Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Interface de Fala para Dispositivos Móveis

por

João Freitas

Submetido ao Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores como requisito parcial para obtenção do grau de licenciado em Engenharia Informática e de Computadores

ISEL, 15 de Setembro de 2007

Interface de Fala para Dispositivos Móveis

por

João Freitas

Submetido ao Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores como requisito parcial para obtenção do grau de licenciado em Engenharia Informática e de Computadores

ISEL, 15 de Setembro de 2007

Autor _____

Aluno n.º 26068, DEETC

Certificado por _____

Maria João Barros, orientador(a) do projecto

Aceite por _____

António Luís Freixo Guedes Osório, responsável de curso, ___/___/_____

Interface de Fala para Dispositivos Móveis

por
João Freitas

Submetido ao Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores como requisito parcial para obtenção do grau de licenciado em Engenharia Informática e de Computadores

ISEL, 15 de Setembro de 2007

Resumo

As tecnologias da fala permitem aos utilizadores interagir com todo o tipo de máquinas através da fala. A interacção homem-máquina varia de acordo com o tipo de sistema automático e os dispositivos móveis (Pocket Pc e SmartPhone) não são excepção. O sucesso de uma aplicação de fala em dispositivos móveis está dependente de uma série de aspectos relacionados com a natureza da tarefa, por exemplo, cenários de utilização, tipo de utilizador, desempenho da aplicação, memória necessária, ruído ambiente e posição do dispositivo em relação ao utilizador. Para ter sucesso, as interfaces de fala devem ser superiores a interfaces alternativas que executam a mesma tarefa. Ao longo deste trabalho são apresentados problemas, desafios, metodologias e optimizações no desenvolvimento de interfaces de fala para aplicações móveis.

Neste trabalho é apresentada uma versão do produto Voice Command adaptada ao Português Europeu. O Voice Command é uma aplicação para dispositivos Pocket PC e SmartPhone que permite ao utilizador comandar e controlar o seu dispositivo através da voz. Para realizar com sucesso esta adaptação foi previamente efectuado um estudo de interfaces fala para dispositivos móveis e desenvolvida uma versão compacta do motor de reconhecimento em Português Europeu. O motor de reconhecimento é depois integrado no Voice Command juntamente com um sintetizador, já existente e também em Português Europeu. No final é apresentada uma versão beta do produto completamente adaptada ao Português Europeu e o resultado de um estudo de usabilidade. Este estudo compara o uso da interface gráfica com a interface de fala ao executar tarefas básicas num dispositivo móvel e permite tirar conclusões sobre como melhorar a experiência do utilizador ao usar uma interface de fala.

Palavras-chave: interface de fala; dispositivos móveis; Voice Command; Português Europeu.

Orientador(a) do projecto: Maria João Barros, DEETC-ISEL, Instituto Superior de Engenharia de Lisboa

Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Spoken Language Interface for Mobile Devices

by

João Freitas

Submited to the Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores as a partial requisite to obtain the degree of Graduated in Computer Science Engineering

ISEL, 15 September 2007

Spoken Language Interface for Mobile Devices

por
João Freitas

Submited to the Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores as a partial requisite to obtain the degree of Graduated in Computer Science Engineering

ISEL, 15 de Setembro de 2007

Abstract

Spoken language technologies allow the users to interact with all kind of machines through speech. Human-computer interaction varies according with the type of automated system and mobile devices (Pocket Pc and SmartPhone) aren't exceptions. The success of a speech application in mobile devices is dependent on a series of aspects related with the nature of the task such as usage scenarios, type of user application performance, required memory, environment, associated background noise variation and device positioning towards the user. To be successful, speech interfaces should be superior to alternative interfaces that perform the same task. On the course of this work, problems, challenges, methodologies and optimizations concerning the development of spoken language interfaces are presented.

In this work it is presented a localized version of Voice Command in European Portuguese. Voice Command is a product designed for Pocket PC and Smartphone devices that allows the user to command and control the device using his voice. To accomplish this objective a previous study of spoken language interfaces is presented focusing on mobile applications. A compacted version of a European Portuguese Speech Recognizer engine was also developed, which is then integrated into Voice Command along with a European Portuguese Text-to-speech engine already developed. At the end it is presented a beta version of the product fully localized to European Portuguese, as well as the results of a usability evaluation. This evaluation allows comparing a graphical interface with a spoken language interface when accomplishing the same tasks in a mobile device. They also allow to conclude how to improve user experience when using a spoken language interface.

Keywords: Spoken language interface; mobile devices; Voice Command; European Portuguese.

Project's director: Maria João Barros, DEETC-ISEL, Instituto Superior de Engenharia de Lisboa

To my father

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Eng. Maria João Barros for her excellent orientation, help and availability throughout this work.

I would like to thank Professor Miguel Sales Dias for the opportunity to accomplish this project and his excellent supervision and orientation.

I would like to thank Eng. António Calado for his excellent co-orientation, advices and guidance in this project.

I would like to thank to the rest of the MLDC members, for their support, advices and help during this project.

I would like to thank to my friends and family, in particular to my mother and Diana, for their invaluable care, motivation and support.

GLOSSARY OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
ASR	Automatic Speech Recognition
BVT	Basic Verification Tests
CAB	Cabinet (file format)
CE	Compact Edition
CFG	Context-Free Grammar
COM	Component Object Model
EP	European Portuguese
G2P	Grapheme to Phoneme
GUI	Graphical User Interface
HCI	Human-Computer Interface
HMM	Hidden Markov Model
HTC	High Tech Computer Corp
HTK	Hidden Markov Model Toolkit
HYP	Internal Microsoft Format
IDE	Integrated Development Environment
IS	Install Shield
ISEL	Instituto Superior de Engenharia de Lisboa
IT	Information Technologies
JNI	Java Native Interface
KB	Kilobyte
MB	Megabyte
MFCC	Mel Frequency Cepstral Coefficients
MLDC	Microsoft Language Development Center
MS	Microsoft
OEM	Original Equipment Manufacture
PPC	Pocket PC

PTG	Portuguese
PTT	Push-to-talk
RCW	Runtime Callable Wrapper
SAPI	Speech API
SCG	Speech Components Group, placed at Redmond
SLU	Spoken Language Understanding
SR	Speech Recognition
STFT	Short-Time Fourier Transform
TAM	Text Analysis Module
TTS	Text-to-speech
UX	User Experience
VC	Voice Command
VUI	Voice User Interface
WM	Windows Mobile
XML	Extensible Markup Language

Chapter 1

Introduction

The present document describes some of the methodologies and processes that are involved in the creation of spoken language interfaces for mobile devices. This includes the development of acoustic models based on Hidden Markov Models (HMMs), spoken language systems, spoken language interfaces as a Human-Computer Interface (HCI), component integration such as, speech recognition (SR) and text-to-speech (TTS) modules and Voice Command (VC) adaptation to a specific language (European Portuguese), all considering a mobility environment.

Spoken language technology has suffered a significant evolution in the last years, being present nowadays in the daily life of many people. The tendency is that systems based on speech continue to extend its presence amongst humans [Huang01].

"The ultimate impact of spoken language technologies depends on whether you can fully integrate the enabling technologies with applications so that users find it easy to communicate with computers."

Huang X., Acero A., Hon H. in
Spoken Language Processing, 2001 [Huang01]

The adoption of spoken language technology as an HCI has the basic goal to improve the interaction between users and computers by making computers more usable and receptive to the user's needs. Nonetheless, the impact and receptiveness of a speech application on the users is dependent on the following aspects:

- Nature of the task – Which is the main goal to accomplish.
- Usage scenario – In which scenario is the task executed.
- Interface design – User interface engineering.
- Language – In which language is the system developed.

- Environment – In which environmental conditions is the task executed.
- Target platform - In which platform it is supposed the software to run.
- User group – Which user profile is the application directed to.

Spoken language applications are able to recognize and/or synthesize speech. Automatic speech recognition (ASR) can be described as the process of converting a speech signal, pronounced by a human user, to a sequence of words, through computer algorithms [Rabiner93]. Text-to-speech synthesis can be viewed as the process of converting normal language text into speech [Huang01], [Black07]. This work focus on the speech recognition problem, but it also describes the integration of a TTS engine on mobile devices.

1.1 Motivations

In this section it is described the following:

- The motivations/challenges behind the adoption of a spoken language as an interface modality or as a control modality.
- The integration process of spoken language technologies on mobile applications as an HCI modality;
- The relevance and importance of a language in a spoken language interface, in this case European Portuguese (EP).

1.1.1 Spoken Language Interface

Since the beginning of times, speech communication has been and will be the dominant mode of human social bonding and information exchange [Huang01].

A spoken language interface allows interacting, through speech, with a computer. Speech can be adopted as a primary interface modality or as a command and control modality in parallel with others, such as mouse, keyboard, touch-screen or joystick. The combination of speech with other interaction modalities is generally more effective than a unimodal interface, due the inexistence of a 100% accurate method to perform speech recognition [Huang01] [Acero06]. Spoken language interfaces impose the challenge that neither speech recognition nor understanding is perfect, therefore application developers

should fully understand the strengths and weaknesses of the underlying speech technologies and identify the appropriate situations where speech technology can be effectively used.

The evolution path leads to a world where we are surrounded by automatic systems [Huang01], and in many situations there is no space left for the incorporation of a keyboard or a graphical interface. For example, with wearable computers it may be impossible to incorporate a large keyboard. Another classic example of "busy hands, busy eyes" scenarios, where a spoken language interface benefits the user, is while driving a vehicle. When driving, safety is compromised by any visual distraction and hands are required for controlling the vehicle.

Spoken language interfaces also offer obvious benefits for individuals challenged with a variety of physical disabilities, such as blindness, physical limitations/handicaps and motor skills.

1.1.2 Mobility

The work presented here targets the mobile environment. Mobile devices are currently widely spread amongst the world population, existing 2,168,433,600 devices worldwide and 11,448,000 devices in Portugal, according to [CIAWeb]. This statistics demonstrate a clear interest in the use of mobile devices by the Portuguese population, having an average of more than one device per habitant (considering that Portugal has a population of 10,536,000 (2004) habitants [GovWeb]).

The common user of a mobile phone wants it to be functional and naturally easy to use. However, the trend leads to smaller devices with an increasing amount of features and services culminating in an interface overload. Although users are attracted by these new features and services, they also want a functional, easy to use device where it isn't needed to read an extensive manual to perform a simple phone call.

The process of integrating spoken language technologies in mobile applications as a HCI modality is highly dependent on the nature of the service provided by such technologies, for example, deleting an email is completely different of checking the battery of a device. In any case, such interface should make the interaction between the user and the device easier. A well-designed HCI requires the consideration of the

particular user group of the application, making sure that the interface matches the way users expect it to behave [Freitas07].

Mobility scenarios add challenges to the development of speech applications, due to hardware limitations, such as memory and performance, and conditions imposed by the different usage scenarios, e.g. environmental noise, device positioning towards the user and lack of privacy. Since these devices are also often used in a noisy environment it is necessary a robust speech recognition system. Chapter 3 illustrates precisely the process of designing and developing an application with a spoken language interface for a mobile device.

Voice Command is an example of an application that provides a spoken language interface to accomplish tasks in mobile devices (described in section 2.4).

1.1.3 European Portuguese in speech applications

Speech applications face another challenge: users want to interact with the application in their native language and use their own pronunciation. The localization[1] of speech or recognition and synthesis modules for a new language includes a complex set of procedures, which vary if we are considering recognition or synthesis. Some are of linguistic nature (pronunciation lexicons, phone sets, annotated and orthographically transcribed speech corpus, etc.) and some are of mathematical/stochastic/algorithmic nature (acoustic models, language models, prosody models, grapheme-phoneme mappings, etc.).

The Portuguese users can also use speech systems in other languages than their native one, when they know a second language, but the recognition rate and usability of the speech system will commonly decrease. The use of non-native accents has a negative effect in automatic speech recognition [Teixeira97]. The acoustic models, which are the main part of the recognition engine, are trained with speech data from native speakers and will present better recognition results with a native accent input. The experiments performed in section also reveal that users tend to cease using a non-native speech system because the commands don't come out on a natural way, preferring a slower graphical interface. This way, it becomes important that users have the possibility to use speech

[1] The term localization, used in the engineering community, refers to the adaptation of a system to a certain language variant. The adaptation is not only linguistically, but also cultural and technological.

systems in EP, especially those with special needs that depend on this type of applications to use a mobile device.

This project enlarges the existent number of speech applications in EP, for mobile devices, providing at the same time the language expansion in the technological world.

1.2 Objectives

The core objective of this project is the study of spoken language interfaces for mobile devices, culminating with the localization of Voice Command to EP. To accomplish the objectives the work has been divided in three main phases:

1. Study and analysis of spoken language technology;
2. Mobile application development;
3. Localization of Voice Command to EP.

The first two phases can be seen as a necessary preparation for the third phase of the project. For the localization of Voice Command it is necessary to have a basic knowledge of spoken language systems, as well as understand how the different components interact between them. It is also important to understand the theory behind these components and what can be done in order to improve them.

1.2.1 Study and analysis of spoken language technology

The study and analysis of spoken language technology for mobile devices consist in understanding the basics of spoken language processing - speech processing, speech recognition, HMMs, acoustic modelling, environmental robustness and speech synthesis – and also analysing typical spoken language systems (architecture and speech interface design).

1.2.2 Mobile application development

In a second phase the development of mobile applications is explored, as well as the changes introduced by the mobility scenario in speech recognition and spoken language systems. As a case study an example command and control speech application, which allows the user to perform phone calls to a contact in EP, was developed.

1.2.3 Localization of Voice Command to EP

The final phase of the project consists in adapting Voice Command to EP. Voice Command (described in section 2.4) is an application especially designed for mobile devices, which allows the user to execute common features and interact with the device using voice commands.

In a first stage it is performed a problem analysis, which includes exploring available features, application architecture, modules, components and source code, in order to understand the requirements for the localization process and the implications concerning the integration of the SR and TTS EP modules, for mobile devices. The localization or adaptation process also includes redefinition, training and altering the existent acoustic models in order to obtain a SR engine specific for mobile applications, such as Voice Command. This involves developing a Compact Edition (CE) SR engine directed for mobile devices in EP. In a second phase, the resulting engine is integrated in Voice Command along with a EP TTS existing engine, from Nuance [NuanceCommunications]. The application interface is redefined and adapted to the EP culture, changing dialogs, features, graphical interface, etc. In a concluding phase it is performed an usability evaluation, verification tests and beta testing of the application already localized to EP, in order to remove interface problems that interfere with the application success.

1.3 Scope

This project results of the cooperation between Microsoft Language Development Center (MLDC) and Instituto Superior de Engenharia de Lisboa (ISEL). MLDC is a development department in the Microsoft Portugal premises at Tagus Park, dedicated to Speech and Natural Language development.

> *"MLDC is the first Microsoft Development Center outside of Redmond dedicated to key Speech and Natural Language developments, and is a clear demonstration of Microsoft efforts of stimulating a strong software industry in EMEA. To be successful, MLDC must have close relationships with academia, R&D laboratories, companies, government and European institutions. I will continue fostering and building these*

relationships in order to create more opportunities for language research and development here in Portugal."

Miguel Sales Dias, Director of the Microsoft Language Development Center
[MLDCWeb]

The presented work constitutes the final course project for ISEL, supervised by Prof. Maria João Barros, and an internship at MLDC, coordinated by Prof. Dr. Miguel Sales Dias, Director of the MLDC. All work was accomplished at MLDC with the orientation of Eng. António Calado.

Until this date, this work has already spawned a paper on an international conference, proving the continuing interest of the scientific community in this area. Publication of this work follows:

- Freitas, J., Barros, M. J., Calado, A., Dias, M. S., "Spoken Language Interface for Mobile Devices", in the Proceedings of 3rd Language & Technology Conference, October, 2007

1.4 Audience

This project targets several interests such as: application development for mobile devices, localization engineering, spoken language processing and spoken language interfaces development.

1.5 Document Structure

The current document is structured as following:

- Chapter 1, "Introduction" - This is the current chapter, which introduces the context in where this work was executed, what motivates it and the goals it accomplishes.
- Chapter 2, "Spoken Language Technology" – In this chapter is presented the current state of the art for the subjects approached in this project.

- Chapter 3, "Development of a Speech Interface for Mobility" – This chapter presents a SR engine localization, speech application development in a mobility context and a study about spoken language interface for mobile devices.
- Chapter 4, "Voice Command European Portuguese Localization" – This chapter contains all relevant aspects of the Voice Command localization for European Portuguese, including a usability evaluation study of the application
- Chapter 5, "Conclusions and Future Work" - Finally, the last chapter is dedicated to conclusions and final remarks. Possible future work directions and lines of research are also discussed.

Chapter 2

Spoken Language Technology

In this chapter it will be analyzed the current state of the art in spoken language technology, namely, the existent components/subsystems and their interaction with each other, in this type of system. Additionally, it will also be analyzed commercial speech applications for mobile devices.

This chapter starts by broadly dissecting the two main modules of spoken language system - ASR and TTS. Then, it is explained how these modules are integrated into speech applications, along with other required components in the development of speech applications.

2.1 Spoken language systems

According to [Huang01], a spoken language system has at least one of the following three subsystems:

- **Automatic speech recognition** – converts speech into words.
- **Text-to-speech** – conveys spoken information.
- Spoken language understanding (SLU) – aims at extracting meaning from natural language speech, by interpreting utterances in a context and carry out appropriate actions (SLU systems are not approached in this document).

In the last decades spoken language systems have been based in data-driven statistical approaches with promising results [Huang01]. These approaches are based on modelling the speech signal using well-known statistical algorithms that extract information from the data.

2.1.1 Automatic speech recognition

According to [Rabiner93] there are three main approaches to ASR:

- The acoustic-phonetic approach
- The artificial intelligence approach
- The pattern recognition approach

2.1.1.1 Acoustic-phonetic

The acoustic-phonetic approach starts by processing and performing an analysis of the speech signal to provide a proper representation of its characteristics, a step common to all approaches. This involves segmenting the speech signal into regions where the acoustic properties of the signal are representative of a phonetic unit and attaching phonetic labels to each region. The problem of this approach is to decode the phonetic units into word strings. In other words, decoding the phoneme lattice (which is the result of a segmentation and labelling process that represents a sequential set of phonemes that are likely matches to the spoken input speech – fig. 1) into a word string, such that every instant of time is included in one of the phonemes in the lattice and assuring that the result (word or set of words) is valid according with the language syntax, may become a hard task. The example from fig. 1 illustrates the difficulty in decoding phonetic units into word strings, where can be derived the phonetic string SIL-AO-L-AX-B-AW-T corresponding to the word string "all about", with the L, AX, and B having been second or third choices in the lattice.

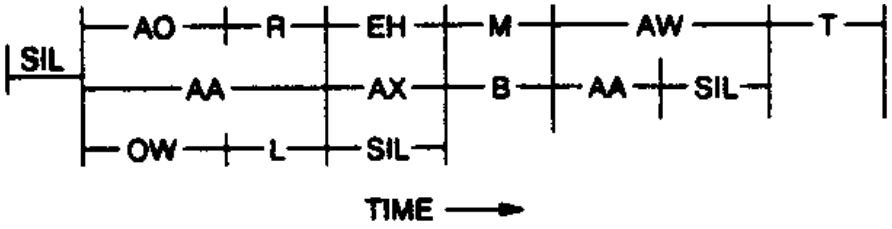

Figure 1 - Phoneme lattice [Rabiner93]

2.1.1.2 Pattern recognition

The pattern recognition approach can be divided in two steps: training of speech patterns and recognition via pattern comparison. The knowledge of speech is brought, through the training procedure, into the system. The concept is to gather enough versions of a pattern to be recognized in order to adequately characterize the acoustic properties of the

pattern. The objective is that the machine learns which acoustic properties of the speech class are reliable and repeatable across all training tokens of the pattern.

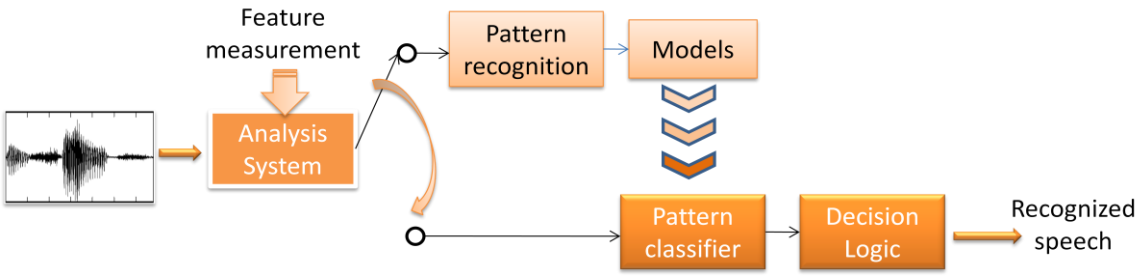

Figure 2 - Basic architecture of a SR system based on pattern recognition

Figure 2 can be divided in two distinct phases in which speech recognition systems based on pattern recognition are commonly separated:

- The training phase – In this phase speech samples (Corpus) are used to build the models used in pattern comparison.
- The testing phase – In this phase the system receives an unknown speech signal as input and determines its most probable pattern.

Based on fig. 2, this approach can be divided in the following steps:

- **Feature measurement** – a sequence of measures is applied to the entrance signal, in order to define a pattern. In speech signals the feature measurement is usually the output of some spectral analysis, such as mel-cepstrum analysis, linear predictive coding or discrete Fourier transform analysis.
- **Pattern or model training** – in this step, pattern representations of the features are created. The patterns correspond to speech sounds of the same class. The result can be a template or a model that characterizes the statistics features of the pattern.
- **Pattern classification** – in this step, speech patterns, which usually consist of a sequence of spectral vectors, are compared and the similarity between them is measured. To compare speech patterns it is used a local "spectral" distance measure and a global time alignment procedure (dynamic time warping algorithm [Sakoe78]) which compensates for different rates or time scales.
- **Decision Logic** – decide which pattern best matches the unknown pattern input.

