlist");
          return;
        }
        else
        {
          // Network Awareness

          string text;
          network = NetworkMonitor();

          if (network)
          {
            text = "REQ-02-" + fileName + "-" + fileName.Length + "-";
            sendBuffer = Encoding.ASCII.GetBytes(text);
            managerSocket.SendTo(sendBuffer);

            exitFlag_NetworkState = false;
            Thread threadNetworkState = new Thread(new ThreadStart(NetworkState));
            threadNetworkState.Start();
          }
          else
          {
            string message = "The current link quality and/or
              Rssi are not favorable, do you want to proceed anyway?";
            string caption = "Network State";
            MessageBoxButtons buttons = MessageBoxButtons.YesNo;
            DialogResult res;

            res = MessageBox.Show(message, caption, buttons,
```

```
            MessageBoxIcon.Question, MessageBoxDefaultButton.Button1);

          if (res == DialogResult.Yes)
          {
            text = "REQ-02-" + fileName + "-" + fileName.Length + "-";
            sendBuffer = Encoding.ASCII.GetBytes(text);
            managerSocket.SendTo(sendBuffer);
            MessageBox.Show("The new content will be ready in approximately "
              + sendTime + " minutes");

            exitFlag_NetworkState = false;
            Thread threadNetworkState = new Thread(new ThreadStart(NetworkState));
            threadNetworkState.Start();
          }
          else
          {
            return;
          }
        }
      }
    }
  }

  /// <summary>
  /// Function to be executed by a thread,
  /// from the moment that a request for info about
  /// a file is sent to the Manager, the thread will be
  /// listening until the connection is closed.
  /// </summary>
  public static void background()
  {
    while(!exitFlag_ProcessAnswer)
    {
      exitFlag_ProcessAnswer = processAnswer();
    }
  }

  /// <summary>
  /// Send a request to the Manager containing the name of the file.
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void requestButton_Click(object sender, System.EventArgs e)
  {
    int index;
    string extension, aux, text;

    downloading = true;
    inProcess ++;

    if (inProcess > 1)
    {
      MessageBox.Show("Extra content is being downloaded currently,
        please wait until it is finish to request for more");
      return;
    }
    else
    {
      MessageBox.Show("Please, enter the name of the file containing the desired
playlist, include the extension (.xml)");

      saveFileDialog.ShowDialog();
      fileName = saveFileDialog.FileName;

      index = fileName.LastIndexOf(@"\");

      if (index == -1)
```

```
      {
        MessageBox.Show("Not a valid file");
      }
      else
      {
        index = fileName.LastIndexOf(".");

        if (index == -1)
        {
          MessageBox.Show("Please, enter a valid file with extension .xml");
        }
        else
        {
          index = fileName.LastIndexOf(".")+1;
          extension = fileName.Substring(index, fileName.Length - index);

          if (extension != "xml")
          {
            MessageBox.Show("Please, enter a valid file with extension .xml");
          }
          else
          {
            aux = fileName.ToLower();
            index = aux.LastIndexOf(@"\")+1;
            fileName = aux.Substring(index, aux.Length - index);
            text = "REQ-01-" + fileName + "-" + fileName.Length + "-";
            sendBuffer = Encoding.ASCII.GetBytes(text);
            managerSocket.SendTo(sendBuffer);
            exitFlag_ProcessAnswer = false;

            Thread threadBackground = new Thread(new ThreadStart(background));
            threadBackground.Start();
          }
        }
      }
    }
  }

  /// <summary>
  /// Place the selected item in the playlist one position up
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void menuItem1_Click(object sender, System.EventArgs e)
  {
    int prev, selected, aux;
    string selectedRow, prevRow;

    aux = listBox.Items.Count;
    if (aux == 0)
      MessageBox.Show("Please, add elements to the list before reordering");
    else
    {
      selected = listBox.SelectedIndex;

      if (selected == 0)
      {
        MessageBox.Show("This is already the first element");
        return;
      }
      else
      {
        prev = selected - 1;
        selectedRow = listBox.SelectedItem.ToString();
        listBox.SelectedIndex = prev;
        prevRow = listBox.SelectedItem.ToString();
        listBox.Items.RemoveAt(selected);
```

