Design of a Speech Interface for Augmenting Desktop Accessibility

David Manuel Fernandes Rodrigues

Dissertação para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Júri

Presidente: Professor Paulo Jorge Pires Ferreira
Orientador: Professor João Paulo da Silva Neto
Co-Orientador: Professor Nuno João Neves Mamede
Vogal: Professor António Manuel Ferreira Rito da Silva

Setembro 2008
ACKNOWLEDGMENTS

I would like to express my sincere gratitude towards Professor João P. Neto, whose leadership, guidance, understanding, and patience, have set an example. I was particularly fortunate to have him as a supervisor and I thank him for the support and flexibility to choose and pursue research ideas.

I must also acknowledge Engineer Renato Cassaca for his invaluable assistance. He was abundantly helpful and offered unconditional support at times of critical need. This project would not have been successful without him.

Appreciation goes out to my many colleagues at L²F. They have provided me with an optimum working environment, which made this whole journey seem so much easier.

Most importantly, I would like to give special thanks to all my friends who helped me get through my years of graduation. Not forgetting my best friends who have always been there during my entire life.

Finally, I want to dedicate this work to my parents for driving me to pursue my master's degree. Needless to say, all my accomplishments would not have been possible without their support. I owe them special gratitude for continuous and unrestricted love and understanding.
ABSTRACT

The goal of this thesis is to describe our work on the development of a platform which empowers the computer interface with multimodal interaction. With this platform, we extend the accessibility of an operating system by introducing a layer of software that stands between our modality engines and the user applications. The system supports speech input and output, respectively, through automatic speech recognition and text-to-speech engines, and also gives visual feedback through an animated face. The target language intended for the voice modality is the European Portuguese.

We present a middleware structure, which supports the incorporation of the referred modalities in the interface’s target environment. This structure abstracts the low-level access to the engines, providing a common way to develop application-level logic that capitalizes from their services. On the other hand, it is also responsible for the engines’ functional integration, allowing the emulation of task-independency.

The integrated multimodal technologies create innumerous opportunities to enhance user interaction with computer applications. These communication possibilities are also explored in the work inherent to this thesis. To do this, we employ an embodied conversational agent interface, designed to operate in the scope of the desktop environment. The agent’s purpose is to assist users on command, control, and dictation tasks.

The overall platform was tested and evaluated with usability criterion. The users’ feedback proved to be indicative of the acceptability, the difficulties, and the interest of the developed multimodal interface.

Keywords: User interface; multimodality; embodied conversational agent; speech recognition; speech synthesis; animated face.
RESUMO

O objectivo da presente tese é descrever o desenvolvimento de uma plataforma que visa potenciar as interfaces do utilizador com interacção multimodal. Esta plataforma pretende extender a acessibilidade no sistema operativo, introduzindo uma camada de software que se situa entre os motores modais e as aplicações. É suportada a entrada e saída de fala, respectivamente, através de motores de reconhecimento automático, e de síntese de fala, e é igualmente fornecido retorno visual através de uma cara sintética. A língua alvo pretendida para as modalidade verbais é o Português Europeu.

Apresentamos uma estrutura de middleware, a qual fornece uma envolvente tecnológica que suporta a incorporação das modalidades referidas. Ela permite abstrair o acesso de baixo-nível aos motores modais, introduzindo uma forma genérica de desenvolver lógica aplicacional que capitaliza dos seus serviços. Por outro lado, esta estrutura é também responsável pela integração funcional das modalidades verbais, permitindo emular independência entre domínios de tarefa.

As modalidades integradas criaram inúmeras oportunidades para melhorar a interacção entre utilizadores e aplicações. Estas possibilidades comunicativas são também elas exploradas no trabalho inerente a esta tese. Nesse sentido, utilizámos uma interface baseada num agente conversacional, a qual foi projectada para operar no âmbito do ambiente de trabalho do sistema operativo. O objectivo do agente é assistir os utilizadores em tarefas de comando, controlo, e ditado.

O nosso sistema foi devidamente testado e avaliado recorrendo-se ao uso de critérios de usabilidade. A reacção dos utilizadores provou ser indicativa da aceitabilidade, das dificuldades, e do interesse da interface multimodal desenvolvida.

Palavras Chave: Interface do utilizador; multimodalidade; agente conversacional; reconhecimento de fala; síntese de fala; cara sintética.
# Table of Contents

**Acknowledgments** ........................................................................................................ iii  
**Abstract** .................................................................................................................................... v  
**Resumo** .................................................................................................................................... vii  
**Table of Contents** ............................................................................................................. ix  
**List of Figures** ..................................................................................................................... xiii  
**List of Tables** ....................................................................................................................... xv  
**List of Acronyms** ................................................................................................................... xvii  

## 1 Outlining the Project ................................................................. 1  
1.1 Background ......................................................................................... 1  
1.2 Motivation .......................................................................................... 2  
1.3 Project Context .................................................................................. 2  
1.4 Project Objectives ........................................................................... 3  
1.5 Project Overview ............................................................................. 4  
1.6 Thesis Outline .................................................................................. 6  

## 2 Background and State of the Art ............................................... 7  
2.1 ECA Background ........................................................................... 7  
2.1.1 Introduction to ECAs ................................................................. 7  
2.1.2 ECA’s Interaction Requirements................................................ 8  
2.2 Proposed Architectures ................................................................... 10  
2.2.1 Perception ................................................................................... 11  
2.2.2 Deliberative Behavior ................................................................. 12  
2.2.3 Reactive Behavior ....................................................................... 13  
2.2.4 Mediation and Action ................................................................. 13  
2.3 Related Work .................................................................................. 14  
2.3.1 Desktop Assistants ................................................................... 14  
2.3.1.1 Peedy .................................................................................. 14  
2.3.1.2 Paper Clip ............................................................................ 15  
2.3.2 Talking Faces ............................................................................ 16
3 MODALITY PROVISION: THE ENGINE LEVEL ............................................. 25