The pattern recognition approach is the basis for the remainder of this work. This is the method of choice for three reasons [Rabiner93]:

- Simplicity of use - The approach is widely spread and used. The method behind it is easy to understand, being rich in mathematical and communication theory.
- Robustness and invariance to different speech vocabularies, users, feature sets, pattern comparison algorithms and decision rules - This property allows the technique to be used with any kind of speech units, word vocabularies and recording conditions.
- Proven high performance - This characteristic is verified in platforms such as mobile.

2.1.1.3 Artificial intelligence

The artificial intelligence approach is a combination of the two techniques previously analysed, since it exploits concepts and ideas of both. The artificial intelligence approach tries to simulate the way a person applies its intelligence in visualizing, analysing, and finally making a decision on the measured acoustic features [Rabiner93]. The system tends to adapt and learn over time the relationships between phonetic events and all known inputs learning how to distinguish similar sound classes. This approach is not detailed or used in this work.

2.1.2 Text-to-speech synthesis

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech synthesizer, and can be implemented in software or hardware. A text-to-speech system converts normal language text into speech. The TTS process can be divided in four parts which are illustrated in fig. 3.

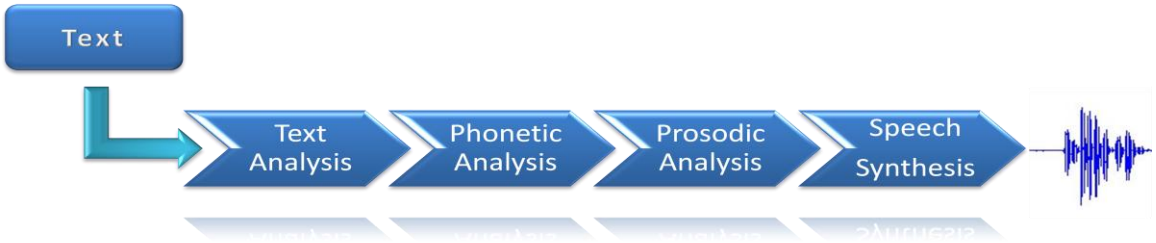

Figure 3 - Basic system architecture of a TTS system

The text analysis component normalizes the text to the appropriate form so that it may be artificially uttered. The input can be either raw text or tagged. These tags are used to assist text, phonetic, and prosodic analysis. The phonetic analysis component converts the processed text into the corresponding phonetic sequence, which is followed by prosodic analysis to attach appropriate pitch and duration information to the phonetic sequence. Finally, the speech synthesis component takes the parameters from the fully tagged phonetic sequence to generate the corresponding speech waveform. In the following sections each module is analyzed in more detail. Some authors, such as [Black07] choose to include the Phonetic and Prosodic analysis in just one stage named "Linguistic Analysis" which has the objective to find pronunciations of the words and assign prosodic structure to them, such as phrasing, intonation and durations. In this document the student choose to split this stage in order to distinguish two distinct actions of a TTS system.

2.1.2.1 Text analysis module

The text analysis module (TAM) is responsible for indicating all knowledge about the text or message that is not specifically phonetic or prosodic in its nature. Depending on the complexity of the system the analysis can be different things, from converting non-orthographic items, such as number to words, or attempting to analyze whitespaces and punctuations, to determine the document structure or perform sophisticated syntax and semantic analysis on sentences, to determine attributes that help the phonetic analysis to generate correct phonetic representation and prosodic generation to construct superior pitch contours.

2.1.2.2 Phonetic analysis module

The phonetic analysis module is responsible for the conversion from an orthographic representation to a phonemic one, including the processing of diacritic information, such as stress placement. In this stage it is performed a grapheme-to-phoneme (G2P) conversion, since phonemes are the basic units of sound. The complexity of G2P conversion is language dependent, more concretely is dependent on the relation between orthography and phonology.

In the specific case of EP the complexity is particularly high, mainly due to the Sandhi effects found in continuous speech [Barros06]. Sandhi effects are basically coarticulation phenomena and phonetic reductions of continuous speech [Braga03], [Amaral99]. When looking at continuous speech we find different phonetic transcriptions for the same word, according to the rhythm and context (if the next word is a vowel, or some consonant, if it is the end of a sentence, etc).

2.1.2.3 Prosodic analysis module

Prosody in linguistic is the study of the intonation and phonetic effects that are employed to express attitude or assumptions. Prosody determines how a sentence is spoken in terms of melody, phrasing, rhythm, accent locations and emotions. Prosody may even carry meaning (e.g. “I like the wine John” vs “I like the wine, John”) [Huang01], [Black07].

In this module the parsed text and the corresponding phoneme string are received as input. The output – prosodic generations – is dependent on the speaking style and contain the prosodic parameters such as, duration of each phoneme, fundamental frequency contour, and intensity.

2.1.2.4 Speech synthesis module

This last module is the one responsible by the waveform generation. The speech synthesis systems can be classified into four types [Barros07a]:

- Vocal Tract models based synthesis;
- Concatenative synthesis;
- HMM based synthesis;
- Unit selection based synthesis.

The vocal tract based synthesis models the movements of articulators and acoustics of vocal tract. It does not use human speech samples at runtime. Instead, the synthesized speech output is created using an acoustic model. Parameters, such as fundamental frequency, voicing and noise levels are alternated over time to create a waveform of artificial speech.

The concatenative synthesis uses using a data driven approach to generate the verbal content of the signal. It consists in the concatenation of recorded speech segments stored in a database, in order to assemble new utterances.

The HMMs based synthesis use a set of HMMs in a statistical framework to generate the speech signal. This and the concatenative approach can both be described as data-driven. In the concatenative approach we are effectively memorizing the data, whereas in the statistical approach we are attempting to learn the general properties of the data. The HMM based speech technology is recently gaining some relevance due to the good quality attained and to recent awarded systems [Blizzard]. They rely on a small database of speech models that resulted from an initial training and preparation based on specific voice features. The quality of the generated speech signal is already as good as unit selection speech systems according to [Blizzard]. The smaller database and the possibility of easily generate another voice using the same database can represent great advantages in some areas.

Unit selection based synthesis operates by selecting units from a large speech database to how well they match a specification and how well they join together. This technique can be considered as a subtype of the concatenive synthesis that uses a richer variety of speech. This has the aim of capturing a more natural variation and relying less on signal processing. The idea is that for each basic linguistic type we have a number of units, whose features vary beyond pitch and timing (e.g. stress or phrasing). During the synthesis, an algorithm (e.g. Viterbi [Viterbi67]) selects one unit from the possible choices, in an attempt to find the best overall sequence of units which matches the specification. The TTS used in this work is based on a unit selection approach.

2.2 Hidden Markov Models

In this section the HMMs are explained and it is shown how this technique can be used in speech recognition.

2.2.1 Definition of Hidden Markov Model

A natural extension to the Markov chain (see Appendix A) it is the Hidden Markov Model, which introduces a non-deterministic process that generates output observation symbols in any given state, with the observation being a probabilistic function of the state [Rabiner89]. There is no longer a one-to-one correspondence between the observation sequence and the state sequence, invalidating the possibility of unanimously determine the state sequence for a given observation sequence (i.e. the state sequence is hidden).

The HMM is viewed as a double-embedded stochastic process, where the first process can be seen as a discrete Markov chain of several states, where the states transitions represent statistical change. Associated with each state of the chain another process that describes the statistics related with the output observations of each state exists.

An HMM is then defined by:

$S = \{S_1, S_2, ..., S_N\}$- A set of states representing the state space. Here S_t is denoted as the state at time t.

$O = \{O_1, O_2, \ldots, O_M\}$ - An output observation alphabet. The observation symbols correspond to the physical output of the system being modeled.

$A = \{a_{ij}\}$ - A transition probability matrix, where a_{ij} is the probability of taking a transition from state *i* to state *j*.

$B = \{b_i (k)\}$ - An output probability matrix, where $b_i(k)$ is the probability of emitting symbol O_k when state *i* is entered.

$\pi = \{\pi_i\}$ - A initial state distribution where,

$$\pi_i = P(S_0 = i) \qquad 1 \leq i \leq N \tag{2.1}$$

Since a_{ij}, $b_i(k)$, and π_i are all probabilities, they must satisfy the following properties:

$$a_{ij} \geq 0,\ b_i(k) \geq 0,\ \pi_i \geq 0\ \forall \text{ all } i,j,k \tag{2.2}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{2.2}$$

$$\sum_{k=1}^{M} b_i(k) = 1 \tag{2.3}$$

$$\sum_{j=1}^{N} \pi_i = 1 \tag{2.4}$$

The model can be specified through the compact notation

$$\lambda = (A, B, \pi) \tag{2.5}$$

to indicate the complete parameter set of the model.

2.2.2 The three basic HMM problems

To apply HMMs in real world problems, three basic problems of interest must be addressed. These problems are the following [Rabiner93]:

1. The Evaluation Problem – Given a model λ and a sequence of observations $O = \{O_1, O_2, \ldots, O_M\}$, what is the probability $P(O|\lambda)$; i.e., the probability of the model to generate the observations?
2. The Decoding Problem – Given a model λ and a sequence of observations $O = \{O_1, O_2, \ldots, O_M\}$, what is the most likely state sequence $S = \{S_1, S_2, ..., S_N\}$in the model that produces the observations?
3. The Learning Problem – How do we adjust model parameters $\lambda = (A, B, \pi)$ to maximize $P(O|\lambda)$?

The evaluation problem – Given a model and a sequence of observations, how do we compute the probability that the observed sequence was produced by the model? To use HMM for pattern recognition we need to solve the evaluation problem, which consists on finding a way of evaluating how well a given HMM matches a given observation sequence or choosing the model that best matches the observations. The solution for this

problem can be achieved through the forward-backward procedure [Rabiner86] or Viterbi algorithm [Viterbi67].

The solution for the decoding problem is an attempt to uncover the hidden part of the model. Unlike the evaluation problem, for which an exact solution can be found, there are several ways of solving the decoding problem. The difficulty lies in defining the optimal state sequence, where criteria can be used. The most widely used criterion is to find the best single path or state sequence through the Viterbi algorithm. The Viterbi algorithm can be regarded as the dynamic programming algorithm applied to the HMM or as a modified forward algorithm. Instead of summing up probabilities from different paths coming to the same destination state, the Viterbi algorithm picks and remembers the best path.

The third problem consists in finding a method to adjust the model parameters to accurately describe the observation sequences. This is by far the most difficult of the three problems, because there is no known analytical method that maximizes the joint probability of the training data in a closed form. This problem can be solved using the iterative forward-backward algorithm.

The following example show how these problems are applied in speech recognition. Consider a simple isolated-word speech recognizer [Rabiner93], where there is an N-state HMM for each word of a W word dictionary and that the speech signal of a given word is represented as a time sequence spectral vectors. For each word in the vocabulary we have a training sequence consisting of a number of repetitions of word representations. The first task is to build individual word models and estimate model parameters for each word (problem 3). However, we need to refine the model and improve its capability of modeling word sequences. With the solution for problem 2 we can segment each of the word training sequences into states and then study the properties of the spectral vectors that lead to the observations in each state or in another words, determine the correspondent sequence of states for a given sequence of observations.

Having a set of HMMs designed and optimized, we can use the solution to the first problem to recognize an unknown word, by scoring each model based upon a given

observation sequence and select the word whose model score is highest (i.e. have the highest likelihood).

2.2.3 Types of HMMS

The types of HMMs can be divided according with the kind of output probability functions. Hidden Markov Models can be classified, depending on the type of probability function chosen for the output symbols, as discrete, continuous or semi-continuous.

A HMM is discrete when the probability density function is defined in a finite space (i.e. there is a finite number of observations). In this case the observations are characterized as discrete symbols chosen from a finite alphabet.

A HMM is continuous when the probability density function is continuous. Usually the probability density function is modeled as multivariate Gaussian mixture.

For a tied mixture or semi-continuous HMM the set of probability density functions is the same for all states and models, only changing the probability of a symbol emission in a determine state.

2.2.4 Structure and topology

A state transition in an HMM generates observation sequences. We can differentiate the kinds of models, regarding the topology, according to the characteristics between the states. An ergodic or fully connected model is defined with all the possible transition between states. This type of model has the main characteristic that every state can be reached by every other state of the model in a finite but aperiodic number of steps.

For speech recognition usually it is used a “left-to-right” or Bakis model, illustrated in fig. 4. In this model the system of states proceed from left to right. As we advance a time unit, the state index increases or stay the same – that is, it is only possible to transact from Si to Si+1 or remain in the same state. These properties allow to readily modelling signals whose characteristics change over time in a successive manner, such as speech.

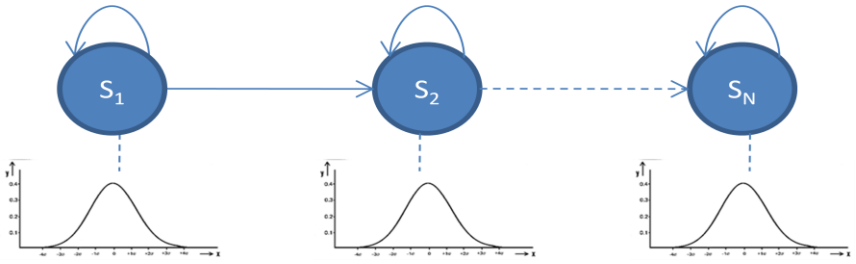

Figure 4 - Left-to-right model

The fundamental property of left-right HMMs is that the state transition coefficients have the following property:

$$a_{ij} = 0 \quad \text{i>j} \tag{2.6}$$

This indicates that no transitions are allowed to states whose indices are lower than that of the current state. Furthermore, the initial state properties have the property:

$$\pi_{\text{i}} = \begin{cases} 1 & \text{i=1} \\ 0 & \text{i} \neq 1 \end{cases} \tag{2.7}$$

because the state sequence must begin in state 1.

2.2.5 HMMs in speech recognition

With speech recognition based on HMMs, each entity of the recognizable vocabulary has a probabilistic model. The recognition is performed by determining the probability of an observed entity being generated by each one of the models [Meneses02].

To build a speech recognition system based on HMMs there should be a set of models, one for each sound class to recognize (phoneme, word, etc). After we had defined the set of sound classes that will match the number of models and chosen a model topology, we need to obtain for each class a reasonable amount of training data, usually named Corpus (section 2.3.1.1). Then, we can train the models using, for example, the forward-backward procedure [Rabiner86].

In speech recognition, we begin by extracting an observation sequence (speech signal). Next, speech parameters are extracted and the probability of the observation sequence, given each model, is computed. The observation sequence will be associated with the model or sound class that has the higher probability. Figure 5 illustrates the basic structure of a recognizer with this kind of models, having N sound classes.

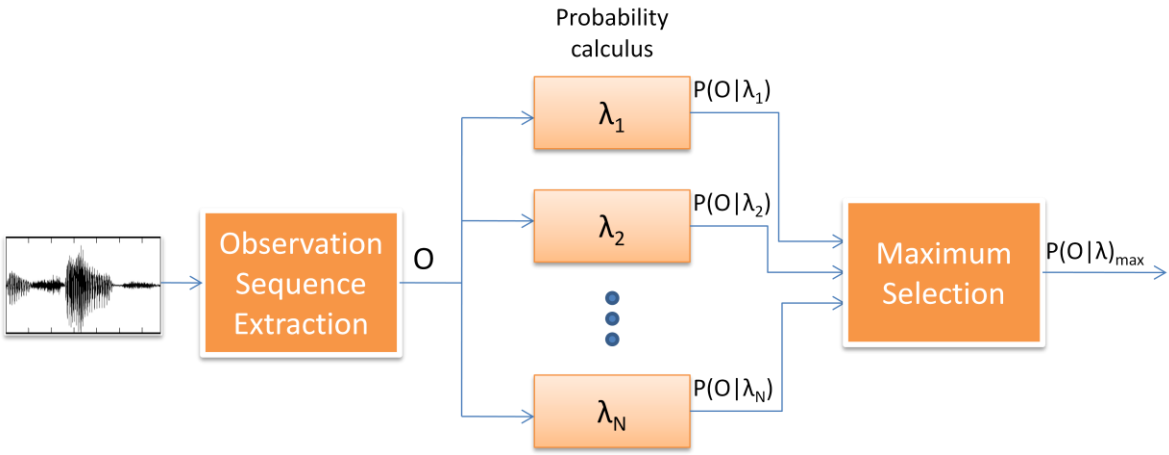

Figure 5 - Block of an isolated entity recognizer

2.3 Speech components for speech applications

Speech applications require several background components to work, such as acoustic models, SR or TTS engines, etc. In fig. 6 the components or modules, used in this project, for speech applications development, focusing on the ASR modules are presented. Figure 6 also shows how these components interact by presenting a flow between the modules that culminates on several types of speech applications.

In the following sections each one of these components and stages will be described, from the acoustic models training procedure, to the several classes of speech applications.

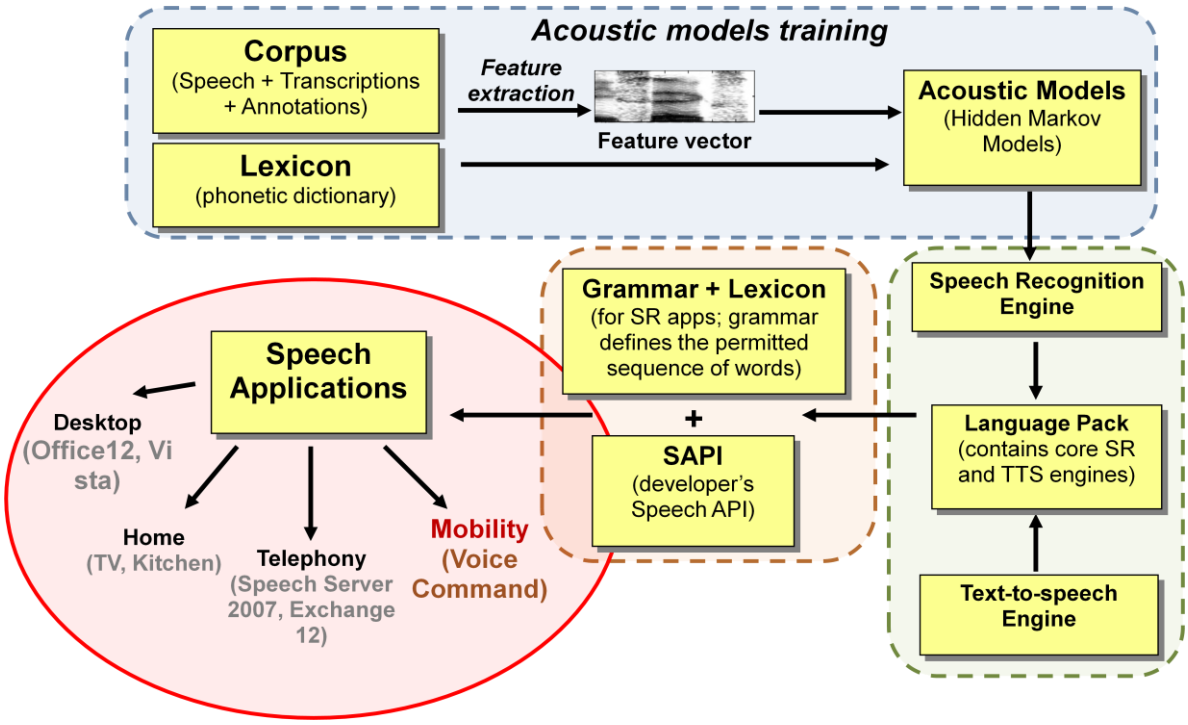

Figure 6 - Speech applications components

2.3.1 Acoustic models training

In this training phase, commonly used in pattern recognition approaches, the acoustic models are built. In order to train an acoustic model two basic inputs are required:

- Corpus;
- Phonetic Lexicon.

2.3.1.1 Corpus

The component represented as "Corpus" in fig. 6 can be defined as a set of speech data and respective transcription and annotation of that data. Taking on account that the recognition process is based on statistical/probabilistic methods, the amount of data is one of the most important characteristic of a Corpus (besides transcription and annotation quality). However, the process of acquiring large amounts of training data, is usually, a tough task because it requires the collaboration of numerous subjects with balanced (or not) characteristics (e.g. 50% male 50% female), poise word distribution, transcription and annotation tools, etc.

The Corpus composition and its characteristics should also be targeted toward the intended domain i.e. it should be appropriated for the specific domain where it will be used in order to obtain better recognition. If the target audience and utterances in the grammar (described in section 2.3.4) of the application matches the data in the Corpus it will provide improved recognition rates (e.g. if the training data does not contain "yes" or "ok", an application for command and control that requires several confirmations may present poor recognition results). Below, some characteristics that influence an application and should be taken under account when choosing a suitable corpus are presented:

- Speaking-style - isolated-word or continuous-speech.
- Speaker-variation - speaker-dependent or speaker-independent.
- Vocabulary-size - small to medium to large vocabulary.
- Recording environment – Office, Street, or others

First and foremost the training data should contain recordings of speakers speaking in a mode which will match the target application. For example, applications may recognize isolated speech or continuous speech. Many phonetic effects occur when words are spoken in context that can only be modelled if the training data contains full utterances rather than isolated words. Utterance length should therefore be considered when choosing a corpus. The speech corpus should also contain a range of accents, ages and gender which corresponds to the target users of the system. Many corpora exist which are specifically designed for training speaker independent systems where the distribution of gender, age, accent, height and/or social status are accounted for. In some cases such data is not available, and the corpus may only consist of speakers from a particular region, for example Lisbon [Braga07a]. This would be a limiting factor of the resulting system if it was required to recognize speakers from all areas of Portugal. Background noise levels and the recording channel must be also taken into account, due to signal-to-noise ratio of speech and channel distortion. For example, digital ISDN lines and analogue lines or cellular phones will distort the speech in different ways.