```
        listBox.Items.RemoveAt(prev);

        aux = selected;
        selected = prev;
        prev = aux;

        listBox.Items.Insert(selected, selectedRow);
        listBox.Items.Insert(prev, prevRow);
        listBox.SelectedIndex = 0;
        listBox.Update();
      }
    }
  }

  /// <summary>
  /// Place the selected item in the playlist one position down
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void menuItem2_Click(object sender, System.EventArgs e)
  {
    int next, selected, total;
    string selectedRow, nextRow;

    total = listBox.Items.Count;

    if (total == 0)
      MessageBox.Show("Please, add elements to the list before reordering");
    else
    {
      selected = listBox.SelectedIndex;
      if (selected == total-1)
      {
        MessageBox.Show("This is already the last element");
        return;
      }
      else
      {
        next = selected + 1;
        selectedRow = listBox.SelectedItem.ToString();
        listBox.SelectedIndex = next;
        nextRow = listBox.SelectedItem.ToString();

        listBox.Items.RemoveAt(next);
        listBox.Items.RemoveAt(selected);

        listBox.Items.Insert(next - 1, nextRow);
        listBox.Items.Insert(next, selectedRow);

        listBox.SelectedIndex = 0;
        listBox.Update();
      }
    }
  }

  /// <summary>
  /// Delete the selected row from the ListBox.
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void menuItem4_Click(object sender, System.EventArgs e)
  {
    int selected, index, index2, length, total;
    string selectedRow, aux;

    total = listBox.Items.Count;
    if (total == 0)
```

```
      MessageBox.Show("Please, add elements to the list before deleting");
    else
    {
      selected = listBox.SelectedIndex;
      selectedRow = listBox.SelectedItem.ToString();

      index = selectedRow.IndexOf("  ",0);
      index = selectedRow.IndexOf("  ",index+2);
      index = selectedRow.IndexOf("  ",index+2);
      index = selectedRow.IndexOf("  ",index+2)+2;
      index2 = selectedRow.IndexOf("  ",index);
      length = index2 - index;
      aux = selectedRow.Substring(index,length);

      listBox.Items.RemoveAt(selected);
      listBox.Update();

      if (listBox.Items.Count == 0)
      {
        prevButton.Enabled = false;
        playButton.Enabled = false;
        nextButton.Enabled = false;
      }
      else
        listBox.SelectedIndex = 0;
    }
  }

  /// <summary>
  /// Opens the Audio Alerts form.
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void alertButton_Click(object sender, System.EventArgs e)
  {
    audioAlertForm = new AudioAlerts();
    audioAlertForm.Show();
  }

  private void timer2_Tick(object sender, System.EventArgs e)
  {
    string time, tmp;
    DateTime currTime, alertTime;
    double hour, min, durationB, durationS;
    int result;

    string programWM = "Windows Media";
    string programAP = "Command Prompt";

    IntPtr hWndWM, hWndAP;

    alertStruct aux = new alertStruct();

    timer2.Enabled = false;

    if (alerts.Count > 0)
    {
      currTime = DateTime.Now;
      hour = Convert.ToDouble(currTime.Hour);
      min = Convert.ToDouble(currTime.Minute);
      tmp = hour.ToString() + ":" + min.ToString();
      currTime = Convert.ToDateTime(tmp);


      aux = (alertStruct)alerts[0];
      time = aux.time;
      alertTime = Convert.ToDateTime(time);
```

```
      result = alertTime.CompareTo(currTime);

      if (result == 0)
      {
        // First check if the audio alert file has arrived
        FileInfo f = new FileInfo(aux.fileName);
        if (!f.Exists)
        {
          int index;
          string auxFile;
          index = aux.fileName.LastIndexOf(".");
          auxFile = aux.fileName.Substring(0,index) + ".txt";

          FileInfo f2 = new FileInfo(auxFile);

          if (!f2.Exists)
            MessageBox.Show("An audio alert was programmed for this
              moment but there was a problem while receiving the audio file");
          else
          {
            TextReader tr = new StreamReader(auxFile);
            string text = tr.ReadToEnd();
            MessageBox.Show("The following text was contained in an audio
              alert for this moment but due to network problems it was not
              possible to deliver it." + " Audio alert: " + '"' + text + '"');
          }
          // Delete the first element of alerts
          alerts.RemoveAt(0);
        }
        else
        {
          // Stop the current playout
          hWndWM = FindWindow(null,programWM);
          if(!hWndWM.Equals(IntPtr.Zero))
          {
            DestroyWindow(hWndWM);
          }

          hWndAP = FindWindow(null,programAP);
          if(!hWndAP.Equals(IntPtr.Zero))
          {
            DestroyWindow(hWndAP);
          }