3.1 SPEECH RECOGNITION OVERVIEW .................................................. 27
   3.1.1 ASR Engine Structure .................................................................... 27
   3.1.2 Dictation vs. Command and Control ........................................... 28
   3.1.3 Vocabulary and Grammars .......................................................... 28
   3.1.4 Task-Independency Problem ......................................................... 30
   3.1.5 Speaker Dependency ................................................................... 31
   3.1.6 AUDIMUS Recognition System ..................................................... 32

3.2 SPEECH SYNTHESIS OVERVIEW ....................................................... 33
   3.2.1 Synthesizer Types ....................................................................... 33
   3.2.2 TTS Engine Structure .................................................................. 34
   3.2.3 DIXI Text-to-Speech Synthesizer ............................................... 34

3.3 THE FACE ............................................................................................. 35

4 THE MIDDLEWARE SUPPORT ................................................................. 37

4.1 INTEGRATING SPEECH IN THE DESKTOP ENVIRONMENT ............ 37

4.2 ACHIEVING INTEROPERABILITY AND TECHNOLOGICAL TRANSPARENCY .......................................................... 39
   4.2.1 Component Object Model Wrapping ......................................... 39
4.3 CONNECTING SPEECH ENGINES WITH APPLICATIONS ........................................... 41
  4.3.1 The First Approach: Microsoft Speech API ......................................................... 41
    4.3.1.1 Introduction to Microsoft SAPI ...................................................................... 41
    4.3.1.2 Our Microsoft SAPI Implementation ................................................................. 42
    4.3.1.3 Drawbacks of Microsoft SAPI ....................................................................... 43
  4.3.2 The Modality Server ...................................................................................... 44
    4.3.2.1 Server Technological Details ........................................................................... 45
    4.3.2.2 Middleware Characteristics ........................................................................... 46
    4.3.2.3 Two-Way Communication ............................................................................. 46
    4.3.2.4 Management of Shared Engines .................................................................. 47
    4.3.2.5 Concurrency Control .................................................................................... 49
  4.4 ACCESSIBILITY AUGMENTATION: SIMULATING TASK INDEPENDENCY .......... 50
    4.4.1 Context Management ...................................................................................... 50
    4.4.2 Language Model Administration and Functional Handling .................................. 51
      4.4.2.1 Base Spoken Interface: The Invariable Command Sets ........................................ 51
      4.4.2.2 Attaining Portability with a Dynamic Command List ....................................... 52
      4.4.2.3 Granting Extensibility: The Add-In Approach .................................................. 55
  5 INTERFACE CONCEPT: THE APPLICATION LEVEL ............................................... 57
    5.1 DESIGNING THE SYSTEM’S SPEECH INTERFACE ............................................. 57
      5.1.1 Feedback ........................................................................................................ 57
      5.1.2 Interface Input ................................................................................................. 60
      5.1.3 Error Recovery ............................................................................................... 61
      5.1.4 Error Correction ............................................................................................. 61
      5.1.5 Application Integration ................................................................................... 62
    5.2 THE SUPERVISED APPLICATIONS ..................................................................... 63
      5.2.1 Word Processing Application ........................................................................ 63
      5.2.2 E-Mail Application .......................................................................................... 64
    5.3 THE INTERACTION’S CHARACTERISTICS ....................................................... 66
      5.3.1 Interaction’s Overview .................................................................................... 66
      5.3.2 Use Case: Creating an E-mail .......................................................................... 66
  6 EVALUATION AND STUDIES ............................................................................. 69
    6.1 HEURISTIC EVALUATION ................................................................................... 69
LIST OF FIGURES

Figure 1 - The platform’s overall architecture ................................................................. 4
Figure 2 - ECA Software Architecture ............................................................................. 11
Figure 3 - Peedy .................................................................................................................. 14
Figure 4 - Paper Clip .......................................................................................................... 15
Figure 5 - DECface ............................................................................................................ 16
Figure 6 - Baldi .................................................................................................................... 16
Figure 7 - Cosmo ................................................................................................................ 17
Figure 8 - STEVE ............................................................................................................... 18
Figure 9 - Helper Agent ..................................................................................................... 19
Figure 10 - Olga .................................................................................................................. 20
Figure 11 - Gandalf ............................................................................................................ 20
Figure 12 - REA .................................................................................................................. 21
Figure 13 - Block diagram of the overall system ................................................................. 25
Figure 14 - Structure of an ASR engine ............................................................................. 27
Figure 15 - Structure of a TTS engine ............................................................................... 34
Figure 16 – The ECA’s facial expressions ......................................................................... 35
Figure 17 - Voice activation of a desktop system ............................................................... 37
Figure 18 - Block diagram of the original system using Microsoft SAPI ............................ 42
Figure 19 - Control panel for choosing among available SAPI engines .............................. 42
Figure 20 - The creation of a document using Microsoft SAPI provided speech services .... 43
Figure 21 - Block diagram of the Modality Server ............................................................. 45
Figure 22 - Inter-process communication ............................................................................ 46
Figure 23 - The process of binding to MS’s named singleton object .................................... 48
Figure 24 - Classification of the platform’s spoken commands .......................................... 51
Figure 25 - Control view of the UI Automation tree and properties of a specific UI element ................................................................................................................ 53
Figure 26 - Dynamical creation of command grammars and storage of respective control patterns ........................................................................................................ 53
Figure 27 - Hash table structure containing the interface control patterns ......................... 54
Figure 28 - Command recognition followed by the invocation of the corresponding action .................................................................................................................. 54
Figure 29 - Add-in binding process .................................................................................... 56
Figure 30 – Controller gadget which returns feedback about the ASR state and the spoken input .............................................................. 58
Figure 31 - Interface loading message ................................................................................. 59
Figure 32 – Interfaces for the selection of the: a) ASR models; b) TTS voice ...................... 63
Figure 33 – The dictation of a document using our system ................................................... 64
Figure 34 – The creation of an email using our system ......................................................... 65
LIST OF TABLES