2.3.1.2 Lexicon

The lexicon is here referred has a phonetic dictionary that lists the phonetic transcription of the vocabulary (contains information about how the word is pronounced) in a given phonetic alphabet, such as IPA [IPAWeb] or SAMPA [SAMPAWeb]. A word can have more than one phonetic transcription in the lexicon. For words that are not in the dictionary a G2P conversion based on a set of rules is performed, trained from the existing lexicon. The G2P correspondence is relatively direct for some languages and can be highly unpredictable for others, like English and Portuguese [Bonardo05], [Barros06].

The lexicon is used for acoustic models training but also in runtime during the recognition process [Jansche01].

2.3.1.3 Acoustic models

The used acoustic models are based on statistical data-driven models of time-series – HMMs. Each HMM in a speech recognition system models the acoustic information of a specific speech segment within a language, being trained on speech recorded from the language and acoustic environment in question, for example European Portuguese telephony speech. These speech segments become representations of the speech units and can be of any size, e.g. whole sentences, whole words or sub-word phonetic units like syllables, triphones, diphones, phones, etc.

The acoustic models are the basis for automatic speech recognition and their training involve mapping models to acoustic examples obtained from training data. Training data comes in the shape of transcribed speech (i.e. waveform data along with a word level transcription - Corpus) and the respective pronunciation dictionary from which acoustic parameters are extracted. However, in order to build a robust set of statistical models, there must be a sufficient number of examples of each of the speech units within the database. For this reason, the most common choice of speech units to model is sub-word phonetic units known as “phones”. The difference between the phone and phoneme is a slim one. A phone is considered part of a phoneme’s acoustic realization, what means that there’s only a finite set of phones for any given language. These phones can be modelled efficiently in different contexts and combined to form any word in the language.

There are different kinds of phone models, context-independent and context-dependent. Context-independent models are known as monophones. Each monophone is trained on all observations of the phone in the training set independent of the context in which it was observed. Context-dependent models are built with triphones, these are models of a single phone in the context of its two surrounding phones. Triphones are able to model contextual variation caused by effects such as assimilation and coarticulation of neighboring phones.

2.3.2 Speech recognition engine

It is in the speech recognition engine that all the recognition process takes place. In this section it is described the basic function of an engine based on Microsoft speech engine – Yakima. This was also the engine used during this project. Figure 7 illustrates the overall functionality of the engine.

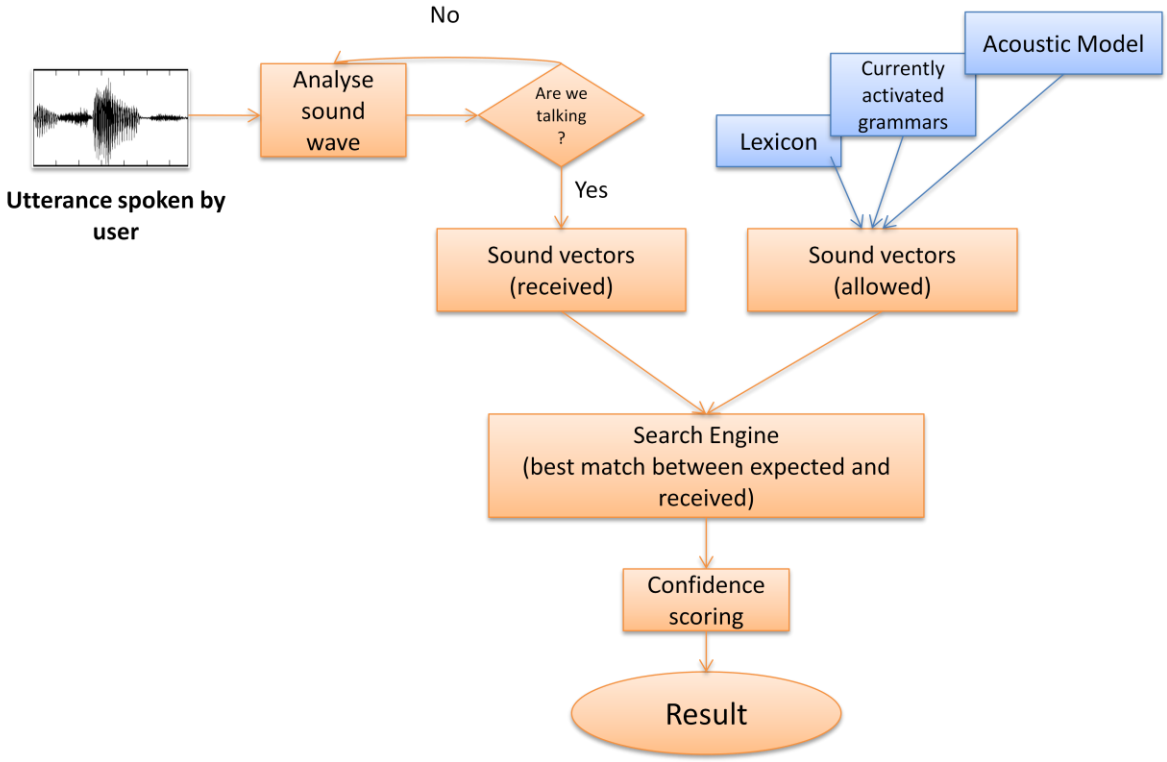

Figure 7 - Engine runtime

The application is responsible for loading the SR engine and for requesting actions/information from it. It communicates with the engine via a Speech API interface

(SAPI) [SpeechSDK]. The application will therefore request, via SAPI, that the engine is loaded. The initial status of the engine is an inactive status. It is loaded into memory together with the resources it needs, and waits for the application to tell SAPI to feed its input from the audio source – the desktop microphone, a wave file or a telephone.

SAPI abstracts the developer from the low level details of the SR engine, nonetheless, it's essential that the developer knows the potential, functionalities and work realized by the engine, in order to model and optimize the target application.

2.3.2.1 Engine recognition procedure

The speech recognition engine is split into Front-end and Decoder. The Front End converts the sound waves (utterances spoken by the user) into features vector and supplies these to the decoder coupled with a noise-probability. The Decoder recognizes the sounds as being a good match for the phonemes, which are the speech units used by the engine. Then these phonemes (which exist in the engine database) are put together to form words or phrases.

The Front-end part of the engine analyses the sound waves and outputs to the decoder, a continual stream of probabilities that the sounds are speech rather than noise. The decoder receives two parallel streams, one with the analysed input sound and another one with the probability of the input sound being speech. This ensures that the engine will only try to recognise speech when someone is speaking. However, if the probability is too low then the decoder will consider the input to be noise and will not analyse the input from the Front End.

The speech recognition process begins, in the Front-end, by extracting the features of the user spoken utterance. To extract the features of a sound wave it is calculated a time-frequency based Fourier Transform – Short-time Fourier Transform (STFT) [Huang01]. The STFT is used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. Figure 8 shows the spectrogram resulting from the transformation.

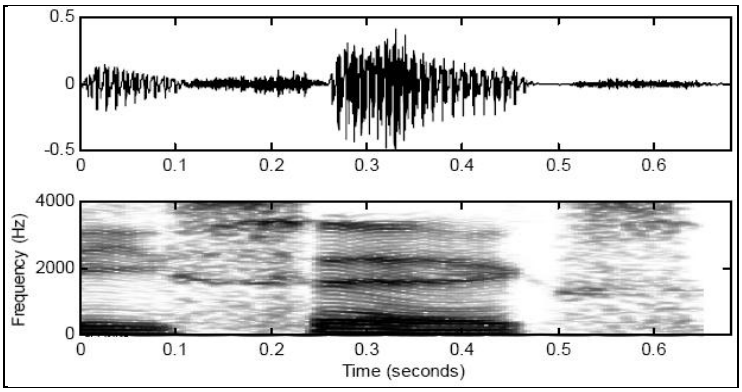

Figure 8 - Speech signal before and after an STFT

The spectrogram is then divided into frames of specific amount of time (e.g. 10 ms in Yakima's case), thus obtaining a feature vector. After computing the frequency spectrum, the volume normalization of the input sounds takes place.

In the following stage it is used the likelihood of a feature vector being generated by a particular sequence of states. Each state of an HMM consumes a frame of the feature vector. The sequence of states with the higher probability of matching the sequence of frames will correspond to the identified unit.

2.3.2.2 Senones

A senone is a sub-phonetic unit context-dependent equivalent to a HMM state of a triphone. A senone encodes information that shows of a combination of feature vectors that matches a certain sound.

2.3.2.3 Confidence scoring

To each recognition process it is given a confidence score based on the sequence of feature vectors that most likely match the sequence of states. Confidence scoring is necessary to allow dialog systems to alter the balance of False/Correct Accept/Rejects (table 1) and to allow Dialog Design Engineers (DDEs) to prevent irreversible actions being carried out due to misrecognition. The collection of prompts and grammars and the logic that links them is known as a dialog.

Correctly accept (CA)	When the recognition is correct and has a confidence score above confidence threshold.

Falsely accept (FA)	If the recognition is wrong but has a high confidence score.
Correctly reject (CR)	When the recognition has a low confidence score and is correctly rejected by the dialog system.
Falsely reject (FR)	When the recognition has a low confidence score and is falsely rejected by the dialog system.

Table 1 - Recognition results

Depending on how we want the system to behave, a threshold for the confidence scoring must be established. When minimizing the number of FA's, then the number of FRs will rise. In this case the system is rejecting any recognition of which the engine is unsure. Thus there is a trade-off between doing the wrong thing and risking user-irritation at asking him/her to repeat what s/he just said. In critical situations such as the prompt asking if the user wants to delete an item, the worse possible outcome is that the system falsely claims to have recognized an utterance, when in fact it recognized the utterance wrongly and this causes an irreversible action to be invoked.

2.3.3 Speech API

A Speech API is an intermediate layer that provides a communication channel between the application and the engines. Although in this project it was used SAPI from Microsoft, there are other solutions in the market such as VSAPI from VoiceSignal [VoiceSignal07].

Microsoft's Speech API was developed by Microsoft Corporation and its function is to facilitate the developer's task, by adding a level of indirection in the access to speech recognition and text-to-speech engines, as illustrated in fig. 9. The developer does not need to worry about low level details needed to control and manage real-time operations of the speech engines. SAPI provides a set of Component Object Model (COM) interfaces to control TTS and SR systems. The communication with the applications is done through events, using standard mechanisms, e.g. Windows Message, callback or Win32 event. SAPI applications are able to perform synchronization with real-time actions as they occur, e.g. phonemes [SAPISDK]. SAPI reduces the code overhead required for an application to use speech recognition and text-to-speech, making speech technology more accessible and robust for a wide range of applications.

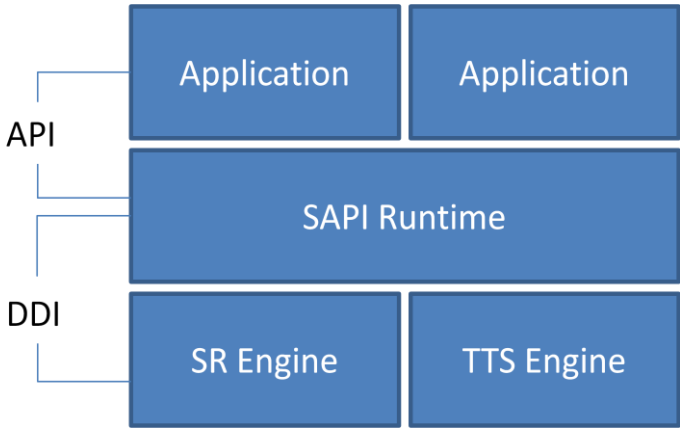

Figure 9 - SAPI overview

2.3.4 Grammars

Grammars contain the list of everything the user can say and consists in the vocabulary that can be recognized. A grammar is the engine's model of allowed utterances. A grammar is conceptually a tree of different phrases. The top of the tree corresponds to the entry point of the grammar and each branch going down represents a phrase that splits. The grammar may be of any size, have optional words (e.g. "please select Calendar" is the same as "Select Calendar") and have semantic properties.

Speech applications are often built to command and control. Command and control features are implemented through context-free grammars (CFGs). This kind of grammar defines a set of imbricated production rules. These rules are able to generate a set of words and combinations of these words that can be used to build all type of allowed sentences. The result of a grammar production can be seen as a list of valid words/sentences, which is passed on to the SR engine. A speech application uses a grammar to improve recognition accuracy by restricting and indicating to an engine which words/sentences should be expected. The valid sentences need to be carefully chosen, considering users profile and the application nature. However, other approaches to SR without using CFG do exist [Acero93][Stern96].

The CFG format in SAPI 5 defines the structure of grammars and grammar rules using Extensible Markup Language (XML) [W3CXMLWeb]. The Grammar compiler transforms the XML tags defining the grammar elements into a binary format used by SAPI 5-compliant SR engines. This compiling process can be performed either before or during application run time.

A speech application uses a grammar to accomplish the following:

- Improve recognition accuracy by restricting and indicating to an engine what words it should expect.
- Improve maintainability of textual grammars, by providing constructs for reusable text components (internal and external rule references), phrase lists, and string and numeric identifiers.
- Improve translation of recognized speech into application actions. This is made easier by providing "semantic tags," (property name, and value associations) to words/phrases declared inside the grammar.

SAPI also enables applications to create CFG structures programmatically using the *ISpGrammarBuilder* COM interface, which is inherited by *ISpRecoGrammar*. The application can use the *ISpGrammarBuilder* API to dynamically update an already loaded SAPI XML grammar, create an in-memory SAPI grammar, and/or save an in-memory SAPI grammar to a memory stream (e.g. saving grammars to the hard disk).

Applications that do not need to modify a grammar at run time, or applications that want to increase performance of their CFG-based application should load the compiled binary form statically.

2.3.4.1 Static and Dynamic Grammar Rules

A Grammar Rule is the fundamental unit in a grammar. There are top-level rules, and all rules may contain sub-rules.

The figure below shows a pseudo grammar that allows the user to know the weather in a determined day and place. The “?” symbol denotes an optional utterance and “TL” denotes top level rules.

<Rule Weather TL>	[Please]? [Tell me/What is](?) <Weather_Day> [in <Place>]? [Please] (?)
<Rule Weather_Day>	[Today’s weather] [Tomorrow’s weather] <Day_Of_Week> [’s weather] [what the weather will be like] [on]? [tomorrow/<Day_Of_Week>/at the weekend]
<Rule Day_Of_Week>	[Monday / Tuesday / Wednesday / Thursday / Friday / Saturday / Sunday]
<Rule Place>	[Lisbon / Porto]

Figure 10 - Pseudo grammar

The above grammar is composed by Static Grammar rules. Though, there might be the need of having rules filled at runtime. These are called dynamic grammar rules because they can be updated dynamically. A dynamic rule is also useful to kept separate static grammars from lists, so that both can be easily managed. For example, instead of having a list with all the possible places inserted into the weather grammar above, the places are loaded in runtime from an external list (fig.11).

```
<Dynamic Rule Place>   [Place.txt]
```

Figure 11 - Pseudo dynamic rule

All these actions mentioned above take place in SAPI and are completely transparent to the engine. The SR engine is agnostic to whether it uses a totally static grammar or dynamic rules.

2.3.5 Speech applications

Speech applications can be divided in several classes depending on their interface. There are four broad classes of speech applications requiring different user interface design:

- **Desktop** – Desktop speech applications include widely used computing environments, such as Microsoft Windows and Microsoft Office.
- **Telephony** - Telephony applications require Server-side speech applications, such as Microsoft Speech Server and Microsoft Exchange.
- **Home** - Home user interfaces are usually localized in the TV, living room or kitchen and introduce a great benefit since home appliances don't have a keyboard or a mouse and the traditional graphical user interface application can't be directly extended for this category. The traditional HCI set-up makes use of remote control devices, but these can be replaced by a speech interface.
- **Mobility** - In the mobile case, cell phone, PDA and automotive are the most important mobile scenarios, due to the physical size and the hands-busy and eyes-busy constraints.

2.3.5.1 Architecture

The typical speech application is commonly constituted by one or more engine that can be either a SR, a TTS or a SLU system and an application programming interface (API) used as a communication bridge between the engine and application, as illustrated in fig. 12. There can be multiple applications interacting with a shared engine via the speech API.

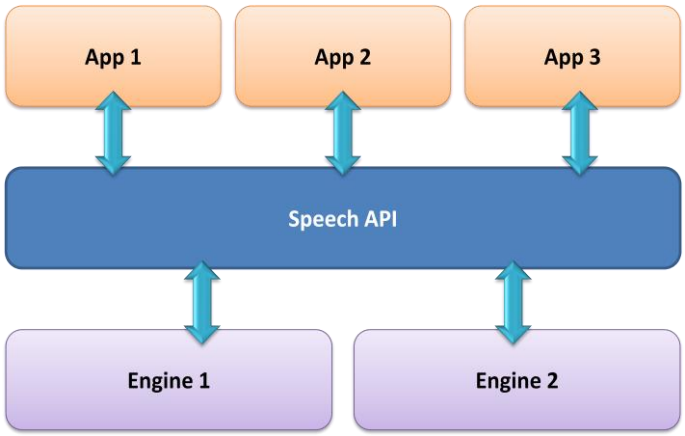

Figure 12 - Components of speech applications

2.4 Microsoft Voice Command

In this section are described the objectives and features of Microsoft Voice Command (VC), which is the main application of this project. In chapter 4 is described all the work performed with VC.

2.4.1 Definition

Microsoft Voice Command is a product for Windows Mobile (WM) 2003, WM5.0 (2005) and WM6 (2007) based Pocket PC, Pocket PC Phone Edition and Smartphone devices. This application allows the user to command and control the device using his/her voice without any previous user training. The application current version is 1.6 and is currently available in English-US, English-UK, German and French.

2.4.2 Application objective and features

Voice Command has the objective of changing the way people interact and think about mobile devices. Voice Command eliminates the large amount of button pushes normally associated with looking up information, dialling a number or doing any number of other tasks on a mobile phone.

The application provides a spoken language interface that allows the user to:

- Place phone calls (through dial or contact numbers)
- Look up contacts.
- Change profiles.
- Check missed calls.
- Change ringer volume.
- Get current date/hour.
- Get battery level.
- Get signal strength.
- Get calendar information.
- Play music.
- Get device status.
- Start programs.
- Check calendar appointments.
- Announce emails and SMS.
- Supports Bluetooth earphones

In the functionalities provided by VC there is a set of options, available in the control panel that allow the user to personalize the application. The user can, for example, choose if he wants VC to announce incoming calls, emails or reminders, enable call and dial confirmations, or just disable the application. A detailed description of VC features and options is available in [VCUserGuide].

2.4.3 How does it work

On Pocket PC, the application can be associated with an "Application Launch Button" on the device and the button will be enabled for "Push to Talk" (PTT). This can be configured in "Start > Settings > Buttons". On SmartPhone there is the possibility of configuring an application option to use the "Record button" or the "App buttons" (this does not work on all Smartphone devices). Once the application has been activated, the user can input a voice command. When the user doesn't know what to say or doesn't remember the voice commands, he/she can just say "help" or "what can I say" and the

application will instruct him. In version 1.6 the user can already use a Bluetooth device for voice input.

2.4.4 Target audience

The application is designed for people on the move. According to [VoiceCommandWeb] eighteen percent of mobile time matches the "busy hands, busy eyes" scenario. The car is only one example of this situation. When we are driving our eyes are focused on the road and our hands on the wheel, so a spoken language interface can be an alternative to interact with our mobile device. Based on the results analysis of Chapter 4 it can be concluded that Voice Command is useful for those who have a large contact database.

2.4.5 Competitors

In comparison with other competitor applications, Voice Command has better recognition accuracy as it is shown in the graph below (fig. 13).

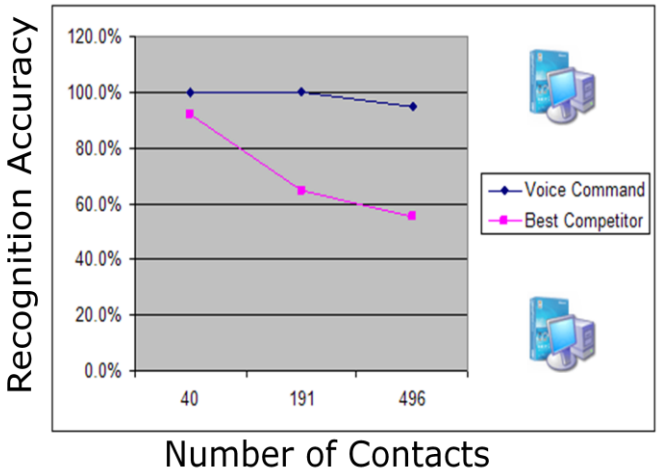

Figure 13 - Comparison between VC and the best competitor (2004)

When the number of contacts is increased, the number of utterances to be recognized is also increased, since we need to add the contact name to the grammar. A larger grammar represents less accuracy, but in comparison with other similar applications Voice Command presents a smaller decrease of accuracy.

2.4.6 Long term key objectives

For future versions of Voice Command it is expected that the application includes improvements in size, speed and memory usage. Currently the application occupies a relatively large amount of memory (3MB ROM + 5MB RAM) for the EN-US version and (10MB ROM + 5MB RAM) for the PT-PT beta version. This amount scales linearly with contacts or media files.

There is also a set of functionalities that can improve user experience:

- Full email triage – delete, file, reply and forward
- Dictation of messages (emails, sms)
- Search capability – Find contacts, music, messages, etc.
- Supported full suite of languages
- Expansion of the supported scenarios
- Etc

2.5 Other commercial solutions

In this section several existent commercial speech solutions for mobile devices are described. It is presented two concurrent solutions to VC and other solutions that show which kind of speech solutions also exist in real world.