          // Delete the first element of alerts
          alerts.RemoveAt(0);

          tmpcurrentTrack = currentTrack;
          currentTrack = aux.fileName;

          FileInfo fi = new FileInfo(aux.fileName);
          durationB = fi.Length;
          durationB = (durationB / 1000) * 8;
          durationS = (durationB / 352.8);
          durationS = Math.Round(durationS,2);

          durationS = durationS;

          LaunchWaveAudioPlayer();

          timer3.Interval = (int)((durationS*1000) + 2000);
          timer3.Enabled = true;
        }
      }
    }
    timer2.Enabled = true;
  }
```

```
private void timer3_Tick(object sender, System.EventArgs e)
{
  timer3.Enabled = false;
  currentTrack = tmpcurrentTrack;
  if (listBox.Items.Count > 0)
  {
    this.playButton_Click(sender, e);
  }
}

public static void playAudioAlert()
{
  LaunchWaveAudioPlayer();
}

public void SpeechRecognition()
{
  byte[] buffer = new byte[128];
  int index, index2, code;
  string command, text;

  while(!exitFlag_ProcessAnswer)
  {
    managerSocket.ReceiveFrom(buffer);

    text = Encoding.ASCII.GetString(buffer, 0, buffer.Length);

    index = text.IndexOf("-", 0)+1;

    code = Convert.ToInt32( text.Substring(index, 2) );

    index = text.IndexOf("-", index)+1;
    index2 = text.IndexOf("-", index);

    command = text.Substring(index, index2-index);

    switch(command)
    {
      case "play":
        play();
        break;

      case "previous":
        previous();
        break;

      case "next":
        next();
        break;

      case "stop":
        stop();
        break;
    }
  }
}

public MainForm()
{
  InitializeComponent();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
```

```
{
  base.Dispose( disposing );
}
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
  this.mainMenu1 = new System.Windows.Forms.MainMenu();
  this.stopButton = new System.Windows.Forms.Button();
  this.playButton = new System.Windows.Forms.Button();
  this.nextButton = new System.Windows.Forms.Button();
  this.exitButton = new System.Windows.Forms.Button();
  this.prevButton = new System.Windows.Forms.Button();
  this.loadButton = new System.Windows.Forms.Button();
  this.speechButton = new System.Windows.Forms.Button();
  this.openFileDialog = new System.Windows.Forms.OpenFileDialog();
  this.listBox = new System.Windows.Forms.ListBox();
  this.contextMenu1 = new System.Windows.Forms.ContextMenu();
  this.menuItem1 = new System.Windows.Forms.MenuItem();
  this.menuItem2 = new System.Windows.Forms.MenuItem();
  this.menuItem4 = new System.Windows.Forms.MenuItem();
  this.timerFinished = new System.Windows.Forms.Timer();
  this.requestButton = new System.Windows.Forms.Button();
  this.saveFileDialog = new System.Windows.Forms.SaveFileDialog();
  this.alertButton = new System.Windows.Forms.Button();
  this.timer2 = new System.Windows.Forms.Timer();
  this.timer3 = new System.Windows.Forms.Timer();
  //
  // stopButton
  //
  this.stopButton.Enabled = false;
  this.stopButton.Location = new System.Drawing.Point(16, 232);
  this.stopButton.Size = new System.Drawing.Size(40, 24);
  this.stopButton.Text = "Stop";
  this.stopButton.Click += new System.EventHandler(this.stopButton_Click);
  //
  // playButton
  //
  this.playButton.Enabled = false;
  this.playButton.Location = new System.Drawing.Point(96, 232);
  this.playButton.Size = new System.Drawing.Size(40, 24);
  this.playButton.Text = "Play";
  this.playButton.Click += new System.EventHandler(this.playButton_Click);
  //
  // nextButton
  //
  this.nextButton.Enabled = false;
  this.nextButton.Location = new System.Drawing.Point(136, 232);
  this.nextButton.Size = new System.Drawing.Size(40, 24);
  this.nextButton.Text = "Next";
  this.nextButton.Click += new System.EventHandler(this.nextButton_Click);
  //
  // exitButton
  //
  this.exitButton.Location = new System.Drawing.Point(176, 232);
  this.exitButton.Size = new System.Drawing.Size(40, 24);
  this.exitButton.Text = "Exit";
  this.exitButton.Click += new System.EventHandler(this.exitButton_Click);
  //
  // prevButton
  //
  this.prevButton.Enabled = false;
  this.prevButton.Location = new System.Drawing.Point(56, 232);
  this.prevButton.Size = new System.Drawing.Size(40, 24);
```