Table 1 – The heuristic evaluation’s results. ................................................................. 71
Table 2 – The usability measures for the document creation task.............................. 73
Table 3 – The usability measures for the e-mail creation task. ..................................... 74
**LIST OF ACRONYMS**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALI</td>
<td>Application-Level Interface</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASR</td>
<td>Automatic Speech Recognition</td>
</tr>
<tr>
<td>BDI</td>
<td>Belief-Desire-Intention</td>
</tr>
<tr>
<td>CFG</td>
<td>Context-Free Grammar</td>
</tr>
<tr>
<td>CM</td>
<td>Context Manager</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>ECA</td>
<td>Embodied Conversational Agent</td>
</tr>
<tr>
<td>ELI</td>
<td>Engine-Level Interface</td>
</tr>
<tr>
<td>FMTB</td>
<td>Functions Modalities Timing and Behavior</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>L2F</td>
<td>Laboratório de sistemas de Língua Falada</td>
</tr>
<tr>
<td>MS</td>
<td>Modality Server</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SAPI</td>
<td>Speech Application Programming Interface</td>
</tr>
<tr>
<td>TTS</td>
<td>Text-To-Speech</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>VU</td>
<td>Volume Unit</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
PART I

OUTLINING THE PROJECT

1.1 BACKGROUND

During recent years computers have witnessed an emerging paradigm in user interfaces (UIs). Arrival of new technologies and devices are allowing the realization of more natural interactions.

In a quest for more flexibility, intuitiveness, and robustness in human-computer interaction, a growing interest has emerged in multimodal interfaces. These user interfaces support more challenging applications, and more difficult usage conditions by combining the strength of multiple and sometimes complementary communication channels.

The interpretation of multimodal input started to be studied with Bolt’s innovative work on deictic gesture, in a project named Put-That-There (BOLT, R. A., 1980). Since this work much research on multimodal systems has been presented: Koons (KOONS, D. B. et al., 1993), Bolt and Herranz (BOLT, R. A. and Herranz, E., 1992), Wahlster (WAHLSTER, W. et al., 1991), Johnston (JOHNSTON, M. et al., 1997) and others.

Nowadays, several applications are trying to take advantage of these sophisticated interface technologies to facilitate the accomplishment of different tasks. However, the concept of multimodality has proven to introduce considerable design and development overheads to the implementation of such systems. Actually, the designing phase of some complex applications must consider support for numerous features, such as: context sensitivity; dialogue capabilities; etc.

The research developed in the scope of this thesis focuses on a particular type of multimodal systems, which are embodied conversational agents. These virtual agents represent animated anthropomorphic characters that can engage in real-time multimodal interactions using speech, gesture, posture, or other verbal and non-verbal behaviors, emulating the experience of human face-to-face dialogue (CASSELL, J. et al., 2000).

ECAs are emerging as the basis for the implementation of software in entirely new domains. They are increasing their popularity, as the promising new interface for many common computer applications. Numerous researchers have turned their attention into the development of this new generation of systems: Cassell (CASSELL, J. et al., 1999), André and Rist (ANDRÉ, E. and Rist, T., 2000), Bates (BATES, J., 1994), Lester (LESTER, J. C. et al., 2000) and others.
The major challenges that have been presented, regarding ECA development, are related to: the definition of which modalities to use at a particular situation; the interpreting of information from different input sources; and the coordination of parallel output modalities. More ambitious researches have also been trying to provide ECAs with self behavioral and personality models, enabling the agents to act reasonably autonomous (CASSELL, J. et al., 2000).

The increasing interest in ECAs has culminated in the fast increase of research issues, and in a wide diversity of approaches for different domains. There have been a great number of used paradigms and development tools. Because of all this heterogeneity it is still very hard to evaluate and compare ECAs (RUTTKAY, Z. et al., 2002).

1.2 MOTIVATION

The primary motivation behind the development of ECAs is the attempt to deal with the increasing complexity of current software systems. These intelligent user interfaces aim to surpass the traditional ones that require and limit users to directly manipulate passive artifacts. Instead, they propose the deployment of opportunistic, delegated and adaptive agents, which have proactive behavior and help users with their tasks.

With the ECA approach, exchanging information becomes possible through the interplay with anthropomorphic characters. The nature of this interaction expresses another major objective of ECAs: the transformation of human-machine interfaces into face-to-face dialogues. The desired advantages become quite obvious considering the role of human body in daily life conversation. In fact, the way we use our speech, body movements, facial expressions, and eye gestures greatly impacts the whole dialogue experience.

For all the referred reasons, we can conclude that the main goal is the deployment of, even more sophisticated, but also simpler, user friendlier, and more natural user interfaces.

1.3 PROJECT CONTEXT

Researching on ECAs draws upon breakthroughs in several fields, which include intelligent user interfaces, multimodal user interfaces, and the technologies that support this multimodality.

This particular project is established on work, developed in L²F, on base technologies such as automatic speech recognition and text-to-speech. AUDIMUS (NETO, J. P. et al., 1999), which is the recognition system for European Portuguese language, and DIXI (CARVALHO, P. M. et al., 1998), the Portuguese text-to-speech system, played essential roles in the achievement of our objectives.

In our laboratory we have been working on these technologies applying them in a wide variety of environments, with distinct goals. They have been integrated, in a generic spoken dialogue system
(NETO, J. P. et al., 2006), and have been used to create ECAs that provide sophisticated user experiences.

We have also been researching speech interfaces in different scenarios including telephone interactive voice response systems and home automation (NETO, J. P. and Cassaca, R., 2004). These interfaces provide the user-friendly means to handle information systems, since speech is probably the most natural of all interaction modalities.

The diversity of contexts, which have been interesting to us, has, however, imposed some designing restrictions to the overall systems’ architectures. We have faced different functional domains, due to the distinct application environments, where our base technologies are intended to work. Nevertheless, the current architectures show flexibility enough to deal with this diversity, allowing different types of user interactions.

1.4 PROJECT OBJECTIVES

The work underlying this thesis tries to present a step forward, in respect to the introduction of multimodal interaction through a virtual conversational agent. The objective is the creation of an ECA which provides an interface between users and desktop applications.