2.5.1 Cyberon Voice Commander

Voice commander [Cyberon07] is a product similar to Voice Command that appeared in the market, by the hand of Cyberon Corporation, when Microsoft VC was still in version 1.5. The key features provided by the application are:

- Speaker-independent voice recognition.
- Mandarin and English bilingual recognition.
- Natural Voice Dialling with support for more than 1500 contacts (according to the specifications)
- Supports Bluetooth earphones.
- Voice query contact.
- Voice Shortcuts.
- Voice Digit Dialling.
- Voice control of Media Player.

- Voice controlled SMS reader.
- Voice controlled Calendar reader.

2.5.2 Cyberon Java Talking Dictionary

Another speech solution presented by Cyberon is the Java Talking Dictionary. The objective is to turn a mobile phone into a full-featured electronic dictionary that through a natural human-like voice gives the user a correct learning of the word's pronunciation. It has a vocabulary of 44,000 English words and 16,000 Chinese words, enough for general purpose learning. Pronunciation is available for every word (including Chinese). The application has smart text input (allows the user to enter text by pressing only one key per letter) and history tracking (saves the most recently used words). Figure 14 illustrates the graphical interface application.

Figure 14 - Java Talking Dictionary screenshots

2.5.3 Vsuite

Vsuite is a speech application, from Voice Signal, similar to MS Voice Command and Cyberon Voice Commander, except that it provides a smaller set of features [VoiceSignal07]. VSuite is a voice dialing and command and control platform that does not require any training. VSuite allows the user to lookup contact information, dial any name or number in the contact list, address text messages and pictures, digit dial any number and access features on the phone or carrier services in a single command.

2.5.4 VSearch

VSearch provides access to vast search capabilities of the web from a mobile phone. VSearch is an advertiser-supported service that gives mobile phone users access to

directory assistance, ringtones, music, games, weather forecasts, sports scores, the latest stock quotes, maps and even special offers from advertisers.

VSearch system sends the processed audio information to the VSearch server via the mobile data network where the user's query is converted into text through a speech recognition engine.

2.5.5 VoiceMode

VoiceMode is a dictation application that runs on a mobile phone and allows users to dictate messages into their mobile device, rather than being forced to create messages using the small constricted keypad on most mobile devices. VoiceMode provides a fast, easy and more natural alternative to multi-tap, T-9 [NuanceT9] or other text creation input methods.

2.5.6 VSpeak

VSpeak is an intelligible text-to-speech application, capable of running within the resource constraints of a mobile device. VSpeak can read text messages, web pages or any other text on a VSpeak enabled handset. By making it possible for people to easily and safely receive text messages on a mobile phone, even when their hands and/or eyes are otherwise occupied, VSpeak enhances an individual's ability to use a mobile phone regardless of where they are or what they are doing. VSpeak allows that the visually impaired will be able to receive text messages and use operator services on their mobile phones, which would otherwise be unavailable to them.

2.5.7 Comparison and analysis

When analysing the speech applications for mobile devices here presented, one can find a large set of common features between them. These features all share the same objective: facilitate the usage of the device.

Voice Signal has presented more specialized solutions (having a larger set of applications), while Microsoft and Cyberon focused all features in one solution. When comparing the amount of features, MS Voice Command takes the lead, followed by Cyberon. However, Voice Signal ships applications with features inexistent in the other

presented solutions, such as the dictation feature. Table 2 provides a comparison between these applications.

Product	Higher number of Features	Largest language portfolio	Speaker independent	Visual feedback	Storage size	Runtime memory
Microsoft Voice Command	X		X	Not always	3 Mb	5 Mb
Cyberon Voice Commander			X	Yes	2.46 Mb	470 Kb
Voice Signal VSuite		X	X	Yes	n.a.	n.a.

Table 2 - Application comparison

The major problems of Voice Command, in comparison with his direct opponents, are the memory it occupies, which is considerable for a mobile device application, and poor visual feedback, which can be a problem or not depending on the type of user. In a "busy eyes" scenario the visual feedback is disregarded, so from that point of view it's useless. However, with a specific type of users, such as those with limited experience with speech interfaces users, the visual feedback becomes a requirement.

2.6 Conclusion

In this chapter an analysis of the spoken language technologies used in this work was done. The spoken language world is composed by an immense background theory. Therefore the student chose only to describe technology that is used along the project.

This chapter can be divided in two distinct parts, one where it is described the used technology that allows the development of speech applications and another where are several speech applications for mobile devices are presented, including Microsoft Voice Command. In the first part of this chapter the main components used in the course of this work are described. This includes ASR and TTS systems, HMMs and speech applications components. In the second part of this chapter, Voice Command and other available similar commercial applications are described and compared. The applications here presented allow the reader to have a view of how speech can be applied in mobile devices, as well as the advantages of speech interfaces in mobile devices. From the

comparison result it can be concluded that Voice Command has the disadvantage of occupying more memory but has the advantage of providing a larger set of features.

Chapter 3

Development of a Speech Interface for Mobility

Given the background theory on Spoken Language Technologies in the previous chapter, the procedures and solutions that constitute this project will now be described. First it will be described how the EP SR engine for mobile devices was localized and compiled, followed by an explanation about the methodologies in the development of speech applications for mobile devices. Finally, some experimental results are presented.

3.1 Speech engine localization

Localizing a speech engine in order that it understands a new language is a daunting task. To accomplish this task it is used a specialized tool named *Autotrain*. This tool can be described as a set of tools designed to help developers localizing a SR engine. *AutoTrain* facilitates this task by providing a framework which developers and linguists can use. The localized engine can be used on MS Speech Server, Windows XP, and Windows CE.

3.1.1 Training overview

For the speech engine localization one needs to train acoustic models, so that automatic speech recognition can be conducted. Acoustic model training involves mapping models to acoustic examples obtained from training data (corpus).

3.1.1.1 Corpus specifications

The corpus data is the base of acoustic models and subsequently influence speech recognition error rates. In this section, the characteristics of the used corpus – SpeeCon EP are presented.

The Corpus database, used in the acoustic models training was collected in the framework of the SpeeCon project [SpeeCon99]. The database contains speech data in European Portuguese, recorded at 16 KHz with 553 different adult speakers, 48% male and 52% female. The speakers present accents from three different regions of the Portuguese mainland – southern, central and northern, as shown in table 3. The subject's ages are shown in table 4.

Region	Male Speakers	%	Female Speakers	%
Southern	138	25.0%	152	27.5%
Central	65	11.8%	65	11.8%
Northern	63	11.4%	70	12.7%
Total	266	48.0%	287	52%

Table 3 - Speakers distribution of over regions

Age interval	Male Speakers	Female Speakers	%
15-30	127	143	48.8%
31-45	91	102	34.9%
46+	48	42	16.3%
Total	266	287	100%

Table 4 - Speaker distribution over age groups and sexes

The recordings took place on four different environments:

- Office
- Entertainment
- Public Place
- Car

The office environment recordings were made in offices with a background noise between 30 to 60dB. The recordings include telephone calls and other typical office environment sounds but do not include meetings and/or discussions.

The entertainment environment recordings were made in common living rooms with soft furnishings and domestic audio-visual equipment. Half of the recordings have the audio equipment in operation positioned behind the speaker, generating approximately 50dB (measured at the speaker position).

Regarding the public place recordings they include common street noises such as, clock-tower bells, fountains and garbage collection. The noise range limitations of 45 to 90 dB were met.

The car recordings were carried out with the speaker seated in the front passenger seat and the recording operator in the back seat, under five different recording conditions:

- Vehicle stationary, engine switched off;
- Vehicle stationary, engine running;
- Vehicle being driven in city conditions (average speed of between 30 and 70 km/h);
- vehicle being driven in main roads, but not freeways, with an expected average speed of between 60 and 100 km/h;
- vehicle being driven under highway conditions, with an expected average speed of between 90 and 130 km/h.

Table 5 shows the speakers distribution over the recording environments.

Environment	Male Speakers	Female Speakers	Total
Office	93	110	203
Entertainment	31	44	75
Public	94	106	200
Car	48	27	75

Table 5 - Distribution of speakers over main environments

SpeeCon contains about 77 hours of recordings of the following types:

- Noise and silence.
- Elicited spontaneous items such as dates, times, proper names, city names, letter sequences, answers to questions, telephone numbers and languages.
- Read speech composed by phonetically rich sentences/words.
- Core words, which can be divided in application specific words/phrases such as messaging, internet browsing, editing, output control, automotive, routing, etc, and general words/phrases such as , isolated digits sequence, natural numbers, analogue and digital dates/time, etc.

In the training procedure, the available corpora was used and there was no selection regarding the type of target application, due to the small amount of specific data and project deadline.

3.1.1.2 Data pre-processing

At this stage of the training procedure we convert speech waveform files to MS waveform format and place all word level transcriptions into a MS transcription file (HYP). The HYP contains information about each acoustic data file to be used in model training. It respects a pre-determined format where each transcription is associated to the respective .wav file, the session/speaker identifier and gender. The HYP also contains annotation information, which is composed by written tags that mark or describe the speech sounds, e.g. prompt echo, background noise or speech, misspelled words, etc.

The HYP file generation is based on Corpus metadata, referred here as MS Tables. More concretely, the generation of first HYP version, usually called raw HYP, is obtained from two relevant MS Tables – UtteranceInformationTable and SpeakerInformationTable - these two tables contain all the relevant corpus information about each recorded utterance, speaker identification, microphone, recording environment, dialect, gender, transcription. The next step is to normalize the training transcriptions. The normalization consists in selecting and preparing the raw HYP file information to train.. These steps are executed using Autotrain batch scripts and configured in a XML file, as depicted in fig.15.

```
<HypStep name="GenRawTrainHyp" run="true" />
```

Figure 15 - Line responsible for the HYP generation in Autotrain configuration file

Figure 16 illustrates the procedures described above. The Lexicon generation procedure is described in the following section.

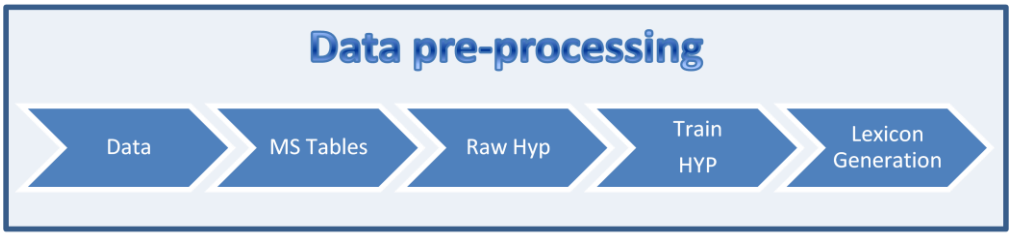

Figure 16 - Data pre-processing flow

3.1.1.3 Lexicon generation

A pronunciation lexicon needs to be produced which includes all the words found in the HYP file. A pronunciation lexicon lists the phonetic transcriptions of the vocabulary. In this stage the annotation tags are mapped to the corresponding sounds.

The transcribed words that aren't found in the phonetic dictionary are generated by G2P and hand checked by a linguistic.

3.1.1.4 Training

In this section it is presented a description of the training process and the basic model topology used. The training with Autotrain can be divided in several stages. Once a model prototype is defined, the speech parameters coding takes place. At this phase the ".mfc" files are produced, given the set of ".wav" files. These files contain speech signal parameters called Mel-Frequency Cepstrum Coefficients (MFCC) [Mermel76]. An MFCC is a speech parameter defined as the real cepstrum of a windowed short-time signal derived from the FFT of that signal, where a nonlinear frequency scale, approximated to the behavior of the auditory system, the Mel scale is used. The figure below illustrates the feature extraction process by taking a speech signal and split the waveform into 10ms frames. Each frame encodes speech information in a form of a feature vector.

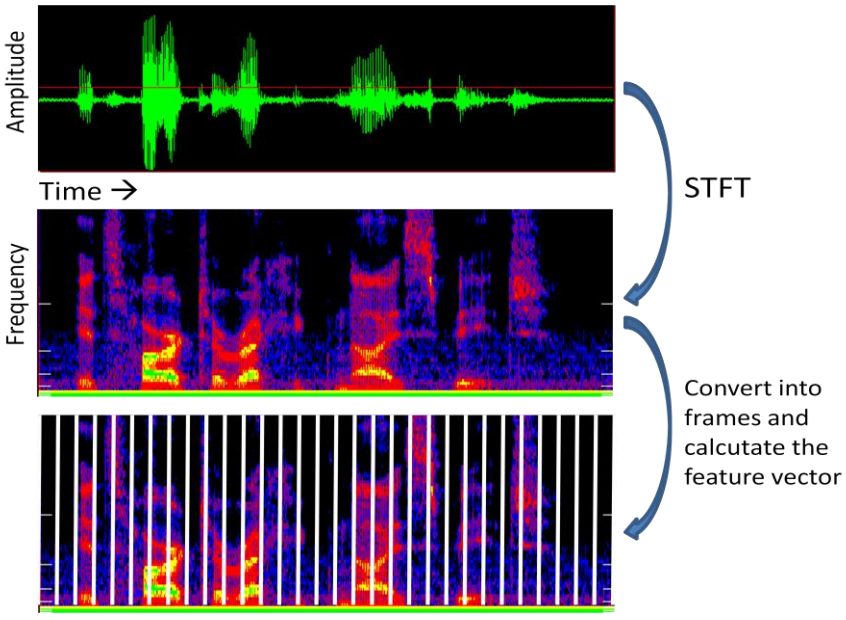

Figure 17 - Feature extraction process

In the following stage a set of HMMs is initialized from the scratch, given a set of word level transcriptions, lexicon, MFCCs and a phone set.

The kind of phone models trained are context-dependent, more exactly triphones.

The HMMs are composed by three emitting states and two non-emitting states. The emitting states have continuous density multivariate output distributions, as illustrated in fig. 18.

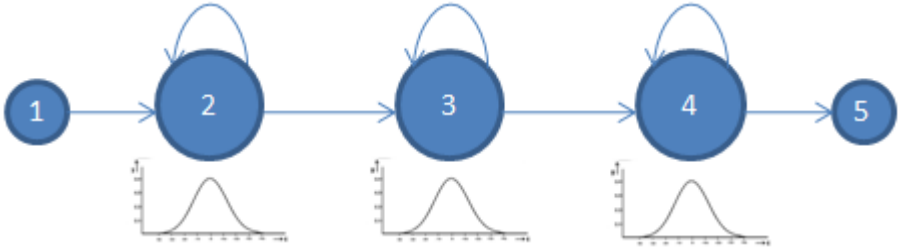

Figure 18 - Used HMM model topology

Each emitting state consumes at least one frame of speech, in this case 10ms. Gaussian probability density functions are associated with each emitting state, representing the speech distribution for that state. The two non-emitting states do not consume any frame of speech. They are used to provide entry/exit to/from an HMM. The transitions in this model are either right to left, linking one state to the next, or self-transitions (allows the states to be occupied for several time frames). The output distribution of the models built is represented by several Gaussians. A single Gaussian is a very rough approximation to the true distribution of speech sounds. To model the speech variation more closely, the state-parameters are represented by weighted multiple Gaussian-mixture components where the weights of all the mixture components of a single state sum to one. Total models size is dependent on the number of mixtures and the number of senones used. Considering the used amount of data and internal Microsoft guidelines, it was defined that the system would have a total of 2000 senones and 8 final mixtures.

3.1.1.5 Compilation

After the training is completed the models are compiled and registered by changing the following options represented in fig. 19 which is correspondent to the Autotrain configuration file.

```
<Stage name="Training" run="false" />
<Stage name="Compilation" run="true" href="Fra.CompileRegister.xml" path="InputsDir" />
<Stage name="Registration" run="true" href="Fra.CompileRegister.xml" path="InputsDir" />
```

Figure 19 - Changed options for model compilation (Autotrain configuration file)

The compilation stage builds the files described in table 6.

Files	Functionality

L2070.phn	Phone set
L2070.smp	Senone map. Maps triphones to SSIDs to senone ids.
L2070.cw	Cross word models map
L2070.ww	Whole word models (not used)
Lsr2070.lxa	G2P rules
AI032070.am	Acoustic model. Contains coded means, variances, and weights for each senone.
L2070.ini	Plaintext-format runtime-switchable settings

Table 6 - Compiled files

In order to obtain Compact Edition (CE) models the compression option was activated in Autotrain configuration, during the compilation stage.

3.1.1.6 Registration

At this stage the engine registration is performed, being divided in the following actions:

1. Registering the engine dlls;
2. Registering the Phone Converters;
3. Setting up the engine Token in the registry with the correct attributes for the platform;
4. Setting up the Engine Token to point to the compiled data files.

The engine token is registered under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Recognizers\Tokens\MS-2070-70-WINCE, as illustrated in fig. 20.

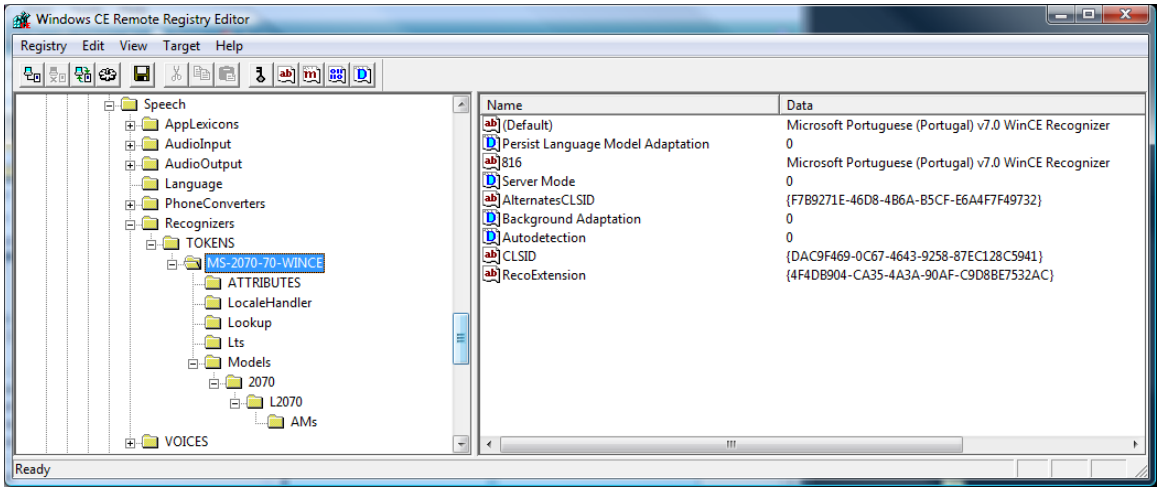

Figure 20 - Engine registry

Autotrain provides the means to register the engine. However, the user can define the desired registry template. In this case is the student defined the:

- Language – PTG (European Portuguese) - and the correspondent code – 2070 (816 in hexadecimal);
- Platform – CE;
- Path that points to the files.

3.2 Speech application development

In this section the methodologies and concerns towards the development of speech applications for mobile devices are presented. This section allows understanding the implications and conditions of developing applications for a CE platform.

The CE platform commonly implies the deployment of the application on a mobile device. The characteristics associated with a mobile device, from a speech developer point of view, are usually, in comparison with a desktop platform, less memory and performance, background noise variation (depending on the place or environment), device positioning towards the user, device model details (e.g. buttons), etc.

To approach the theme of speech applications for mobile devices an example application was developed - Pocket Reco.

3.2.1 Pocket Reco

In this section the stages of development of a speech application that allows a user to place a phone call to a contact, using a Pocket PC device, are described. The intention

is to present the architecture and the relevant architectural decisions, considering the language, mobile environment and speech recognition procedure.

The example application is referenced as "Pocket Reco" and it is a C++ application developed for Pocket PC devices, interacting with the user through a spoken language interface in EP and influenced by applications such as Microsoft Voice Command [VoiceCommandWeb], Cyberon Voice Commander [CyberonWeb] and Voice Signal [VoiceSignalWeb].

The application allows the user to place phone calls to any contact in the MS Windows contact list and to consult his/her agenda. It was developed using Microsoft Visual Studio 2005 [VisualStudioWeb] and MS Windows Mobile 5.0 Pocket PC SDK [MobileSDK]. The application is localized for EP.

The number of speech recognition applications in EP is very restricted due to the lack of linguistic resources (the ones that exist are expensive [ELRA]). The amount of existing applications with spoken language interfaces in EP in the market is reduced. If we consider only mobile devices applications, that number decreases even more. It's important that Portuguese users have the possibility to use speech recognition and speech synthesis applications in European Portuguese, especially those with special requirements that depend on this type of applications to use a mobile device.

3.2.2 System description

A Pocket PC or a Smartphone device provides innumerous features but placing a phone call is commonly the primary and most used function of a mobile device.

When the user initiates the application, by pressing a button (associated with Pocket Reco) or by running the executable file, the speech recognition and text-to-speech modules are initiated. To notify the user, an icon is loaded to the notifications tray and a spoken message is synthesized. At this point the application expects the user to input a voice command. If no expected command is recognized there is a predefined timeout. Once the voice command is recognized, a set of actions associated with the command are executed. When these actions are finished the application ends, unloading all the resources. The user can input commands to make phone calls or check the calendar. When the user chooses to make a phone call by saying the respective command followed by the contact name, it is probable that the contact has more than one phone number (mobile, home, work, etc). In this situation the user is notified to choose between the available numbers by saying the respective

command. The activity diagram below (fig. 21) describes the flow for making a call to contacts' phone number.