```
this.prevButton.Text = "Prev";
this.prevButton.Click += new System.EventHandler(this.prevButton_Click);
//
// loadButton
//
this.loadButton.Location = new System.Drawing.Point(16, 200);
this.loadButton.Size = new System.Drawing.Size(96, 24);
this.loadButton.Text = "Load PlayList";
this.loadButton.Click += new System.EventHandler(this.loadButton_Click);
//
// listBox
//
this.listBox.ContextMenu = this.contextMenu1;
this.listBox.Location = new System.Drawing.Point(16, 16);
this.listBox.Size = new System.Drawing.Size(208, 145);
//
// contextMenu1
//
this.contextMenu1.MenuItems.Add(this.menuItem1);
this.contextMenu1.MenuItems.Add(this.menuItem2);
this.contextMenu1.MenuItems.Add(this.menuItem4);
//
// menuItem1
//
this.menuItem1.Enabled = false;
this.menuItem1.Text = "Up";
this.menuItem1.Click += new System.EventHandler(this.menuItem1_Click);
//
// menuItem2
//
this.menuItem2.Enabled = false;
this.menuItem2.Text = "Down";
this.menuItem2.Click += new System.EventHandler(this.menuItem2_Click);
//
// menuItem4
//
this.menuItem4.Enabled = false;
this.menuItem4.Text = "Delete";
this.menuItem4.Click += new System.EventHandler(this.menuItem4_Click);
//
// timerFinished
//
this.timerFinished.Tick += new System.EventHandler(this.timerFinished_Tick);
//
// requestButton
//
this.requestButton.Location = new System.Drawing.Point(120, 200);
this.requestButton.Size = new System.Drawing.Size(96, 24);
this.requestButton.Text = "New Content";
this.requestButton.Click += new System.EventHandler(this.requestButton_Click);
//
// alertButton
//
this.alertButton.Location = new System.Drawing.Point(16, 168);
this.alertButton.Size = new System.Drawing.Size(96, 24);
this.alertButton.Text = "Audio Alerts";
this.alertButton.Click += new System.EventHandler(this.alertButton_Click);
//
// speechButton
//
this.speechButton.Location = new System.Drawing.Point(120,168);
this.speechButton.Size = new System.Drawing.Size(96, 24);
this.speechButton.Text = "Enable CnC";
this.speechButton.Enabled = false;
this.speechButton.Click += new System.EventHandler(this.speechButton_Click);
//
// timer2
```

```
      //
      this.timer2.Enabled = true;
      this.timer2.Interval = 10000;
      this.timer2.Tick += new System.EventHandler(this.timer2_Tick);
      //
      // timer3
      //
      this.timer3.Tick += new System.EventHandler(this.timer3_Tick);
      //
      // MainForm
      //
      this.BackColor = System.Drawing.Color.CornflowerBlue;
      this.Controls.Add(this.alertButton);
      this.Controls.Add(this.requestButton);
      this.Controls.Add(this.listBox);
      this.Controls.Add(this.loadButton);
      this.Controls.Add(this.prevButton);
      this.Controls.Add(this.exitButton);
      this.Controls.Add(this.nextButton);
      this.Controls.Add(this.playButton);
      this.Controls.Add(this.stopButton);
      this.Controls.Add(this.speechButton);
      this.Menu = this.mainMenu1;
      this.Text = "MediaPlayer";
    }
    #endregion

    /// <summary>
    /// The main entry point for the application.
    /// </summary>

    static void Main()
    {
      IPHostEntry hostInfo = Dns.Resolve("unnamed");
      IPAddress ipaddress = hostInfo.AddressList[0];

      managerSocket = new UDPSocket(ipaddress, LOCAL_PORT, REMOTE_PORT);

      Application.Run(new MainForm());
    }

  }
}
```

IMIT/LCN 2005-17

www.kth.se