Since ECA development is a very broad field, we needed to constrain our goals for this project. For this reason, we have mainly focused on the speech interaction potentialities. With this said, our platform’s primary objective is to empower people to interact with their applications by voice. It is designed to allow a significant reduction of mouse and keyboard use, while preserving or even boosting the overall productivity of human users.

In order to improve the computer’s accessibility we expose our general effort to explore and understand the interaction opportunities which exist in the context of desktop environments. The goal is to present a usable ECA interface which is able to guide the user to complete the task at hand. Regarding the dialogue processing, we suggest a simple spoken dialogue system with the purpose of controlling the overall interaction.

In terms of modalities, we propose a character that is embodied with an animated face, and supports the combination of facial expression, speech recognition, and speech synthesis. The ECA makes use of all these communication modes to exchange information with the users. Nevertheless, in this particular project, we give much more emphasis to the agent’s spoken interface. For this reason, a substantial part of the presented work will relate to the integration, application, and usability of ASR and TTS technologies.

The system is specifically aimed to work under Microsoft Windows. The main reason why we chose this operating system (OS) as our target platform refers to the fact that we want to make our technology available to the largest possible audience, so we had to go with the dominant OS in the world’s personal computer market.
In order to describe and analyze our approach we mainly refer to its implementation on the Windows Vista OS. We will use this work to expose the possibilities that our modal technologies create in order to enhance user interaction with application interfaces. This implementation is also essential to test and evaluate the overall platform. The users’ feedback is indicative of the acceptability, the difficulties, and the interest of the ECA interface.

1.5 PROJECT OVERVIEW

Figure 1 exhibits a general view of our platform's architecture and component arrangement.
The project’s starting point was founded on the speech engines referred in section 1.3. Our first concern was the low-level integration of these base technologies in the platform’s target environment. We were interested in a solution supporting technological transparency and an efficient employment of engine resources. Additionally, we also wanted to deploy an interface which represented a common protocol on which clients could access the speech services.

With the above purposes in mind, we began by developing a software approach specifically aimed at fulfilling the gap between our base technologies and desktop applications. The engines were wrapped with component object model (COM) technology, which conferred interoperability, allowing clients to access them as if calling objects within their respective environment. On the other hand, we implemented a generic modality server responsible for centralizing the access to engine services. With this middleware component we became able to control the lifetime of singleton engine instances, and to handle innumerable issues, such as concurrent access and resource management.

The modality server was initially designed to support speech recognition services. Later we got interested in expanding it with speech synthesis capability, ending up with a platform that is extensible in respect to input and output modalities.

Concerning the agent’s visual representation, we adopted an avatar face with reactive animation. Since this project is not particularly focused on the agent’s embodiment, we chose a simple approach accomplished by an independent reactive module. The lack of complexity in this component did not justify its encapsulation by the modality server. For this reason, we defined its services directly accessible from the platform’s application-layer.

The second phase of our work consisted in the development of application-level logic, which would capitalize from the previously integrated technologies. In a prior moment of this stage, we had to concentrate on the architectural structure supporting the future multimodal interface.

A fundamental aspect of our approach was the desire to make the platform generic. We wanted to make it available in applications with different functional domains. Nevertheless, since speech engines are not task independent, we had to define a strategy that would allow us to support a large diversity of application contexts.

The solution that was adopted establishes on intelligent runtime administration of engine resources, allowing our system to adjust to a broad variety of situations. However, certain capabilities became crucial to back up this approach. We had to develop a component responsible for context management and awareness. The knowledge provided by this module enabled the detection of the correct timing for engine adjustments, and the respective modifications.

Another focal point of our concerns was on the platform’s extensibility and adaptability. In fact, we were interested in the future expansion of the ECA’s behavior and spoken interface. Furthermore, we pretended to accomplish a system capable of adapting to unforeseen interaction opportunities.
Regarding the extensibility problematic, we adopted a strategy based on add-in integration. We defined an application interface which would be accessible to these independent extension modules and implemented a control mechanism to supervise the commutation between their procedures.

In respect to adaptability, we established screen reading routines responsible for returning information about unpredictable interface possibilities, including their respective control patterns. In this way, our platform became acquainted of all available UI elements, and able to invoke their functionalities.

The third and final phase of this project covered the functional programming. At this point we had to focus on: the design of the speech interface; the implementation of handler routines for the spoken commands; the definition of the agent’s behavior; and the combination of multimodal output information.

The resulting platform enables innumerable ways to control the OS desktop environment with voice. It allows people to dictate to their computers and have their speech automatically transcribed into their email or word processing programs. Starting and switching between applications, managing e-mails, selecting portions of text, correcting sentences, synthesizing documents, and selecting menu options are just some of the interaction possibilities.

Concerning the ECA’s behavior, we defined a set of proactive actions based on reactive and deliberative intelligence. In this way, the agent became able to engage dialogues with users, on certain key moments, with the purpose of helping them to fulfill their objectives.

At the final stage of this project our main focus was to maximize the quality, utility, and usability of the provided modalities. The system has been presented to some beta testers reveling itself to be a popular approach to home desktop use. Yet, considerable effort is still required in order to validate the interface. All the same, this work represents an excellent opportunity to make our base technologies generally available and at the same time give us a source of feedback which leads to continuous improvement.

1.6 THESIS OUTLINE

In Part 2, we are going to dissect the ECA concept, and present relevant background information. Additionally, we’ll detail, in some extent, the state of the art of ECAs, analyzing work of other researchers.

Part 3 gives an overview of how the modalities are provided on our platform, presenting the engines that were employed. It also introduces fundamental knowledge related with speech technologies, and essential for a sustained understating of our work.

Part 4 focuses on the platform’s architecture details. It describes the middleware structure supporting the multimodal interface, and explains how the speech engines were integrated on the OS environment.

Part 5 specifies the platform’s application-level. It lists the principles that were followed while designing the multimodal interface, and describes the interaction’s characteristics.

In Part 6, some usability studies are exhibited, as well as the tests that were made to the system.