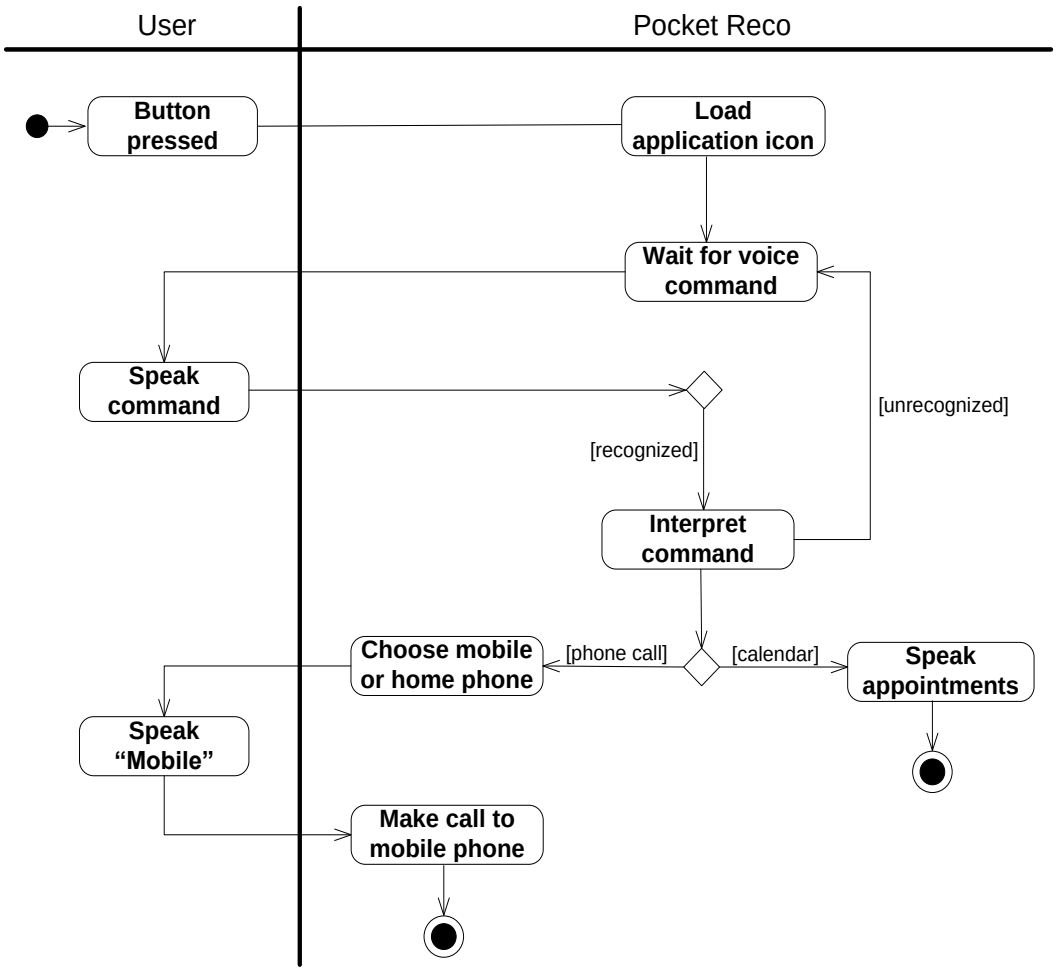

Figure 21 - Activity diagram for the application features

On Pocket PC devices powered by Windows Mobile (WM) there is the possibility to associate a button to an application. i.e., when the button is pressed the associated application is launched. The process to do this is simple. When installing the application we just need to place a shortcut (that obviously points to the desired application) on *\Windows\StartUp* and the application will be available in the button association menu.

3.2.3 Architecture

The application architecture was based on the one presented in fig. 9 and on the integration necessity of an SR and TTS engine.

Pocket Reco contains a SR and a TTS module, which communicate with the respective engines through SAPI. The SR module uses SAPI API for speech

recognition and TTS module uses SAPI API for speech synthesis. SAPI communicates with the engines through a device driver interface [SpeechSDK]. Both SR and TTS engines contain a SAPI interface layer, allowing using the SAPI runtime. Figure 22 illustrates the interaction of the application with the engines, through SAPI.

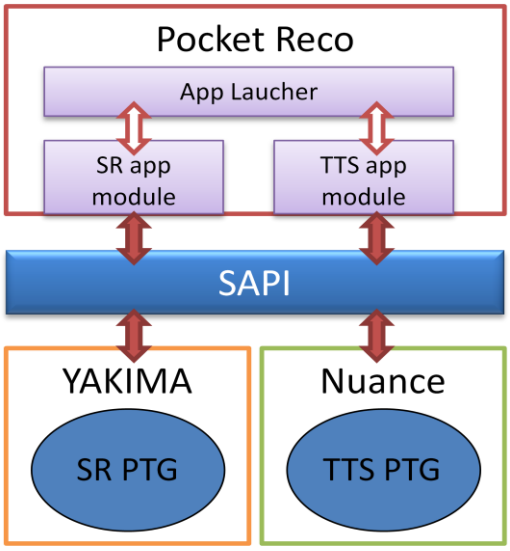

Figure 22 - Diagram of the speech architecture

The integration of the SR and TTS engines in the application required the installation on the device of the following products:

- YAKIMA Speech Recognition Engine for Windows CE
  - Located in European Portuguese, in the scope of this work
- ScanSoft RealSpeak Solo SDK for Windows CE
  - Portuguese Voice – Madalena, provided by Nuance

Usually, speech recognition engines for mobile devices present lower performance comparing with desktop engines. Optimizations, like fixed point calculations, lead to less recognition accuracy. Due to this, applications should perform a compensating effort in terms of recognition, e.g. minimize the number of grammar items, take in consideration the speech corpus used to train the acoustic models and model the application flow to avoid errors.

A performance improvement could also pass by modifying the moment when the initiation process is made. In the current implementation the application is initiated, as well as the required modules, when the user wishes to use it. Another possible approach would be to perform all the possible initialization work when the operating

system of the device starts. When the device is turned on, the application is initiated and waits for the user notification to use it. This way the processing time between pressing a button and the notification to enter a command is diminished about 50 to 80%. The main disadvantage however resides in the occupied resources.

3.2.4 Implementation

SAPI provides a set of COM interfaces that allow communicating with the engines. For this reason, when using managed languages such as C# or Java, a wrapper is required in order to perform the connection between the unmanaged and the managed world. With C# the interoperability is provided by COM interop wrapper classes. Whenever a client calls a COM object method the runtime creates a runtime callable wrapper (RCW) [COMWrap]. RCWs abstract the differences between managed and unmanaged reference mechanisms. It also provides adequate marshaling for calls that cross the boundary between COM and the .NET Framework. In Java we can use Java Native Interface (JNI) [JNISpec] to interoperate with applications and libraries written in other programming languages, such as C, C++, and assembly.

These wrappers facilitate the access to the unmanaged world provided by SAPI, but there is a price in terms of performance that comes in the form of parameter marshaling, object wrapper creation, etc. Since we are dealing with a limited platform in terms of performance and memory using managed languages can have an impact on the application performance. One of the advantages of using the managed environment, besides the obvious object memory management, resides on new APIs (e.g. Direct3D Mobile, DirectDraw, Global Positioning System, ActiveSync Interaction, etc) and existing operating system features that are now exposed as managed APIs (e.g. Telephony, Outlook Mobile, Messaging, etc).

For the reasons stated on the previous paragraphs and because most of Voice Command source is written in C++, this example application was developed in C++. The objective of this application was not “to reinvent the wheel” but to understand the mechanisms that “makes the wheel turn”.

The application implementation is divided into the following modules:

- PocketRecoApp – Main application module.
- SR - Provides interaction with the SR engine.
- TTS - Provides interaction with the TTS engine.
- MobileDevice – Provides all the features associated with the device.

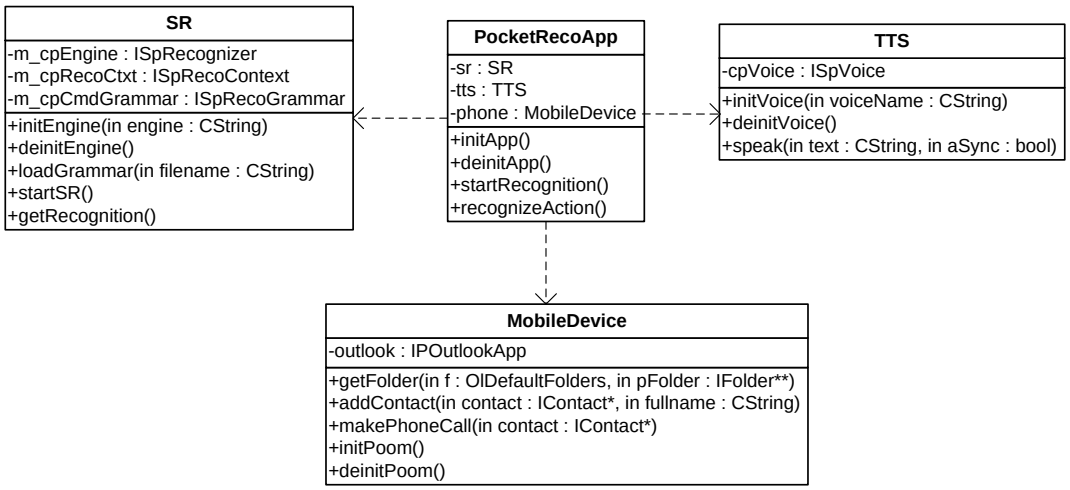

Figure 23 - UML diagram of the implementation

Figure 23 represents, through an UML diagram, the modules that compose the application. PocketRecoApp is the main module where all the feature/business logic is written, being also responsible by initiating/terminating other modules, such as TTS and SR. The SR module provides services related with SR engine, using SAPI. This includes loading or building grammars, initiating/terminating the recognition engine and return semantic information about the recognition. The TTS module interacts with the TTS engine also through SAPI and is responsible for initiating or terminating the engine and providing the service that allows synthesizing written text. The MobileDevice module provides services related with the device, such as initiating Pocket Outlook Object Model, obtaining Contacts folder or executing the actions associated to each command, like making a phone call.

3.2.4.1 API for Text-to-Speech

Applications can control the TTS engine using the ISpVoice Component Object Model (COM) interface. Once an application has created an ISpVoice object it only needs to call ISpVoice::Speak to generate speech output from some text data. In addition, the IspVoice interface also provides several methods for changing voice and synthesis properties such as speaking rate, output volume and changing the current speaking voice. The IspVoice::Speak method, can operate either synchronously (return only when completely finished speaking) or asynchronously (return immediately and speak as a background process). When speaking asynchronously

(SPF_ASYNC), real-time status information such as speaking state and current text location can be polled. Also while speaking asynchronously, new text can be spoken by either immediately interrupting the current output (SPF_PURGEBEFORESPEAK), or by automatically append the new text to the end of the current output.

This interface also allows flow control, such as pausing, resuming or skipping speech synthesis. An example where flow control is important is when one wants to stop the TTS once a command is recognized. This allows the user to realize that his/her spoken command has reached the application.

3.2.4.2 API for Speech Recognition

The interface *ISpRecognizer* enables applications to create different functional views or contexts of the SR engine. Each recognizer object is connected to one or more recognition contexts - *ISpRecoContext*. Through the contexts, the recognizer can control the recognition grammars to be used, start and stop recognition, receive events and recognition results/information.

Similarly to *ISpVoice* which is the main interface for speech synthesis, *ISpRecoContext* is the main interface for speech recognition. Like the *ISpVoice*, there is the *ISpEventSource*, which is the speech application's vehicle for receiving notifications for the requested speech recognition events. This interface enables applications to control aspects of an SR engine and its audio input. Each object implementing this interface represents a recognizer for a single engine.

A speech application must also create, load, and activate an ISpRecoGrammar, which essentially indicates what type of utterances to recognize, i.e., dictation or a command and control grammar.

3.2.4.3 Events

SAPI communicates with applications by sending events using standard callback mechanisms (Window Message, callback procedure or Win32 Event). For TTS, events are mostly used by applications to sync to real-time actions as they occur such as word boundaries, phoneme or visage (mouth animation) boundaries.

Pocket Reco uses Win32 events, in speech recognition. When a word from the grammar is recognized, SAPI sends a notification to the application.

3.2.5 Grammar

The application uses a CFG grammar to parse the recognizer output. The grammar is composed by dynamic and static rules. The dynamic rules are empty when the application starts and their content it's updated in runtime. The separation between static and dynamic rule contents allows that the application starts and loads the static content, without loading the SAPI grammar compiler to prevent the delay in the start up sequence. When the application loads the dynamic content it forces SAPI to initialize the back-end grammar compiler [Microsoft Speech SDK].

In Pocket Reco there is a distinction between static rule content and dynamic rule content, in order to develop a well-designed grammar and to improve initial SAPI grammar compiler performance. The dynamic content is, in this case, the set of contacts names because they can only be accessed at runtime. The static rules of the grammar that need the contacts names contain a reference to the rule with the dynamic content.

3.2.5.1 Semantic-based recognitions

Speech recognition engines use CFGs to constrain the user's words to words that it will recognize. If the CFG is augmented with semantic information (property names and property values), a SAPI component converts the recognized word string into a name/value-meaning representation. The application then uses the meaning representation to control its part of the conversation with the user.

For CFG recognitions it is more important to recognize the semantics of what has been said than the actual words in the utterance. For example, recognizing "Please call John Doe" or recognizing "Call John Doe" has the same semantic value. The semantically significant parts of an utterance are denoted by assigning a semantic tag to a particular grammar rule.

There are a number of scenarios where semantic tags can be useful for an application such as Pocket Reco. One possibility is to use these tags to increase the robustness to failures. When a false recognition occurs, the application can detect the last semantic property returned and display an error message relevant to the attempted voice command. Another possibility can be increasing the application response time. When we have long voice commands the ambiguity can be broken before the voice command is complete, using the first unambiguous semantic property to be received.

Semantics-based recognition also facilitates the localization of speech applications. This technique supplies application independence towards the grammar, this is, the utterances to be recognized aren't "hard coded", and instead can be dealt with has resources.

3.2.6 Name matching algorithm

A spoken language interface with command and control modality, like Pocket Reco, should supply intuitive voice commands, from the user's point of view. This way, the user will not need to memorize the commands and the required words will be spoken naturally. The speech command responsible for placing a phone call to a contact in the contact list is composed by a static part, such as "Chama" ("Call"), that resides in the grammar file, and a dynamic part composed by the contact name. If we consider a user with a large contact list, he/she might not remember the exactly contact's name. In another scenario, we can consider the case of persons with many names (six and seven names occur frequently for the case of Portuguese native speakers), which might be called by their first, their last, their middle or any combinations of these names and more. For example, if we download a contact list from an Exchange server with a contact named João Paulo de Oliveira Esperança Gonçalves, he might be known as "João Paulo" by user "A" and "João Esperança" by user "B". Considering the functionality provided by Pocket Reco to make a phone call to a contact in the contact list, the correspondent command is dependent on the contact name. The developed name matching algorithm allows the user to say the contact name in any form he/she is used to. Instead of residing in the grammar only the first and last name of each contact, all different names from the contact list are placed into the CFG grammar in separate phrases, without repetitions. This will allow the recognizer to accept any combination of names, e.g. "Chama João Paulo Gonçalves", "Chama João Gonçalves", etc. After the recognition stage, the spoken contact name must be analysed. The application will access the contact list, which is, in this particular case, residing in memory (loaded when the application starts), and search, through this list, the contact that has the highest number of matches with the spoken name. With the proposed algorithm there is the chance of name ambiguities. When this happens, the algorithm either chooses the first name occurrence or gives to the user the chance of breaking the ambiguity, which is the most correct approach. This can be achieved by giving to the user the contacts possibilities and respective full names (through TTS),

asking him to choose the desired contact. The disadvantage of providing flexibility to the user's interface with this algorithm is the increase of the number of items in the grammar, leading to an increase of the word error rate in the recognition process. There must be a compromise between application flexibility and recognition accuracy, depending of the application user's group and the recognition engine.

3.2.7 Experimental results

In this section it is presented a subjective usability experiment that compares the accomplishment of a task through the use of a graphical interface and a speech interface. This set of subjective usability tests compares the usage of a graphical interface with the usage of a speech interface in performing a normal task in mobile devices. The tests were applied to a universe of 30 unpaid adult persons, 16 male and 14 female (fig. 24), from the Portuguese academia. Two groups, each with 15 persons, were considered: one group had prior experience with speech interfaces in mobility; the other group presented no previous experience with speech interfaces and the experience with mobile devices is resumed to cell phones. Both groups covered all the considered ranges of ages as shown in the table 7. Each person performed two tasks: calling to a predefined contact number that resided in a contact list with 20 contacts, through the graphical HCI and the same task through the speech HCI, e.g. "Call to Pedro Silva mobile phone". The usability experiment collected the time duration of the task as a metric that relates to its difficulty. Each user received an explanation with clear instructions and a demonstration on how to perform both tasks. The tasks were executed by each user in a random order, under the same conditions, with Pocket Reco running on a Pocket PC (HTC P3600- described in table 13), in the following environments: Office, Home, Party (Crowded room) and Car. Both groups presented similar average value of recognition accuracy - around 84% - with an advantage of 0,5% for the users with previous experience on speech applications in mobile devices. Table 8 shows the average time (in seconds) taken to accomplish the proposed tasks with a voice user interface (VUI) and a graphical user interface (GUI).

Age	**Users**	
	With experience	*Without experience*
< 24	4	4

24 - 29	6	2
30 - 35	3	3
> 36	2	6
Total	15	15

Table 7 - User distribution

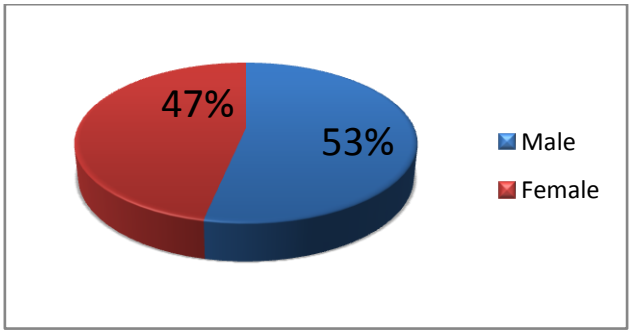

Figure 24 - Gender distribution

Users	GUI (seconds)	VUI (seconds)
With experience	15	12,2
Without experience	26,5	14,9
Total	20,8	13,6

Table 8 - Time taken to perform the tasks

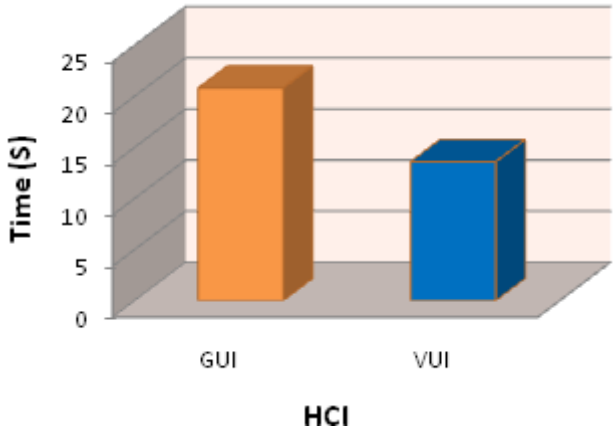

Figure 25 - HCI comparison

In the presented results there is a clear advantage, in the time metric - 34,6% lower - for the actions executed with the speech interface, as shown in fig. 25. Nonetheless, some of the users (especially those with no experience) had to execute more than one attempt, due to recognition errors, to accomplish the task using the speech interface. The recognition errors happened due to background noise and the way users interact

with the speech interface, such as bad positioning of the device, microphone covering, directing the voice not towards the microphone and pronouncing the words in an unclear way, often recurring to the use of abbreviations.

The time difference between the two types of interfaces decreases drastically according to the level of the user expertise with speech applications in mobile devices, as shown in fig. 26.

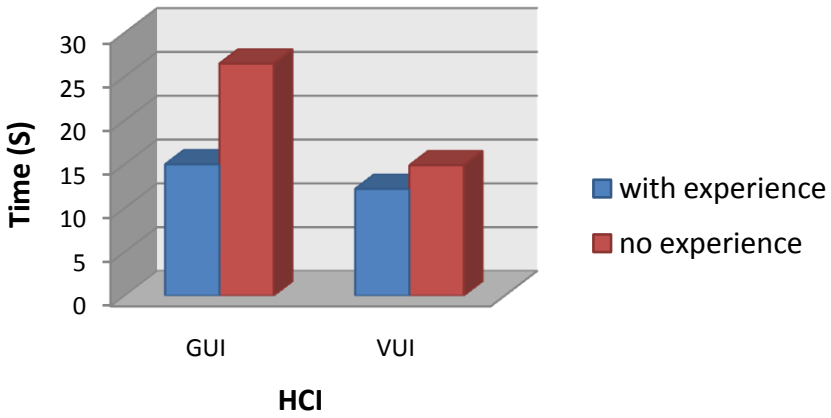

Figure 26 - Time difference between HCIs

3.3 Experiment analysis

The disadvantage of speech HCI, including in mobility, is that speech recognition is not 100% accurate. This fact may lead to recognition errors on which the application must expect and try to resolve, for example, ask for confirmation on a given voice command. In some cases, with noise environments or less experienced users, several attempts are needed to successfully accomplish the task. In speech recognition for mobility, the optimizations performed over the models and SR engine often decrease the components performance and the application implementation can compensate this decrease in quality. The grammar is an application component that the developer can use to optimize the recognition.