Finally, the conclusions are presented in Part 7.
2.1 **ECA Background**

2.1.1 Introduction to ECAs

Virtual agents, with human features, are becoming an integral part of interfaces, providing a multitude of services and utilities. With their employment, the humans’ experiences with computers are becoming more interactive, friendly and natural.

These virtual characters can take on a great variety of forms and representations. They can be: embodied or disembodied; personified or anthropomorphic; represented by 2D cartoons, 3D models, or video sets; etc. They may also exert many distinct functions. They can serve as: guides; avatars; pedagogical agents; intelligent assistants; entertainers; etc.

The work being developed in the scope of this thesis focuses on ECAs, which are a specific kind of virtual characters and a particular type of multimodal systems.

These conversational agents may be defined as those having similar properties to humans in face-to-face dialogue, which include being able to (CASSELL, J. et al., 2000):

- Recognize and react to verbal and non-verbal input.
- Deal with conversational functions.
- Perceive and give signals that indicate the state of the conversation.
- Generate verbal and non-verbal output.
- Bring new propositions to the conversation.

ECAs are being increasingly employed in the development of a new family of user interfaces for a wide diversity of applications. The goal is to try to take advantage of these sophisticated interface technologies in order to facilitate the accomplishment of different tasks.
2.1.2 ECA’s Interaction Requirements

The interactions with an embodied conversational interface may rely on several modalities like speech, prosody, gaze, gesticulation, facial expressions and head movements. The speaking entity makes use of these communication channels, in parallel, combining modalities as needed for appropriate content elaboration. On the other hand, the listening entity simultaneously constructs multimodal feedback and appropriate answers.

Representing virtual agents with faces and bodies provides the opportunity for a wide range of behaviors that, when performed in close synchronization with speech, carry out a communicative function (CASSELL, J. et al., 2000). It is important to understand that particular behaviors can be employed in a variety of circumstances to produce different communicative effects, and that the same communicative function may be realized through different sets of behaviors.

The behaviors that directly contribute to the content delivery or the organization of the conversation are termed conversational behaviors and are the surface form of the exchange.

But it is also important to identify the functions that these conversational behaviors serve. Typical discourse functions include conversation initiation and termination, turn-taking and interruption, and finally, giving and requesting feedback.

Concerning the engaging and disengaging of conversations, elaborate rituals must be taken by the agents to act humanly.

Typical conversational behaviors that carry out these communicative functions include:

- Smiling, embracing, raising eyebrows, glancing, or waving, when it is pretended to invite contact or make a salutation.
- Looking, head nodding, waving, or glancing around, when it is pretended to terminate conversations.

A turn-taking sequence must be imposed on the conversation to prevent interlocutors from talking at the same time.

Typical conversational behaviors that carry out conversational turn-taking and interruption include:

- Looking, raising eyebrows, or stop talking, when it is pretended to give turn.
- Raising hands, when it is pretended to wait for a turn.
- Glancing away, or start talking, when it is pretended to take turn.

Finally, concerning the non-verbal feedback from listeners, conversational behaviors that carry out this communicative function include:

- Looking, or raising eyebrows, when it is pretended to request feedback.
- Looking, or head nodding, when it is pretended to give feedback.
This mapping from behavior to conversational function depends on a basic separation of communicative goals (CASSELL, J. et al., 2000): the contribution to a dialogue can be partitioned into interactional information and propositional information.

Propositional information correlates with the essence of the dialogue and consists of meaningful discourse as well as gesture, head nods, body movements, facial expressions and intonation used to refine or add detail upon the dialogue content.

Interactional information contains hints that influence the conversational process and contains a spread of non-verbal behaviors as well as regulatory speech.
2.2 PROPOSED ARCHITECTURES

Materializing the idea of an ECA into a tangible computer system involves the development of a sophisticated control architecture. State of the art ECAs, which can efficiently engage in face-to-face dialogues, possess certain key characteristics (CASSELL, J. et al., 2000): multi-modal input and output; real-time; propositional and interactional information; conversational function model; modularity and extensibility.

Several architectures have been proposed in the field of multimodal systems and conversational agents. The general approach of most researchers has been focusing in linear architectures (BESKOW, J. and McGlashan, S., 1997)(BALL, G. et al., 1997). The main idea in these architectures suggests that data flows from user input, passing through different internal modules before reaching agent output.

Different approaches, like the subsumption architectures, have however been suggested. These architectures try to decompose complicated intelligent into many simple modules, which are organized into layers. Each one of these layers contains a particular goal of the agent, and higher layers are increasingly more abstract.

Nagao and Takeuchi (NAGAO, K. and Takeuchi, A., 1994) presented a conversational agent based on the subsumption architecture. In this way, their agent is based on a horizontal decomposition of task achieving modules. Each one of the modules competes with the others to see which is active at a particular moment.

Considering the definition of internal modules and the interaction between them, many researchers have taken different paths in their architectures. However, similarities exist, mainly due to the common effort of implementing some, or all the key features presented in the beginning of this section.

Work developed in the Faculty of Technology University of Bielefeld, has resulted in an ECA, named Max (KOPP, S. et al., 2006), controlled by a cognitively motivated agent architecture displaying the perceive-reason-act paradigm. The perceive-reason-act triad is a classic approach still essential to understand almost all of the architectures presented until today.

Max also displays many strategies generally applied in the development of ECAs. It presents a reactive component, responsible for reflexes and immediate responses. This component makes a direct connection between perception and action. It also contains a reason section where deliberation processes take place. The BDI model (RAO, A. S. and Georgeff, M. P., 1991) is adopted, meaning that the agent is driven by believes, desires and intentions. Finally, it processes the perception, reason, and action concurrently so that reactive responses and deliberative actions can be simultaneously calculated.