A set of optimizations to improve recognition accuracy and user interaction experience, in the framework of a spoken language interface in a mobile device has been described. In order to conclude about the approach, it was developed an example application that places a phone call to a contact, through a spoken language interface. The application interacts with the SR engine through SAPI and implements a name match algorithm to improve user experience. It was also presented some results of

experimental tests, based on user's experiment, to demonstrate the advantage of using a speech HCI versus a graphical HCI. In this work, we have shown that the set of words chosen to be recognized influence the application performance. This choice should be influenced by the acoustic models, the quality of the speech recognition engine and language specifications. The developer has the responsibility to balance the application according to its usage scenario and also needs to adapt the application to the user group. In this paper we presented an application that takes into account all these considerations, discussing the advantages and drawbacks of the various design choices. The name matching algorithm here presented is an example which trades recognition accuracy for a more intuitive and natural interface. Nonetheless the conducted tests show that we can manipulate the grammar to improve the performance of the algorithm. The conducted subjective usability tests demonstrated that users with no experience in speech applications and mobile devices can easily learn how to execute a simple call action in the device, through a spoken language interface. The presented results show that the time to accomplish a simple task, like placing a phone call, is 34,6% lower when using a speech interface as a replacement for a graphical interface. The time difference between the two types of interfaces decreases drastically according to the level of the user expertise with speech applications in mobile devices.

Chapter 4

Voice Command European Portuguese Localization

Voice Command is a fully developed speech application (already described on section 2.4) that was initially localized to English-UK, English-US, French and German. The "Voice Command in European Portuguese" is a MLDC project [VCProject] that has the objective of releasing a beta version of the product localized to European Portuguese. This section describes the proceedings and techniques used in the application's localization to EP.

4.1 Behavior Localization

Voice Command has a behaviour that is tailored for an US English target audience. The behaviour needs to be reviewed for cultural issues, target audience and appropriateness when localizing for other languages. This potentially affects all files. If the direct translation is not compliant with the specification, multiple changes throughout the product are needed. This stage has the goal of enabling a safe, natural speech user interface for existing device features on Pocket PC and Smartphone.

4.2 Initial status and resources

A study of the application was made before the starting the localization process. This consisted in acquiring all available knowledge (objective, features, limitations, etc) about Voice Command. Section 2.4 shows the relevant information (for this work) gathered in that analysis.

The amount of available information about Voice Command is very limited. Most of the gathered documents contain only commercial and marketing facts/data. There is no accessible exposition of the application architecture, modules, development techniques or any kind of technical advice.

All the facts presented in the following sections were deduced from the code tree, which is composed by more than 24000 files and 1800 folders.

4.3 Architecture

Voice Command can be decomposed in several modules, each one associated to a feature group. Figure 27 illustrates this separation presenting a simple view of the application, organized by features.

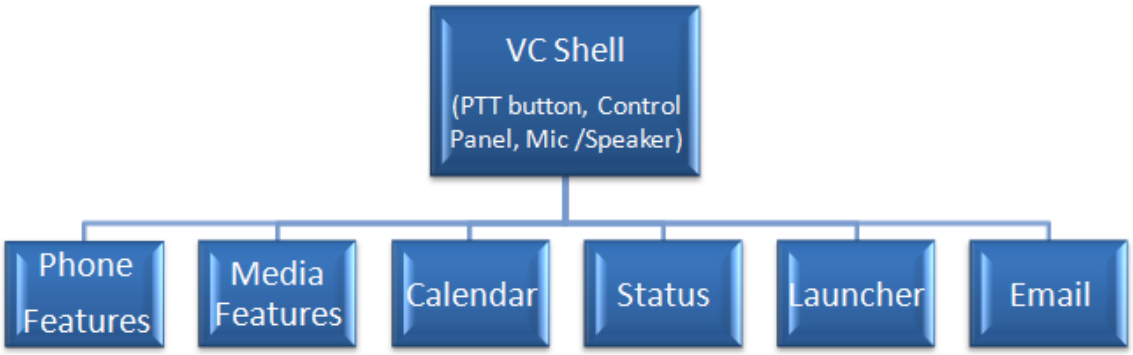

Figure 27 - Features architecture

The Shell module represents all the logic responsible for the interaction with the user such as the PPT button, options frame, microphone and speaker and therefore it is placed on top of all the modules. When the user wants to use the features he/she must pass encapsulated logic by the Shell.

The name of the modules is easily related with their function, though only a short description for each one will be presented as follows:

- **Phone** – contains all the logic for placing and announcing a phone call. It also interacts with the contact's manager.
- **Media** – contains a media manager, which represents the entire collection of media present in the device, an mp3 and wma parser, and a handler for media related events.
- **Calendar** – provides services such as listening to calendar and announcing reminders.
- **Status** – responsible for announcing device status (battery, signal strength, etc)
- **Launcher** – allows launching applications resident in the device
- **Email** – announces not only emails but also SMS messages. It also interacts with an email manager.

All modules contain event handlers (for the different event types) and a list with the semantic properties (names and identifiers) of the correspondent grammars, acting as a bridge between the grammars and the code.

Figure 28 presents another view of the application, less feature oriented (when compared with the above figure). In this figure, the "Shell" layer also represents the interaction with the user and all the application logic resides in the layer referred as "Main". It can be seen as a business layer. The resources gather all text definition associated with the application, e.g. text for speech synthesis, graphical labels, installation instructions (on the device), being an obvious target for localization. The set of word normalizers and grammars is here referred as skins. The foundation classes represent all lower level services such as:

- Grammar manipulation - loading grammars from file or resources, load back-end compiler, add/remove items, etc.
- Speech services - initiating SR and TTS engines, recognition, etc.
- Interaction with the device - announcements, information retrieval, etc.
- Recording audio
- Etc...

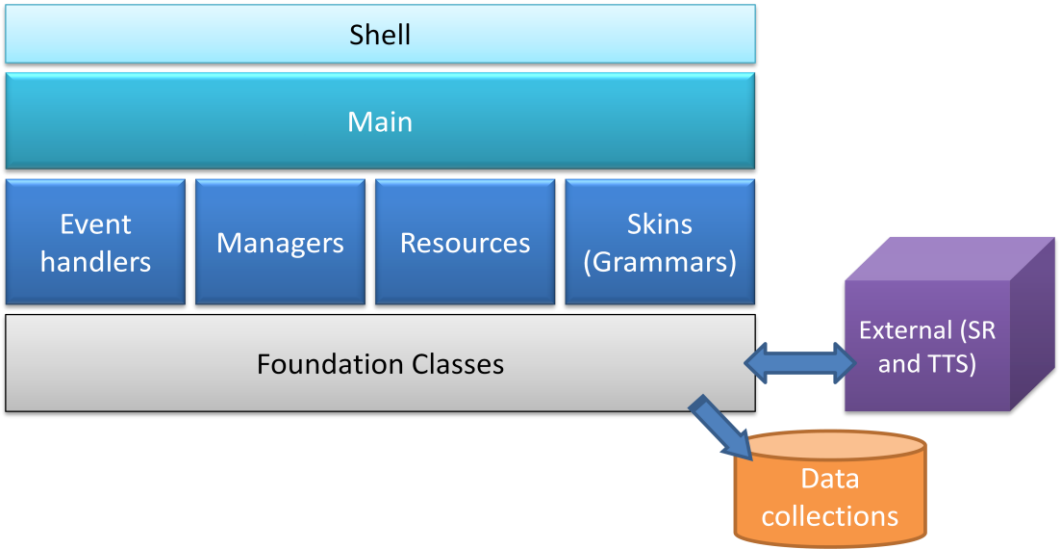

Figure 28 - Application deconstruction

Voice Command interacts with the engine through SAPI in a similar form as described in 3.2.3, being required for the engines to be SAPI compliant.

4.4 Development environment

4.4.1 CoreXT Development Environment

Voice Command uses the CoreXT Development Environment. CoreXT is a layer of rules, processes, features and enhancements on top of the NT build environment. It is an alternative infrastructure for developers, testers, builders and labs alike. This environment is completely generic and decoupled from the applications being built and provides large extensibility, consistent organization, tools and processes. Unlike the core NT build environment which is dedicated solely to satisfy the requirements of the Windows NT group, CoreXT is a community effort aiming at providing a functional build system for all development groups at Microsoft.

Through CoreXT, the depot administrator has created a branch with source access to the VC 1.6 tree, for the localization process. The access to the product's code has been supplied through an enlistment. An enlistment is a source-depot (see next section) client view on all or part of the product's code, the build tools and the support infrastructure.

The application tree, which was deployed in the branch, has the following structure:

<table>
<tr><td>build</td><td>The build directory contains the necessary infrastructure to overwrite includes, options and processes.<table>
<tr><td>sources.all</td><td>Overwrite CoreXT/NT build environment compile-time defaults; this file should be included by all sources files.</td></tr>
<tr><td>paths.all</td><td>Overwrite CoreXT/NT build environment path defaults; this file should be included by all sources files.</td></tr>
<tr><td>makefile.inc</td><td>Add specific makefiles to build dummy targets, the CoreXT distribution uses a makefile.inc to define the way targets are built.</td></tr>
<tr><td>placefile</td><td>Defines target directories for placing built targets.</td></tr>
<tr><td>preenv.cmd</td><td>Defines custom actions to run when a new build environment session is opened.</td></tr>
</table></td></tr>
<tr><td>private</td><td>All source code goes into this directory. Usually, a new directory is created under private for each new component or set of components.</td></tr>
</table>

private/A_mode	Contains the source code and represents the project directory.
public	Directory for publics, components being distributed and externals.
public/ext	Contains all external dependencies and tools, including ntpublic, tools, perl, sdepot, etc.
public/ext/bin	The compiler, linker, resource compiler, nmake and other tools required for producing builds.
tools	General depot and CoreXT tools.
tools/coretools	Minimal files required for an enlistment.
tools/build	Default placefile, makefiles and localization files. This directory contains the default sources.all and paths.all. To overwrite those defaults we use the build directory at the top level.
tools/build/automation	The automation directory contains the extensions for daily builds and for the current environment. It allows the extension of the daily build process with actions such as copying of produced bits to a release share or sending mail once a build has completed.
tools/path1st	The first path for scripts and executables. It also contains the default myenv.cmd that defines several variables values relevant in the build process, such as source depot configurations.

Table 9 - Code tree structure

This table resumes the code tree. In the localization process only the private directory, tools directory and some components of the public branch were used.

4.4.2 Source Depot

Source Depot is command line-based and provides file revision control, source control, file sharing, branching, merging, integration and labelling for groups of files. The provided code tree, mentioned in the previous sections, is a mapping of the VC code tree. A Source Depot Mapping is a selected set or subset of all the directories and subdirectories stored in the depot. A mapping is a correspondence of the directories stored on the Source Depot server with the directories created on the client. A mapping tells the server where to synchronize the files stored on the client.

Source Depot exhibits similar functions to Microsoft Team System [TeamSystemWeb] when considering source control. Nonetheless Team System, due to being more recent, has a much more user friendly interface (graphical instead of command prompt) and a larger set of features.

To use Source Depot one needs firstly to setup a Source Depot Environment, by copying the program file sd.exe to the VC path and setup the client. To setup the client it is needed to set the server and port number in the configuration file. Next, we map the client view, which requires some knowledge of how the project is organized and involves mapping the entire depot into the client workspace. To map a client view it is required to establish the folders in the depot and to which folder (in the client machine) they are mapped into. Finally, the code tree can be synchronized and basic commands from Source Depot, such as add, edit, delete and submit, can be used.

4.4.3 Build

Voice Command team uses the Build Utility to execute builds of the product. The Build Utility, build.exe, was created by Steve Wood to build Windows NT [BuildUtilWeb]. The Build Utility provides a structured way of building things and it is the default build process for Windows NT. The Build Utility has been successfully used by other groups, such as Exchange [ExchangeWeb], Media Foundation [MediaFoundationWeb], and MSN [MSNWeb].

The great benefit of using this Build Utility, rather than creating your own make or nmake files, is:

- The Build Utility isolates all target-dependent information in a centralized location (including header and library include paths, compiler names and switches).
- Products can be built for multiple hardware platforms without specifying any target-specific information.

The files used by the Build tool are:

- sources - lists the rules and macros needed to build a specific component subdirectory.
- dirs - contains macros called DIRS and OPTIONAL_DIRS that specify the directories to open recursively and the order in which they should be opened.

The tool looks into the dirs file looking for the macros DIRS and OPTIONAL_DIRS. It then accesses each listed directory where it might find another dirs file indicating more directories to access or a sources file indicating that

something needs to be built. In the source files it looks for the macros SOURCES, INCLUDES, TARGETNAME, and TARGETPATH. These are parsed to determine the dependencies, the list of files to build, and the end result. Depending on the options passed to build, it does the appropriate action and then calls a make program to build the component. There are files (makefile and makefile.inc) that the build will recognize in special circumstances (e.g., only on a clean build or only when the NTTARGETFILES flag is specified in the source file). When the entire directory tree is built, the build terminates.

The build tree is structured to allow Pocket PC and Smartphone binaries to be built in one pass (per processor, currently only ARMV4). Pocket PC specific binaries have a _PPC suffix and Smartphone specific binaries have a _SP suffix.

To perform a build it is required to define several environment variables. The developer must define _NTDEVELOPER - alias of the developer, INETROOT - path of the code tree and add to the system variable path the directory containing the source code and the path for scripts and executables (“tools/path1st”).

For each build it is advisable to clean the build environment using BldCleanPrepare.cmd script available at tools/build/environment. This batch file will clean the code tree of all changes that weren’t submitted in the branch and prepare the environment to perform a build. Next, we can call BldStart.cmd and the build specified in the setup variant builder configuration file (BuildSetupVariants.ini) will be created.

4.4.4 Setup variant builder

The setup variant builder – defined in BuildSetupVariants.ini - allows configuring the possible builds of Voice Command. This configuration file is interpreted by a pearl script - BuildSetupVariants.pl. Both files reside in the tools/build/setup directory.

BuildSetupVariants.ini allows controlling the generated builds. This file needed to be changed in order to support a EP build.

In order to describe this component the student has divided it into pseudo-sections as follows.

4.4.4.1 Main section

In the "main" section of the file we can define which variant we want to build. Among the available build types the ones used in the project were:

- **Ship** - Selected when the product is ready for shipment.
- **Debug** – Access to all kind of debug information during installation and runtime.
- **Trialship** – Trial version of the product ready for shipment.
- **Trialdbg** – Trial version of the product with access to debug information.

For each build type we can select the setup types which are divided in two groups: Pocket PC and Smartphone. For each kind of device we have setup types that allow updating the product from previous versions (e.g. PPC_v1_5_v1_6_updater), sign (or not) the CAB file and trial setups.

In the variant builder we also need to specify the products GUID for all different combinations build/setup of the product. To achieve that, it was verified which keys were common and which were unique in each of the several languages (note that this file is common for all languages). For the ones that were unique guidgen.exe (available for example in the Visual Studio package) was used to generate them, while the others were copied from the other languages.

4.4.4.2 Localization section

In the localization section the languages to be built are defined. Here the "PT-PT" symbol is added. For each language it is needed to:

- Define variables for language identification (e.g. language code).
- List all language specific files such as the ones that compose the TTS and SR modules.
- Define the respective skus (business identifier that defines the product).

4.5 European Portuguese Engines Integration

4.5.1 Speech Recognition engine

The used SR engine was developed by the Microsoft Speech Components Group (SGC) and its training and compilation is described in section 3.1. This model only supports command and control and has no dictation capabilities. At a physical level, the correspondent 2070 SR branch is created and the files described in table 6 (that

contain the acoustic models, senone map, etc), Yakima core (spsreng.dll) and SAPI dynamic link library are added.

Information on how to install the engine, such as registry settings, is provided and VC setup is configured to consider this directory, through makefile manipulation.

4.5.2 Text-to-speech engine

The EP TTS engine that was integrated in the code tree is an evaluation version from Nuance. The evaluation package brings several versions of the EP TTS for Windows CE.

The chosen version of the TTS to integrate in VC is Scansoft Madalena_DRI20_11kHz and occupies 76kB of static RAM and 7.67MB of ROM (the smallest in the package). The several modules of the TTS are divided into language independent (table 10) and dependent (table 11) components. The language dependent components differ from language to language.

Language independent components	ROM	RAM
RS Solo API, TTS engine and common resource modules	488 KB	44 KB
11 kHz synthesizer	108 KB	4 KB
SAPI5 interface layer	76 KB	12 KB

Table 10 - Size of the language independent modules

Language dependent components	ROM	RAM
Madalena voice	98 KB	4 KB
Speechbase for DRI20, 11 kHz	5.28 MB	---

Table 11 - Size of the language dependent modules

As in the SR engine integration, the respective TTS files and installation/registry information is added to the 2070 branch. Registry information regarding the TTS was obtained through introspection of the .inf files residing in the evaluation package supplied by Nuance.

4.5.2.1 Broker Mechanism

One relevant concept used in the TTS module is the broker mechanism that enables the discovery of all modules and data files at runtime. The broker collects information of all available components and loads those data files or modules at runtime. To run

the broker mechanism, the broker header files (e.g. sapi5_conv_ptp.hdr) must be under the mandatory directory "Speech". The developer can choose any directory for the installation but the "speech" directory is required under the installation directory. So, if we want to install the TTS under the directory, "\mydirectory", then we need to create the "speech" directory under "\mydirectory". The second mandatory directory is the product directory, "rssolo". This directory should be created under "speech" directory and all broker files must be placed here. This was taken into account in the installation script.

Each runtime module is linked to a broker header and this header file contains the information of the component, interface and its location. Every runtime component should have the broker file except some modules that are dynamically linked when the application starts, such as combrk.dll and comrsrc.dll.

There are also extra header files, which are not linked to any runtime component, (e.g. Realspeaksolo.hdr, Rssolo_ptp.hdr) that contains information about the product version and language (e.g. language code, manufacture, etc).

4.5.2.2 Modules to be linked to the Application module

Some of the runtime modules are linked to any broker file and must be dynamically linked to the application when the application starts. These files must be installed in the same directory as the application or the location, where those modules are installed and must be registered as an application path. The use of SAPI5 interface requires the additional link of some extra modules such as, rs_sapi5_solo.dll.

4.6 Features Localization

In the Features branch resides the source code related with the different features provided by VC such as Calendar, Email, Phone, Media, etc. Among the subdivisions one can find resource and grammar files that need to be localized.

The localization of a product such as VC requires that grammars and User Experience (UX) design to be adapted. This process can be divided in the following stages:

- Cultural Research
- Digit Dial Design
- Localization of Voice Commands and Voice Prompts

- Translation - Translate Help, Setup Strings, Readme, etc

4.6.1 Cultural Research

Cultural Research is the notion of understanding the locale specific issues relative to Voice Command. All localizable files (skins, help, etc) are affected by this type of issues. Considering VC nature, the following aspects relevant to the localization process can be identified:

- **Digit Dialing patterns and conventions** – The US English product currently support dialling 7 or 10 digit numbers. In some other places this will not be enough and need to be modified. In the EP product version there is support for the most common dial patterns with 9 and 12 digit numbers (see next section).
- **Number Confusion** – If there are numbers that are not distinctly recognized from each other when spoken.
- **Verb synonyms** – Identifying the best translation of the English verbs and the appropriate localized verbs and synonyms.
- **Identifying naming algorithms** – Checking what is the common way of placing a call. In other words, check if it is common for users to say "Call John Henry at Work" or "Make a phone call to John Henry at home" etc.
- **Identifying relevant names** – For the names, emphasis should be put on international popular artists and local specific artists as well as artist with abbreviated, numeric, intentionally misspelled, or special characters in the names (e.g. Blink-182, T.A.T.U).
- **Identifying Calendar titles** – Creating a list of calendar/appointment names with emphasis on abbreviations and local specific behaviour (e.g. Go to Dr. appt. ASAP).

4.6.2 Digit Dial Design

Another important issue is the Digit Dial Design. It is needed to determine the proper support for dialling numbers in a localized product. This issue concerns with the resource and grammar phone feature files. A good digit dial must consider:

- How many are the minimum numbers of digits that can be dialled.
- How many are the maximum numbers of digits to be dialled.
- How do area codes affect number dialling.
- How does calling a neighbouring country affect dialling patterns.

- What special characters are spoken when dialling (e.g. +, =, *, #).
- What special numbers are spoken (1-800, One Eight Hundred or in the EP case 112, “Cento e Doze”).

For the localization to EP, a minimum number of 4 digits and a maximum of 14 are considered (or 12 digit number previewed with a plus “+” optional symbol). The 12 digit number is composed by the country code, area code and the respective phone numbers. Special numbers such as the emergency number (112) and the information number (118) were also taken into account.

4.6.3 Localization of Voice Commands, Voice Prompts and GUI

Once the proper cultural research has been performed, the various files that create the Voice Command UI can be localized. The resource components contain two types of items that need to be localized: Voice Prompts and GUI Labels/Text. Most of the Voice Prompts are defined to receive parameters only known at runtime and the identifier does not always allow understanding the scenario where the prompt is used. We need to place the prompt in the application flow, based on the source code, so that the localization result doesn’t come out of context. The figure below is a simple example of an ambiguous voice prompt, where the context where it is used must be known, as well as its parameters.

```
IDS_CONTACT_THIS_CONTACT                "%1 o contacto, %2 ? "
```

Figure 29 - Excerpt of a resource file containing voice prompts

The most important part of this entire procedure is the grammar localization. The grammars will determine the behaviour of the application interface and need to carefully build. Voice Command uses SAPI grammars, but they are encapsulated in a “waml” format. A “.waml” file is basically a XML file that besides encapsulating the grammar it also contains relevant information associated with the skin. For example, it can contain configuration information about help prompts, if interruptions are allowed or application constants.

These files are compiled using the Windows Automotive Binary Skin Compiler (Wabsc) that generates a “.wabs” file. The wabsc compiles the skin files but does not report structural grammar errors. This fact may lead the developer to believe that the grammar is valid. This kind of errors will only appear during runtime when the grammar is loaded. The error message that reaches the user/tester is quite ambiguous

and leads to wrong assumptions of exactly where the error occurred. To avoid this issue a grammar compiler was used over the altered grammar (that is inside the skin file) to verify its integrity.

4.6.4 Translation – Translate Help, Setup Strings, Readme, etc.

Non-functional strings need to be translated into the target language for the localization process to be complete. The translation process is applied to help files, user guide and installation strings. Some of the files that require translation are very similar, for example a user guide for Pocket PC and a user guide for Smartphone have several common phrases and words. To take advantage of these similarities, all these files are generated based on a single HTML [HTMLWeb] file, except desktop installation strings.