Another architecture proposal is the one suggested by latest Cassell’s investigation towards the REA agent. This work is very important in the state of the art of ECAs, being, probably, the most complete architecture that has been presented. The architecture meets all the requirements for an ECA that can
participate in a face-to-face conversation, handling real-time response to interactional cues, generation of multimodal inputs and outputs, and deep semantic understanding. It maps specific features of all the modalities into conversational functions and uses a common knowledge representation format. This use of a uniform knowledge representation allows also the system, to be extensible with respect to multimodalities, and modular with respect to plugging in new modules implementing alternative discourse theories.

In most of the analyzed systems, there is a common idea, of how to partition the architecture into modules. At a high level, the architecture of an ECA should contemplate the components responsible for: perception; deliberative behavior reactive behavior; mediation and action. Fig. 4.1 illustrates an architecture with these properties.

![ECA Software Architecture](image)

**Figure 2 - ECA Software Architecture.**

### 2.2.1 Perception

A perception component is responsible for collecting inputs from all the available modalities, provided by access to different sensory input channels. This module will receive information from gadgets and devices that provide speech recognition results, user gesture, gaze information, and other input modalities.

Perception, however, is not supposed to be only a simple acquisition of data but should also do it in a flexible way, allowing it to be led by context focuses.
Usually this architectural component is also responsible for converting the data into a form usable by other modules of the system, and for routing the results to those modules responsible for processing the multimodal input information. The component responsible for the agent's deliberative behavior is generally the one that processes this data. However, because of the need to directly trigger reactive responses, interactional information is, sometimes, also forwarded directly to a component responsible for reactive behavior.

In systems adopting the BDI model, the acquired perceptions are usually kept in buffers, and when new important data is recognized (the agent’s beliefs), the most relevant information is written in memory.

In some proposed architectures, as the one of Cassell's Real Estate Agent, the perception component can also aggregate semantic representations to the input events.

### 2.2.2 Deliberative Behavior

The processes from this component handle long-term planning and deliberative responses such as the understanding and synthesis of content. They are responsible for the agent’s dialog strategies during user interactions, analyzing perceptual inputs, and planning and elaborating action outputs.

In terms of discourse functions, these components do the mapping between them and the signal features, processing inputs and outputs at a semantic level.

The BDI architecture, reflects deliberative behavior, and is used in many systems. This approach provides the required resources for modeling intentional actions in form of plans, allowing the performance of complex tasks under presented context conditions. An agent based on the BDI paradigm is driven by beliefs, desires representing its persistent goals, and plans that formulate adequate intentions.

Desires come from internal processing or from interactions with the environment.

Planners make context dependent plans using internal memories. Plans represent intended courses of action and can directly trigger intended behaviors to act, but may also invoke dynamic sub-planners that try to achieve sub-goals. The modularity of the BDI framework actually allows the incorporation of these sub-planners that can operate on certain knowledge representations. This way, complex plans can be hierarchically expanded by the definition of sub-level plans.

Cassel suggests not one, but three architectural modules responsible for delivering deliberative behavior to a conversational agent: an understanding module; a response planner; and a generation module.

The Understanding Module accesses sensorial inputs, knowledge about the application domain, and the discourse context. It is responsible for fusing all this information into a coherent understanding of what is being done by the user.

The Response Planner generates sequences of actions that fulfill communicative or task goals.

The Generation Module produces a set of coordinated primitive actions, realizing conversational functions.
2.2.3 Reactive Behavior

Like the deliberative module, the reactive component of an architecture, instantiates behavior that leads to the performance of actions. Unlike deliberative agents, pure reactive agents only base their courses of action on the information that sensors are picking at that particular moment. In this way, this component is suited to handle real-time interactional functions.

The more sophisticated systems, and state of the art ECAs, combine reactive and deliberative behavior using a hybrid paradigm. In these architectures the reactive module often facilitates immediate responses to events and user actions, being, sometimes, also responsible for incessant secondary behaviors that make the agent appear more lifelike.

2.2.4 Mediation and Action

Most of the ECAs interact with the user using different modalities, making essential the existence of modules, in the architecture, that synchronize and execute output actions across those modalities.

Instantiated behaviors activate themselves dynamically when preconditions have been satisfied, requiring control over actuators. A mediation module of an architecture resolves conflicts during the access to actuators. Many approaches implement mediation, favoring the behavior with a highest priority value.
2.3 RELATED WORK

The development of ECAs, which serve as interface assistants, is an endeavor more difficult and elaborate than simply deploying multimodal systems. These embodied multimodal interfaces use conversation and discourse knowledge to create more natural interactions.

Besides requiring multimodal support these agents also rely on the research from various areas such as: artificial intelligence, computer animation, interface design, sociology, and psychology.

Some successful and unsuccessful systems, which were crucial to the actual state of the art in this area, will be described next.

2.3.1 Desktop Assistants

2.3.1.1 Peedy

“Peedy the Parrot” is the name of a virtual character, embodied as a parrot, which was created by Microsoft Research, in a program named Persona. This agent can be ordered using verbal commands, and its purpose is to manage and play music tracks (BALL, G. et al., 1997).

Peedy was implemented with a framework that allows the creation of desktop based animated characters. This framework is provided by the Microsoft Agent API, which is the basis of many well known agents.

Three major requirements, regarding assistive interfaces, were summarized to guide this project:

- Support for combined initiative interaction.
- Interact with users, exclusively, if the cost of the user making an error is higher, than the cost of the agent interrupting the user.
- Learn, just like people, what is the proper behavior, depending on time of the day, the task and the mood of the users interacting with the agent.

The agent’s architecture is linear, suggesting that data flows from user input, passing through different internal modules before reaching agent output.
Peedy incorporates a basic conversational dialogue manager with speech input, reactive 3D animation, and recorded voice output. In spite of it all, it only uses speech input, and so, cannot perceive other modalities such as non-speech audio and computer vision. Additionally, it can only react when the user finishes an utterance, resembling, in this way, with other multimodal systems described earlier.

A more critical issue is the fact that conversational functions are entirely ignored. The mapping of semantic representation of input is done directly to a task-based representation.

The agent’s embodiment permits the use of “wing gestures” and facial displays. These actions, nevertheless, express only behaviors, instead of instances of specific conversational functions. In this way, individual behaviors cannot be originated as a function of the modalities that are available, or of the previous behaviors, when a function is needed.

In this agent, there is, also, no difference between propositional and interactional behaviors such that both the message and conversational process can be represented.

### 2.3.1.2 Paper Clip

It looks like you’re trying to work. You know that you would prefer to head to Starbucks instead. Would you like to:
- Go for a vanilla white chocolate mocha?
- Go for a caramel macchiato?
- Get a non-free trade cup of Joe, where you can taste the blood, sweat & tears of the children who were forced to pick the beans?

**Figure 4 - Paper Clip.**

One example of a not so perfect embodied interface, that is well known, is the Microsoft Office Assistant (the infamous “Paper Clip”, also known as “Clippy”).

Clippy intervenes in a disturbing and socially inappropriate way, and when the user isn’t actually typing, the character manifests its deep tedium in the user’s work by engaging in conversationally irrelevant behaviors.

However, interruption by itself isn’t a big issue; participants in dialogues often interrupt each other. The problem arises in the nature of the interruption, which must follow the protocol of the dialogue.

Interrupting a user must be related with demands of the dialogue, like requests for further content or feedback in respect at what is being said.
2.3.2 Talking Faces

2.3.2.1 DECface

![DECface images](image.png)

Figure 5 - DECface.

DECface was created at Digital Equipment Cooperation, and relates with the work of this thesis, for probably being the first talking head trying to emulate human interaction (Waters, K. and Levergood, T. M., 1995). Although not an entire embodied virtual character, DECface was one of the earliest 3D virtual faces founded on a realistic muscle model. After the success of DECface, full embodied virtual agents began to show up.

2.3.2.2 Baldi

![Baldi images](image.png)

Figure 6 - Baldi.

Baldi is a 3D computer-animated talking face that is capable of synthesizing speech (Massaro, D. W., 2003). The research behind this agent mostly focuses on visible speech, which quality and intelligibility has been constantly modified and evaluated to accurately simulate naturally talking humans. This agent possesses teeth, a tongue, and a palate to imitate the interior of the mouth.

Baldi proves that there are several reasons why the use of auditory and visual information from a talking head is so successful. These include:

- Content delivery in visible speech.
- Robustness of visual speech.
- Complementarity of auditory and visual speech.
- Optimal integration of these two sources of information.
2.3.3 Proactive Helping ECAs

2.3.3.1 Lester’s Pedagogical Agents

Other major approaches, in the field of embodied conversational agents and proactive interface assistants, were proposed by Lester et al., who developed animated pedagogical agents (LESTER, J. C. et al., 2000).

Herman the Bug (LESTER, J. C. and Stone, B. A., 1997) represents one of these agents, who inhabits a learning environment and proactively provides advice while the student designs plants.

Another virtual agent, that helps the user to accomplish a learning task, is Cosmo (LESTER, J. C. et al., 1997), which operates in the scope of an interactive, screen based learning environment, called Internet advisor. This character is portrayed as a three-dimensional personage that is constantly present in the environment, watching every action of the user and providing hints, explanations, and help.

![Figure 7 - Cosmo.](image)

Empirical studies proved that such interface assistants are able to boost both the learning performance and the student's motivation. The constantly present communicative behaviors have an expressive positive influence on users' perception of the learning experience, which result in a more conscientious and engaged audience.

In respect to output modalities, these virtual characters make a conjunction use of gesture and speech. Still, deictic gestures, in most part, only appear in contexts of referential ambiguity.

Cosmo, for instance, tries to map gesticulation and animations into conversational functions, taking advantage of aiming and referring expressions, to resolve ambiguities in the dialogue, when an item, present in the scene, is referred to.

The combination between verbal and non-verbal behaviors is additive, meaning that, the content delivered by gesticulation is always redundant with the content delivered by speech. In this way, the interface does not take advantage of the potentialities of the body, for those tasks where it might perform even better than speech.
2.3.3.2 STEVE

All the previously described systems relate to situations where the communication is made in static scenarios. The ECAs simply fulfill the role of presenters and humans tend just to observe the presented scenes. However there are some pedagogical applications, which allow human students to execute the tasks that are part of the instructing program, while being supervised by tutoring characters. In this way, these ECAs have to conjugate a communicative behavior with the capability to monitor and give feedback to environmental alterations. These characteristics translate in greater demands for more general cognitive and awareness capabilities.

One example of these pedagogical applications is strictly related to one of the major approaches in the field of ECAs. The agent is known as the Soar Training Expert for Virtual Environments (“Steve”) (RICKEL, J. and Johnson, W. L., 1998), which is an embodied character that trains human how to accomplish manual tasks.

The users interact with Steve, immersing together with the agent, in a virtual reality interface, on which they are able to perform actions making use of multimodal input devices, and see the effect of those actions on the environment. The system supports dialogue functions, using non-verbal cues to orientate attention, giving feedback to the user, and managing turn taking.

However, like Cosmos, the combination between verbal and non-verbal behaviors is additive.

In latest work, the system’s architecture was expanded by a comprehensive dialogue system that supports multimodal, multi-party, and multi-utterance conversations with open, unpredictable dialogues. Although this model is only partially implemented, it grants a large foundation for dialogues, in interactive virtual environments, to distinguish between several layers each concerning a distinct aspect of the dialogue’s information state.
2.3.3.3 **COACH**

Another pedagogical agent, which provides proactive help in software education situations, is “Coach”. This ECA boosts the learning performance of the Lisp programming language students, by giving some feedback about the user’s knowledge and prognosticating the user’s future objectives and actions (SELKER, T., 1994).

It supervises the learner’s actions to create a user model, which is kept to decide when and what kind of unasked advices will be given. Usability studies, that were done, to analyze the “Coach” system with and without the proactive guidance, showed that learners in the proactive guidance cluster on average used all available resources, felt more comfortable with the programming language, had higher confidence, and elaborated five times as many functions than users without the proactive guidance.