The above files contain strings that are not related to the speech user experience and can be translated as written. A review of these translations will need to occur later, in a joint collaboration between a linguistic and the localization engineer.

A style guide that describes the formatting and indenting rules for VC documentation must also be followed.

4.7 Product Setup

The setup branch is divided in several sections:

- **Setup library** - source code for setup procedures
- **Static files** – files copied to VC directory during installation
- **Installation Shield (IS)** – tool used to produce the final setup file.
- **CAB** – installation package for mobile device
- **Tools** – tools used in this branch for creating cabs, signing cabs, certificates, etc.

The Setup library contains all the code referent to the installation in the device. To localize the Setup library we need to modify it in order to access the EP (2070) components and also localize all associated resources for Pocket PC and Smartphone. The resources contain all the possible messages that can be outputted during the product installation (in the device).

To build the product and generate a setup file, it is necessary to have IS installed in the desktop computer where the compilation is made. InstallShield 8 [InstallShield] was used. In the project referent to the desktop installation, information about SAPI, SR and TTS registry was added. Here, the strings used during the VC installation process were also defined.

Cabinet or CAB files are installation files for Pocket PC and Smartphone devices that are going to be copied and installed in the target devices. CAB is the Microsoft Windows native compressed archive format. It supports compression and digital signing, and it is used in a variety of Microsoft installation engines. To generate a "cab" file the cabwiz.exe tool is used. Cabwiz receives an installation file (.inf) file as input and outputs a ".cab" file. The figure bellow illustrates the creation of a cab file for Pocket PC.

```
cabwiz "Microsoft Voice Command PT PPC 1.60 for M2M.inf" /cpu PPC420_StrongARM
```

Figure 30 - Using cabwiz tool to generate a Pocket PC CAB file

The INF file provides installation instructions and it is where all the steps and information about the installation in the device resides. The INF file was based on [INFFileWeb], [INFFileWeba] and Voice Command installations results in French, German and English.

This file architecture is divided into several sections, which obey to a determined format where the application name, version, destination directories, shortcuts, install directories, platform, processor, files and all registry information (e.g. TTS and SR installation, SAPI registration and Voice Command definitions) is defined.

Visual Studio is an alternative to generate installation packages for mobile devices but has some limitations in terms of features and does not allow configuring all the desired parameters required in the VC EP CAB file.

In the Destination Directories section, the directories that will contain the files of the application in the device were established. For the Smartphone platform there is the need to place shortcuts (to start and configure the application) in the start menu due to the inexistence of a settings menu, like in a Pocket PC. This will provide an easier user access to the application executable or shortcut, considering the impossibility or difficulty in defining a PTT button on some devices.

The application total size is 10.520Kb, mostly due to the TTS size which is extremely large to ship in this kind of application. Based on the TTS modules from the other languages, an acceptable size for the TTS engine would be about 3MB.

4.8 VC Localization Issues

4.8.1 Cosmetic bugs

Mobile devices screen can come in several shapes and sizes depending on the model and manufacturer. When localizing GUI sentences or words, the student tried to find the most suitable translation for the EP users. However, this may lead to cosmetic bugs. This term is used when the resource localization results in hidden sentences due to screen or GUI components size, such as dialog boxes and buttons.

Figure 31 and fig. 32 are examples of cosmetic bugs resulting from the localization process. In the figure in the right we can see that the options label is incomplete, missing the word “voz” (voice). In the figure in the left, the dialog box of the third sentence is too small.

Figure 31 - Incomplete options label

Figure 32 - Imperceptible option in definitions menu

There are two solutions for this kind of issues: rewriting the phrases or change components size. The second is not always possible, due to screen limitations.

Even being aware of this issue it isn’t always visible or perceptible in a resource file, which contains GUI components size, which sentences will fit properly or if we can change the component size without changing the product. We must also take into

account that usually this work is performed by a third party (linguist) and this entity only has access to the sentences, not to the full resource file.

In the Voice Command localization to EP the translation was performed by the localization engineer, with the advice of a linguist on obtaining shorter sentences with the same meaning as the result of the direct translation.

4.8.2 Debug

During the localization process there were several bugs/errors that needed to be solved. Without access to a proper debug environment it becomes difficult to identify and solve upcoming problems. To overcome this issue three possible forms of debugging were identified and studied:

1. Change the application source code and write an output message to the repository.
2. Use platform builder kernel debugger.
3. Use CoreXT build system to produce debug binaries and use Visual Studio debug.

Writing a message to standard output or to a repository (file) it's maybe the most primitive way, but effective, in some cases, of debugging. However, due to, source code size, compilation time (a full compilation takes between 2 to 3 hours depending on the host machine) and facing several multithread scenarios made this form of bug tracking difficult.

Another option to debug the application was to use Microsoft Platform Builder for Windows CE. This is an integrated development environment (IDE) for building customized embedded operating system designs based on the Microsoft Windows CE operating system. Platform Builder comes with all development tools necessary to design, create, build, test, and debug a Windows CE–based operating system design. The IDE also provides a single integrated workspace that can work on both OS designs and projects. To use Platform Builder, a pre-compiled system image with VC installed is required. The image is then downloaded to the device after the configuration of the remote services. However this solution has some drawbacks: the main drawback is obtaining a pre-compiled system image of Windows Mobile, which was not available during the work of this project. Second, using Platform Builder is

not a trivial task, so the developer would need to learn how to use it (which obviously takes time). Finally, using a Kernel Debugger to resolve a localization issue takes the debugging into an even lower level, which involves major additional work.

The chosen solution to solve the localization errors was to create a debug build of the application, in order to obtain debug binaries, and use Visual Studio 2005 Debugger to run the application and have access to debug messages, such as module loading/unloading. This kind of procedure is also used when in the migration of solutions from Embedded Visual C++ to Visual Studio 2005 and can be described in the following steps:

1. Create a Debug Build of the Application
    a. Create a Smart Device Visual C++ STANDARDSDK_500 [StandardSDK] Console Application dummy project using Visual Studio 2005
    b. Remove everything from Selected SDKs list and move STANDARDSDK_ 500 from Installed SDKs to Selected SDKs.
2. Connect to Windows CE Device (development machine)
    a. Setup Project properties of the Dummy project to launch the application
    b. Configure binary files location, source directory, remote executable, etc
3. Specify symbol path
    a. Configure Visual Studio to search a specific location for Symbol files (.pdb)
    b. Ready for debugging.

This technique allowed solving the problems resulting from the application localization, using a known development environment. It allowed the student to understand in which modules or procedures the error was and consequently isolate it and solve it. For example, the Locale Handler library is different for each language and this was hardcoded in source code. This form of debugging allowed to understand that EP was not amongst the contemplated languages in the application source code.

4.9 Basic Verification Tests

To verify the build stability, a Basic Verification Test (BVT) of Voice Command was performed. BVT requires the execution of a combination or automated and manual tests that will cover from the setup to the removal of Voice Command.

The BVT tests have several objectives:

- Prove that the build is stable by executing the BVT;
- Verify that Voice Command is installed correctly;
- Verify that Voice Command can be removed from the device without errors;
- Verify interoperability with other applications within the device's operating system.

The verified features are presented in table 12

#	Feature	Area
1	Verify Setup of Voice Command	Setup
2	Verify Removal of Voice Command	Setup
3	Verify Main App	Main App
4	Verify Dial	Phone
5	Verify Call	Phone
6	Verify Show Contact	Contacts
7	Verify Calendar	Calendar
8	Verify Launcher	Launcher
9	Verify Media	Media

Table 12 - Features list of the BVT

To perform the BVT tests a HTC P3600 Pocket PC device with Magneto (Windows Mobile 5.0) was used. This Pocket PC has the specifications described in table 13.

Specifications	
Processor	Samsung® SC32442A 400 MHz
Platform	Microsoft® Windows Mobile® 5.0
Memory	ROM: 128 MB RAM: 64 MB SDRAM
Cellular Radio Module	Tri-band HSPDA/UMTS: 850, 1900, 2100 Quad-band GSM/GPRS/EDGE: 850, 900, 1800, 1900
Connectivity	Infrared IrDA SIR Bluetooth® 2.0 Wi-Fi®: IEEE 802.11 b/g HTC ExtUSB™ (11-pin mini-USB and audio jack in one)
Audio	Built-in dual (microphone and speaker) Headphone: AMR/AAC/WAV/WMA/MP3 codec
Expansion Slot	miniSD™ memory card

Table 13 - Test Pocket PC characteristics

To complete the testing activities a localized list of contact cards, calendar appointments, digit dial examples, program names and media (stock data) were set in the device. These are described on table 14.

Stock Data	Amount
Contacts	325
Song names	57
Appointments	25

Table 14 - Stock data information

The table bellow describes the performed verification tests.

Pre-requisites	Action	Say…	TTS	Say…	Expected	Pass /Fail
Active Sync	Install Voice Command				Start >>Program >> File Explorer>> Program Files -> Voice Command -> check on VC files, Speech Engine Files and Text To Speech Files.	Pass
Active Sync	Install Voice Command				Help file displays at the end of the installation (desktop and device)	Pass
Active Sync	Install Voice Command				Help file displays on device when search. Start >>Help>> Voice Command.	Pass
Install Voice Command	Remove Voice Command				Voice Command Files are removed from \Program Files\Microsoft Voice Command	Pass
Install Voice Command	Remove Voice Command				Voice Command Directory is Removed	Pass
Install Voice Command	Remove Voice Command				Voice Command Button	Pass

					Reverts to default and no longer is mapped to voicecmd.exe	
Install Voice Command	Remove Voice Command				Help Files are removed	Pass
Install Voice Command	Remove Voice Command from Desktop. Control Panel>>Add or Remove Program				Voice Command Files are removed from the Add or Remove Program	Pass
Install Voice Command. Make sure device is still syncing with desktop	Remove Voice Command				Voice Command Files are removed on device	Pass
Install Voice Command	Go to the Device Settings\Personal				Voice Command Control Panel Applet displayed in the Personal Setting Tab	Pass
Install Voice Command	Go to the Device Settings\Personal and click on the Voice Command Icon				That the Voice Command applet opens up after a single click	Pass
Install Voice Command	Go to the Device Settings\Personal and click on the Voice Command Icon				Voice Command Applet displays appropriate extensions (Calendar, Contacts, Media, Phone, Start Menu)	Pass
Install Voice Command	Press the Push to Talk Button				Voice Command Button is properly assigned	Pass
Install Voice Command	PTT	Dial <Minimum Length Number Supported>	Dial <Number> Correct?	Cancel	Voice Command will Cancel	Pass

Install Voice Command	PTT	Dial <Maximum Length Number Supported>	Dial <Number> Correct?	Cancel	Voice Command will Cancel	Pass
Create at least 100 Contacts (Last,Middle,First with home, home2, work, work2, radio, mobile, assistant)	PTT	Call <Contact>	Call <Contact> Correct?	Cancel	Voice Command Respond with cancel tone	Pass
Contact with Duplicate Entries (Last,Middle,First with home, home2, work, work2, radio, mobile, assistant)	PTT	Call <Contact>	This <Contact> Correct?	Next	Voice Command will go to the next duplicate contact	Pass
Contact (Last,Middle,First with home, home2, work, work2, radio, mobile, assistant)	PTT	Show <Contact>			Voice Command Displays the correct contact card	Pass
Contact (Company name only)	PTT	Show <Contact>			Voice Command Displays the correct contact card	Pass
Contact with Duplicate Entries (Last,Middle,First with home, home2, work, work2, radio, mobile, assistant)	PTT	Show <Contact>	This <Contact> Correct?	Next	Voice Command will go to the next duplicate contact. Contact cards are displayed.	Pass
Create several appointments for today and tomorrow	PTT	What's my next appointment ?			VC will TTS the next appointment	Pass
Create several appointments for today and tomorrow	PTT	What's my schedule for today?			VC will TTS today's appointments	Pass
Create several appointments for today and tomorrow	PTT	What's my schedule for tomorrow?			VC will TTS tomorrow's appointments	Pass
Copy Media Songs to the Storage card	PTT	Play Everything	"Everything"		Voice Command will open Media Player	Pass

					and will begin playing all of your songs	
Copy Media Songs to the Storage card	PTT	Play Artist	"Which Artist?"	Cancel	Voice Command will list the artist name in your device.	Pass
Copy Media Songs to the Storage card and Play any song	PTT	Next Track	"Next Track"		Voice Command will go to the next track in your play list	Pass
Copy Media Songs to the Storage card and Play any song	PTT	Stop	"Stop"		Voice Command stop playing music	Pass
Copy Media Songs to the Storage card and Play any song	PTT	What song is this?	…		Voice Command tell the song information	Pass
Copy Media Songs to the Storage card and Play any song	PTT	Play Everything	"Everything"		Verify that the Voice Command Skin displays (Only for Media Player 8.0)	Pass

Table 15 - BVT tests

The build has passed all verification tests, so it can be concluded that is stable and is working as expected.

This build was also tested with success in the following devices: HTC S710, HTC Touch, i-mate JAQ4, Samsung i600, GSmart i600 and HTC S620.

4.10 Usability evaluation

To determine the usability of the system, an evaluation methodology was adopted and a usability test was developed to assess the usefulness of the EP version of Voice Command in comparison with the mobile device GUI provided by Windows Mobile. The device used was the same HTC P3600 that was used in the previous tests and it is described in table 13. The tests were conducted in home and office environments. In the home environment the tests took place under a common living room scenario (television on, though relatively quiet). In the office environment the tests took place in a room with an average 4 to 5 people talking/working.

4.10.1 Objective

This usability evaluation was conducted with the following objectives:

- To validate the level of acceptance of speech recognition technologies combining the local language and mobility.
- Improve VC VUI based on the analysis of the most common recognition errors and the feedback supplied by the subjects.
- Detect and identify which are the "local" needs in this area.
- Allow the users to really have a "Hands on Lab" experience and hear their opinions.
- Advertise the localized product and influence its adoption.

4.10.2 Usability evaluation methodology

The usability testing experiment was designed to assess the usefulness of VUI interaction compared to traditional GUI interaction. For this, the time taken and efficiency of the subjects when accomplishing two groups of tasks commonly available in a mobile device (Smartphone or Pocket PC) is evaluated. The tasks are described in table 16 and the script is available in Appendix B, page 8.

Nowadays, there are numerous actions and services provided by a Pocket PC or a Smartphone; so we selected a first group of tasks that are the primary reason or essential when using a device and consequently more common amongst users. A second group tasks was also selected and represents actions that are not so popular, but depending on the user they can be more or less used. This second group was harder to select. The student based the selection of this second group on the Original Equipment Manufacturers (OEMs) preference (the features that they advertisement) and the tasks were designed with the intent of being the most global and representative of a group of services.

Common tasks	Other tasks
1. Placing a phone call	5. Lookup contact information
2. Checking calendar	6. Open an application
3. Verifying device status	7. Play media
4. Change profile or sound options	8. Manipulate media

Table 16 - Tasks performed in the usability evaluation test.

The tasks are described in detail as follows:

- **Task number 1** – Placing a phone call – The user is required to place a phone call to the mobile phone of the contact named "João Freitas". In the contact list there were 300 contacts with several (5 to be more precise) "João" in the list, but only one "João Freitas". There were no repeated contacts.

- **Task number 2** – Checking the calendar –The subject is required to understand what commitments are scheduled for the next day. In the device there are several appointments registered for the current and two commitments for the next day.

- **Task number 3** – Verify the device status - In this task the subject should obtain the battery percentage level.

- **Task number 4** – Change profile or sound options – In this task the subject is asked to reduce the volume to the minimum or enable the device's silent mode.

- **Task number 5** – Lookup contact information – The subject is asked to consult the available numbers of the contact named "João Freitas". This task may be similar to the first task when using the graphical interface, depending on the user methodology to perform the tasks.

- **Task number 6** – Use of an application installed in the device – The subject is asked to open Internet Explorer.

- **Task number 7** – Play media –The subject is asked to play a set of songs of a local artist, e.g. "Ornatos Violeta". All the multimedia is located in a memory card and it is shown to the subject the audio location and how to access it.

- **Task number 8** – Manipulate media – The subject is required to move on to the next song. The device should be set in a state that allows the execution of this task.

These sets of tasks are performed using both interfaces alternately (e.g. the subject performs the tasks with the VUI and then performs the same tasks using the GUI; the next subject first performs the tasks with GUI and then performs the same tasks with the VUI). By alternating the interface order independence is gained, breaking any kind of relation between the interfaces that may affect test results.

The tests, regarding each interface, are preceded by a four minutes session for interface adaptation. The adaptation is divided in a two minute explanation and followed by a two minute training session, so that the subject is able toadapt himself/herself to the device. Each subject also receives a two page user guide and a half page quick user guide, about each interface, explaining how to use the device. The tutorial about the user interfaces include the tasks mentioned in table 16. The tutorials are as clear and equivalent as possible and shown in Appendix B, page 2-7.

Each testing session took about 15-25 minutes, depending on the user feedback.

4.10.3 Subject profiles

The usability experiment was run on 35 unpaid users with ages between 23 and 65. There were 26 male and 9 female subjects with a variety of business occupations (IT engineers, Senior Consultants, Retired, Students, etc). In order to obtain a reference profile the subjects were asked to fill a questionnaire about their previous experience on speech interfaces, use of stylus (on mobile devices) and mobile devices in general. The results reveal that the majority of the subjects had low experience with speech interfaces and in a scale of 0 to 3 (0 – None, 1 – Low, 2 – Medium, 3 - High) presented an 1, 2 average value; a medium experience in the use of the stylus, with an 1,8 average on the same scale and high experience with mobile devices, presenting an 2,7 average (same scale of 0 to 3). These results are presented on the graphic of fig. 33 below and were gathered through a questionnaire filled before executing the tasks. This questionnaire can be found in Appendix B.

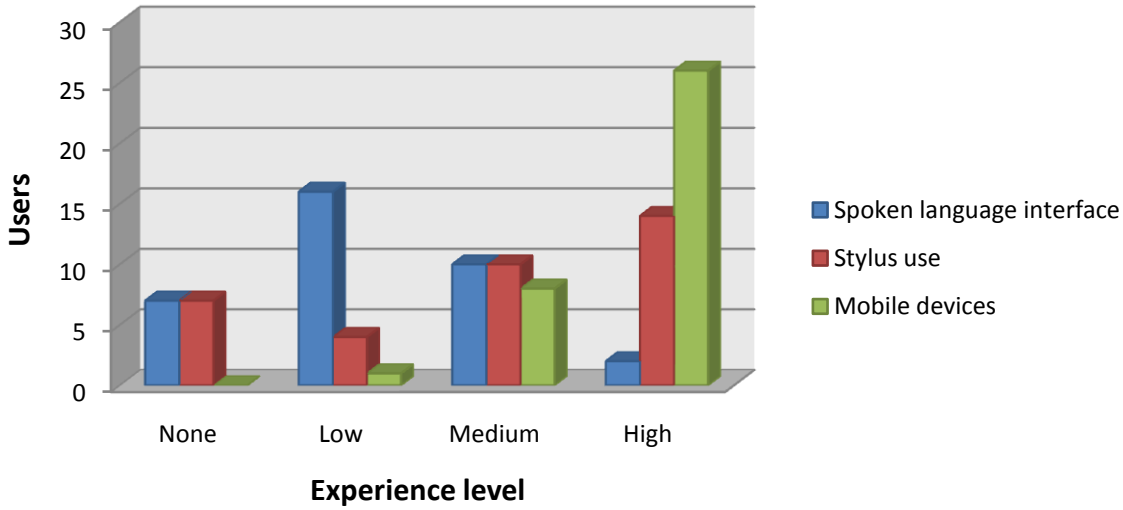

Figure 33 - User experience profile

4.10.4 Evaluation results and analysis

All the subjects were successful in completing the experiments with both interfaces, showing different degrees of efficiency and taking different durations to fulfil the tasks. During the experiments the student has became aware of some problems that rose while using the system and received many comments and suggestions from the test subjects which will be described in the following section of this document. An analysis was also performed on:

- The time that the subjects spent to perform the experiment.
- The number of attempts when performing a task.
- Why subjects failed to use the VUI.
- Questionnaires filled after the usability test.

When analysing the times taken in the several tasks there is a clear advantage of the VUI placing a call to a determined contact, checking a contact card and playing a group of songs (fig. 34). VUI's advantage in such tasks is explained by the task complexity. All of these tasks have in common the fact that they require a set of actions, such as menu navigation, scrolling, etc, to be executed with success. The VUI gives a way of accessing all of these services through a single action, as simple as issuing a voice command. The inverse situation can be noticed when checking tomorrow's appointments or passing to the next song. To execute these tasks both interfaces only need a single action, so it is natural to have a balance in terms of time

when executing them. Despite the time a task takes to accomplish we will see that the subject not always prefer the fastest way to perform the task.

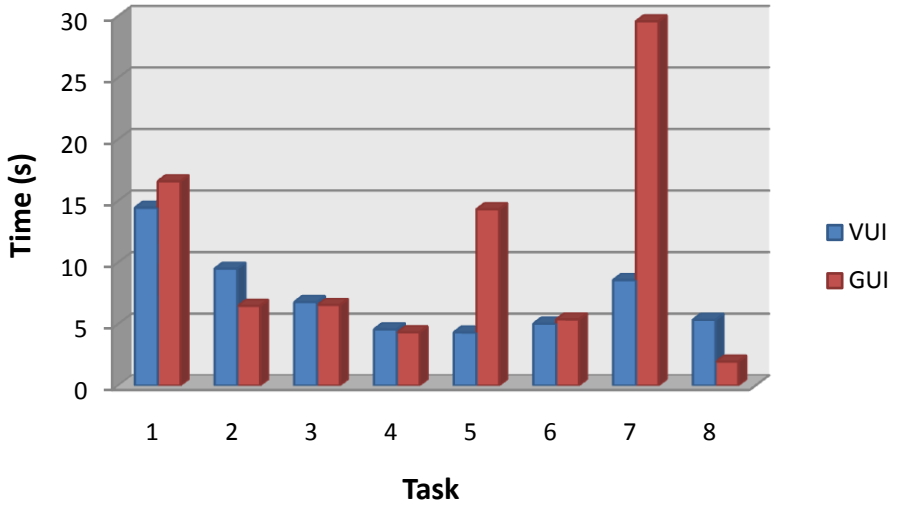

Figure 34 - Time taken in accomplishing the tasks (VUI and GUI)

When using the VUI it is normal to occur recognition errors, especially considering that the majority of the subjects present a low experience level with speech interfaces. The main reasons for recognition errors in these tests were:

1. The spoken command is not included in the grammar.
2. The command sentence was spoken in an incorrect order (e.g. in the grammar is available the command sentence "Que horas são?" and the user says "São que horas?").
3. Covering the microphone, with the hand that holds the device, when entering the voice command.
4. Bad interaction with the PTT button, such as saying the command without pushing the button.
5. Taking more time than usual to input the command (after pressing the PTT button).
6. Push the PTT button and not wait for the noise or visual signal from the application to input the voice command.
7. Talking before introducing the voice command (when the application is waiting for a command).

In the graphic analysis of fig. 35 and fig. 36 the number of attempts taken to accomplish a task, are presented. Notice that the number of attempts decreases during

the test execution, except in task number five. The decrease in the number of attempts can be justified by the quick adaptation of the subjects to the VUI, since the tasks where independent. This was also visible during the tests. The exception verified in task number five is due to the lack of synonyms in the application grammar to perform this particular action.

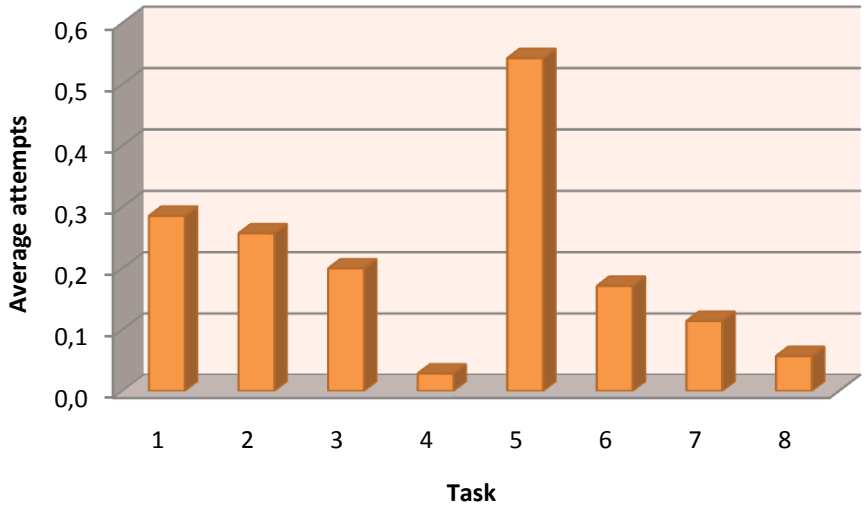

Figure 35 - Average number of attempts for each task

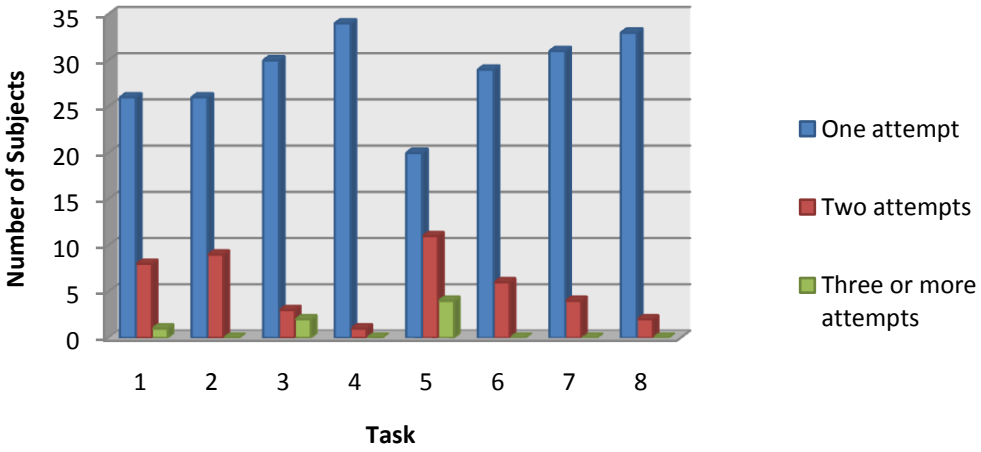

Figure 36 - Number of attempts for each task

4.10.5 Subject comments

The subjects provided valuable feedback during and after the experiment, mostly regarding the VUI interface. Right after the testing session, the subjects were asked to fill a short questionnaire (Appendix B, page 9) that allowed to extract opinions and preferences of the subjects, when asked to compare both interfaces and about the VUI.

From section 4.10.3 one can conclude that the subjects are more familiar with the GUI, as opposed to VUI. Ideally the subjects had equal familiarity with both modalities so that we could make a more precise comparison based on the users. In spite of that, there are still some interesting conclusions that one can extract.

Figure 37 shows the questionnaire results. When analyzing the information provided by the users a clear user preference for the VUI in accomplishing the tasks is observed, despite the time taken to accomplish the task. The tasks one and seven reveal that interaction via VUI is overwhelming, when compared to the GUI.. This is justified by the fact that, when using the GUI, inherent actions such as finding a contact to place a call, or having to go to the music repository to play a group of songs requires additional attention, time, visual interaction and manual interaction with the device. On the other hand, the VUI only requires a single voice command to index a contact number or an album artist.

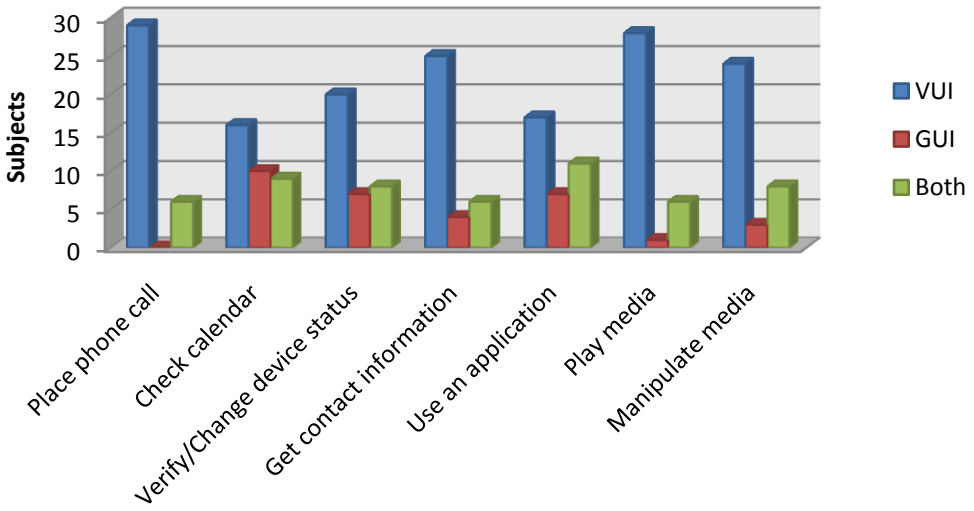

Figure 37 - Questionnaire results

In the final question the subjects were asked to present ideas for new functionalities that are, or not, already available on their devices with the GUI and would like to see implemented with a VUI. The most demanded functionalities involve dictation and interaction with external servers, more precisely:

- Dictating email or SMS.
- Email triage - archive, delete, reply, etc.
- Dictating appointments – introduce new appointments in the agenda through speech.

- Editing agenda – move, delete appointments.
- Defining alarms.
- The possibility of defining macros - defining a voice command that performs a set of actions.
- Interaction with Exchange server, GPS or other web servers.
- Active Sync with speech

During the testing period the subjects gave their opinion on the application, presenting a set of suggestions that include having a more stronger component of visual feedback when stimulated with a voice command, e.g. when asked for available albums the application gives a list of the available albums in the device (on which we can navigate with speech). This could be implemented has an option.

To conclude the analysis about the subject's feedback, 88,6% of the subjects prefer the VUI interface when asked to choose between the two interfaces, and only 11,4% prefer the GUI interface. Nonetheless, 21% of the users mentioned that his option on which interface to use depends on the situation and location. These subjects commonly choose the GUI interface for public places such as shopping malls, and the VUI for "busy hands, busy eyes" scenarios such as, automotive and sports.

4.11 Result analysis

This chapter described all the proceedings and techniques used in the localization of VC for EP. This includes code tree analysis, features localization, resources and grammar localization, engines (SR and TTS) integration, installation procedures, testing the product and a usability evaluation study.

Voice Command uses a set of MS internal tools for developing, source control and builds generation, such as CoreXT, Source Depot, and the Build Utility. These were unknown for the student and consequently an obstacle that was overcame. This also applies to the application architecture, which due to the lack of accessible documentation, had to be deduced from the complex code tree. Once the localization targets were identified, the adaptation of the application to the EP culture and language took place at every level. Along the localization process it became clear that the application wasn't predicted to exist in EP. The localization process had a strong

reverse engineering component with basis on existing versions of the application, such as French and German.

In order to validate the conducted work basic verification tests were executed. The BVT tests allowed verifying that the build was correct in every aspect that goes from the installation in the desktop to the removal of the device.

An usability evaluation study was also conducted, which allowed testing the build, divulgate the product, understand some of the reasons why speech recognition fails, perform a comparison between the GUI and the VUI, this time including several other actions/tasks, verify the usefulness of the VC in EP and improving the product final beta version. The conclusions from that study show that complex tasks, such as playing a group of songs, are easier to accomplish with a VUI. In scenarios such as sports or automotive, users tend to prefer the VUI. A quick adaptation by the users to the spoken language interface, during the conducted tests, was also verified, resulting in lesser attempts to accomplish a task. This study also allowed the student to improve the grammars, by realizing what were the most common verbs to describe an action, and consequently improving the product command and control interface for EP users. At the end of the tests the subjects were asked to give ideas about features and improvements which could be implemented, in order to improve user experience with mobile devices. The most demanded features were related to dictation, external server's interaction and more visual feedback.

A specific Voice Command corpora acquisition was performed during the usability evaluation study, recording all the commands issued by the several subjects. However, no transcription/annotation or any kind of work over this data was performed until the current date.

Currently, we have 10 persons constantly using the product located for EP, providing a constant feedback. None of these users required using a manual to identify the voice commands and the build is working flawlessly. From the given feedback extremely positive reactions to the product were obtained, such as "This is much better than the English version".

As future work, the localized resources should be reviewed by a linguist expert in parallel with a localization engineer (due to strings contextualization) and approved by the product's development team. A bigger set of BVT tests should also be considered with a larger amount of stock data.

Chapter 5

Conclusions and Future work

This chapter will be dedicated to an analysis of the proposed project goals and what has been actually achieved. The student also documents the intentions for future work and possible directions of research in the area of this project.

5.1 Summary of accomplishments

As it was described in the introduction chapter, the goal stated for this project is the localization for VC to EP, preceded by the development of a spoken language interface in a mobility context analysis.

In order to achieve these objectives, a classical methodology was followed. The student started by analyzing existent spoken language technologies, focusing on speech applications for mobile devices, similar to Voice Command, identifying methodologies and best practises. In this chapter, it was also described the used spoken language system architecture, the used ASR system, based on a pattern recognition approach, more concretely, HMMs and the common architecture of a TTS system. Voice Command features are also described, as well as other similar applications in order to understand the inherent advantages and weaknesses of the application, when compared with the current state of the art. We conclude that Voice Command is significantly larger, in terms of memory, in comparison with other existent commercial applications, but also presents a higher set of embedded features. The analysis of these applications also shows how spoken language interfaces can be applied in mobile devices.

The third chapter focus on the development of spoken language applications for mobile devices. It starts by analysing the localization of the SR engine, an essential component for the speech applications used in this work. To accomplish this task internal MS tool - Autotrain – was used. This tools provides a recipe with all the required steps to generate a fully compiled EP SR engine. In this chapter an analysis

in the framework of a spoken language interface for a mobile device was also presented. In this analysis a set of optimizations to improve speech recognition accuracy and user interaction experience was shown. To support this study, an example application that places a phone call to a contact, through a spoken language interface, was developed. The application interacts with the developed SR engine through SAPI and implements a name match algorithm to improve user experience. This represents a practical exercise that precedes the localization of Voice Command. The development of this demonstrative application implied learning basic concepts, such as the integration of the SR and TTS modules, which are also applied in the localization process.

Results of conducted usability tests, based on user's experiment, to, were also presented. These experiments showed that spoken language interfaces can provide easier and more efficient use of the device when performing a task, like placing a phone call, especially with users less experienced on handling mobile devices.

The fourth chapter describes the VC localization process to EP, which includes exposing the application's architecture, application adaptation to EP, the SR engine integration and a testing stage, posterior to the localization. Here, the encountered problems during localization as well as their respective solutions were presented. Some of the key problems found were unknown MS internal development procedures, application setup localization and features localization. These problems motivated a longer analysis of the code tree and techniques involved in the product build. This was overcome through reverse engineering techniques, based on other VC versions. To confirm the built stability the product is currently in beta testing and was subjected to several a set of generic BVTs. The conducted usability evaluation study over VC allowed the improvement of the application interface (grammar tuning) for EP, providing a more natural VUI, and understand why and where speech recognition fails. This evaluation also shown that complex tasks, such as playing a group of songs, are easier to accomplish with a VUI and that users prefer the VUI, especially in "busy hands, busy eyes" scenarios. Users also present a quick adaptation to the VUI based on number of attempts to accomplish a common task.

5.2 Future work

We believe that there are several possible future directions of research raised by the work presented in this project.

Voice Command development is currently stopped, however there are several aspects which could be improved such as, application personalization, integration of a dictation module, interaction with Exchange server, etc.

Relatively to the built version of VC in EP and in order to ship the product in EP we need to integrate our own TTS, due to the license price and size of the current integrated TTS. It is also scheduled to rearrange the grammar with the commands that lead the subjects to fail, so that users in general find the interface natural and familiar not needing to memorize voice commands. The objective is that the first two appointed reasons (section 4.10.4) for the applications to fail become absolute. This leads to the need of evaluating what is the impact of increasing command synonyms, in recognition accuracy, so that it stays acceptable. User experience will benefit with the addition of options, in terms of voice commands, however recognition accuracy will decrease due to the increased number of grammar items. The BVTs should be performed with larger amount of stock data and specifically to each feature, exploring every possible situation. There is also the need of a linguist correction of the current translation and a product approval of the VC Team.

For last, but not least, the student desires to accomplish a vision shared with the Voice Command development team:

"*Change the way people interact with and think about Mobile Devices*"

REFERENCES

[Acero06] A. Acero, "Building Voice User Interfaces", in MSDN Magazine. Feb. 2006.

[Amaral99] R. Amaral, P. Carvalho, D. Caseiro, I. Trancoso, L. Oliveira, "Anotação fonética automática de corpora de fala transcritos ortograficamente", PROPOR'99 - IV Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada, Evora, Portugal, September 1999.

[Barros06] M. Barros, C. J. Weiss, "Maximum Entropy motivated Grapheme-to-Phoneme, Stress and Syllable Boundary Prediction for Portuguese Text-to-Speech", Technologia del Habla, November 2006.

[Black07] A. W. Black, K. A. Lenzo, "Building Synthetic Voices", Language Technologies Institute, Carnagie Mellon University, January 21, 2007

[Blizzard] The Blizzard Challenge, http://festvox.org/blizzard/index.html, last visited on 15-09-2007

[Bonardo05] D. Bonardo, P. Baggia, "SSML 1.0: an XML-based language to improve TTS rendering", January 19, 2005.

[Braga03] D. Braga, D. Freitas, H. Ferreira, "Processamento Linguístico Aplicado à Síntese da Fala". 3º Congresso Luso-Moçambicano de Engenharia, Maputo, Mozambique. August 2003.

[Braga07a] Private conversation held with Daniela Braga about dialectal regions.

[BuildUtilWeb] MSDN, Build Utility Reference, http://msdn2.microsoft.com/en-us/library/ms792380.aspx, last visited on 15-09-2007

[CIAWeb] Central Intelligence Agency World Factbook 2007, https://www.cia.gov/library/publications/the-world-factbook , last visited on 15-09-2007

[COMWrap] MSDN, COM Wrappers, http://msdn2.microsoft.com/en-us/library/5dxz80y2.aspx, last visited on 15-09-2007

[CyberonWeb] Cyberon Corporation, http://www.cyberon.com.tw/, last visited on 15-09-2007

[ELRA] European Language Resources Association, http://catalog.elra.info/, last visited on 15-09-2007

[ExchangeWeb] Microsoft Exchange Server, http://www.microsoft.com/portugal/exchange/default.mspx, last visited on 15-09-2007

[Freitas07] J. Freitas, M. J. Barros, A. Calado, M. S. Dias , "Spoken Language Interface for Mobile Devices", in the Proceedings of 3rd Language & Technology Conference, Poland, October, 2007

[GovWeb] Portal do Governo, http://www.portugal.gov.pt/Portal/PT/Portugal, last visited on 15-09-2007

[HTMLWeb] W3C, HTML Working Group, http://www.w3.org/html/wg/, last visited on 15-09-2007

[Huang01] X. Huang, A. Acero and H. Hon, *Spoken Language Processing: a guide to theory, algorithm, and system development*, Prentice Hall, 2001.

[INFFileWeb] MSDN, About INF File Architecture, http://msdn2.microsoft.com/en-us/library/aa741215.aspx, last visited on 15-09-2007

[INFFileWeba] MSDN, Using INF Files, http://msdn2.microsoft.com/en-us/library/aa741213.aspx, last visited on 15-09-2007

[InstallShield] Macrovision, InstallShield, http://msdn2.microsoft.com/en-us/library/ms696274.aspx, last visited on 15-09-2007

[IPAWeb] The Internacional Phonetic Association, http://www2.arts.gla.ac.uk/IPA/ipa.html, last visited on 15-09-2007

[Jansche01] M. Jansche, "Re-Engineering Letter-to-Sound Rules", NAACL 2001.

[JNISpec] Java Native Interface Specification, http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/jniTOC.html, last visited on 15-09-2007

[MediaFoundationWeb] Microsoft Media Foundation SDK, http://msdn2.microsoft.com/en-us/library/ms696274.aspx, last visited on 15-09-2007

[Meneses02] ISEL, Processamento Digital de Fala, Sebenta de Apoio, Carlos Meneses, 2002

[Mermel76] P. Mermelstein, "Distance measures for speech recognition,

psychological and instrumental, in Pattern Recognition and Artificial Intelligence, C. H. Chen, Ed., pp. 374-388, Academic, New York, 1976

[MLDCWeb] Microsoft Language Development Center, www.microsoft.com/portugal/mldc, last visited on 15-09-2007

[MobileSDK] Windows Mobile Developers, http://www.microsoft.com/uk/windowsmobile/developers/default.mspx, last visited on 15-09-2007

[MSNWeb] Microsoft Network, http://www.msn.com/defaultO.aspx, last visited on 15-09-2007

[NuanceT9] Nuance T9 Solutions, http://www.nuance.com/t9/, last visited on 15-09-2007

[SAMPAWeb] UCL, Speech Assessment methods phonetic alphabet, http://www.phon.ucl.ac.uk/home/sampa/index.html, last visited on 15-09-2007

[Sakoe78] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", IEEE Trans. Acoustics, Speech, Signal Proc., ASSP-26 (1): 43-49, February 1978

[SpeechSDK] Microsoft Speech Technologies, http://www.microsoft.com/speech/speech2007/default.mspx, last visited on 15-09-2007

[Speecon99] Corpus Design, http://www.speechdat.org/speecon/index.html, last visited on 15-09-2007

[StandartSDK] The STANDARDSDK_500 can be found in http://www.microsoft.com/downloads/details.aspx?FamilyID=FA1A3D66-3F61-4DDC-9510-AE450E2318C3&displaylang=en, last visited on 15-09-2007

[Rabiner86] Rabiner, L. R. and Juang, B. H., "An introduction to Hidden Markov Models, IEEE ASSP Maganize, 4-15, January, 1986

[Rabiner89] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". Proceedings of the IEEE 77(2):257–286, February 1989.

[Rabiner93] L. Rabiner e B. Juang, *Fundamentals of Speech Recognition*, Prentice Hall Signal Processing Series, 1993.

[TeamSystemWeb] Microsoft Visual Studio Team System, http://msdn2.microsoft.com/pt-br/teamsystem/default.aspx,

last visited on 15-09-2007

[Teixeira97] C. Teixeira, I. Trancoso, and A. Serralheiro: Recognition of Non-Native Accents. in Proc. European Conference on Speech Communication and Technology (Eurospeech), vol. 5, pp. 2375–2378, 1997

[VCProject] MLDC, Voice Command Project, http://www.microsoft.com/portugal/mldc/projects/voicecommand.mspx, last visited on 15-09-2007

[VCUserGuide] Voice Command User Guide

[VisualStudioWeb] Microsoft Visual Studio, http://msdn2.microsoft.com/pt-br/vstudio/default.aspx, last visited on 15-09-2007

[Viterbi67] Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. on Information Theory, 13(2), pp. 260-269, 1967

[VoiceSignalWeb] Voice Signal Technologies 2007, http://www.voicesignal.com/solutions/index.php, last visited on 15-09-2007

[W3CXMLWeb] World Wide Web Consortium (W3C), “Extensible Markup Language”, Architecture domain, http://www.w3.org/XML/, last visited on 15-09-2007