2.3.3.4 **Helper Agent**

![Figure 9 - Helper Agent.](image)

The Helper Agent system exemplifies how to take advantage of the support provided by virtual agents, to promote human-human dialogues in the context of video chats (ISBISTER, K. et al., 2000). Each participant possesses an avatar which can navigate freely around the environment of the interface. The Helper Agent consists of a dog-faced animated character, which sole mission is to pay attention to the conversations between the users and to identify silence. Whenever any reticence occurs, by the current speaker, the character comes near the user’s avatar and directs a sequence of simple interrogations to both the dialog participants, which results in the potential suggestion of new topics to discuss about.

The Helper Agent manages to take advantage of social hints by executing the following behaviors:

- Directing the virtual face to the user at the time that he is making the questions.
- Physically approaching and departing from the dialogue.
- Transmit a social attitude to the user, supported by certain animations like: using non-verbal hints to make interrogations, showing feedback to answers, and making recommendations.

The interface assistant related to this thesis will also initiate contact with the users, giving them hints about certain tasks available. This proactively smoothly presents to the users, the available services.
2.3.4 ECAs with Multimodal Input and Output

2.3.4.1 Olga

Figure 10 - Olga.

Olga is an embodied virtual character, with human appearance (BESKOW, J. and McGlashan, S., 1997). This agent engages dialogues in the context of information kiosks. The character speaks about microwave ovens, and is portrayed with a cartoon-like anthropomorphic figure, animated in real-time.

Olga uses a linear architecture, which was previously presented.

The system combines spoken conversation, speech synthesis, gesticulation, a 3D animated talking face, and a direct manipulation interface (KOONS, D. B. et al., 1993). The architecture is distributed and the communication follows a client-server paradigm. Distinct modules are responsible for language processing, interaction management, direct control interface, and output animation. All these modules communicate through a central server.

Olga is compelled by events, meaning that only reacts to the user and isn’t independent enough to initiate output on its own. Like Peedy, Olga doesn’t support low-level audio and visual cues as input modalities.

2.3.4.2 Gandalf

Figure 11 - Gandalf.
Figure 11 presents Thorisson’s work, on a character named Gandalf (THÓRISSON, K. R., 1996). This ECA is part of a pedagogical application, and its mission is to speak about a visual model of the solar system. It is represented with a virtual hand and a head, which are displayed on a monitor, or, as alternative, on a large screen.

Gandalf is an important part of the “Ymir” system, which focus on the integration of multimodal input, from human users, and on the production of multimodal output. The agent perceives and presents interactional cues, also producing propositional content, represented as tinned talking events. These features make the system able to recognize and generate such cues as, turn-taking and backchannel behaviors that conduct to a very genuine dialogue interaction.

In terms of architecture, the “Ymir” system’s approach consists on a multi-level multimodal architecture, which can support fluid face-to-face conversation between a person and the virtual character (THÓRISSON, K. R., 1996).

Input modalities include gesticulation, gaze, speech, and intonation. However, the system is only able to produce restricted multimodal output, in real time, through the animated character.

2.3.5 State of the Art ECAs

2.3.5.1 REA

![REA](image)

Figure 12 - REA.

The “Real Estate Agent” (CASSELL, J. et al., 1999) is probably the most complete reference in the state of the art of ECAs. The system underlying this agent supports multi-modal real-time interaction, through a conversational interface, which is founded in many dialogue protocols. The goal is to simulate the experience of chatting face-to-face with another human, making interactions as natural as possible.

As the name suggests, this agent impersonates, in the present task domain, a real estate salesperson questioning the user about specifications in its repository, and showing users a view of the virtual real state. To interact with the system, users have to stand right next to a big projection screen on which the virtual character is displayed. The agent is embodied with a fully articulated visual figure and perceives humans, in a passive way, using cameras and audio recognizers. Supported modalities include speech
with intonation as well as facial displays and gesticulations, such as hand waves and head movements. Timing emphasizes the need for prompt response as well good synchrony between output behaviors.

This ECA can perceive verbal and non-verbal behaviors directly from users. Computer vision techniques bring the power to react to movements, taking away the necessity of encumbering technology.

Additionally, REA is capable of supporting conversational functions. It can deliver hints that give feedback about the state of the dialogue. It can also react to cues, such as gesticulation for turn taking, allowing being interrupted, and then taking the turn again when it is able. Finally, this agent can start dialogue error correction when something said by a human is misunderstood.

Unlike other approaches, REA was implemented with a dialogue model based on research breakthroughs for human face-to-face interaction. REA’s answers are created by an incremental natural language generation engine that was expanded to synthesize redundant and supplementary gestures synchronized with voice output.

To conclude, the system’s architecture is founded on the Functions, Modalities, Timing and Behavior (FMTB) dialogue framework, on which dialogue functions can be interactional or propositional.

2.3.6 Related Work Analysis

The investigation made in the scope of this chapter gave an overview of important past work related to this thesis. It helped to clarify the concepts indispensible for a better perception of our approach, and also allowed us to situate our system within a broader context of related research.

To conclude the present chapter, we are going to summarize the major features of the described ECAs, categorizing them accordingly. We are also going to explain how our platform interrelates to the mentioned ECAs.

2.3.6.1 Related Work Summary

Various approaches were analyzed, within the domain of human-computer interfaces, and it was illustrated how they can take advantage of sophisticated resources, such as multimodality and intelligent agents, to magnify the user experience.

In this section, we are going to recapitulate the different strategies, adopted by the mentioned approaches, in order to support several requirements of ECA development. These requirements are listed below:

- **Aggregation of modalities:**
  - Systems that only perceive the speech input (*i.e. Peedy*).
  - Speech driven systems (*i.e. Put-that-There*).
  - Systems with additive combination between modalities (*i.e. Cosmo; STEVE*).
  - System that support redundant and complementary modality employment (*i.e. REA*).
• **Reaction capabilities:**
  ◦ Systems which interpret input only when the user finishes an expression. The phrase is the shortest time scale between events being fired (*i.e.* Put-That-There; Peedy).
  ◦ Alerted systems which can react immediately to users. They possess mechanisms that enable them to have environmental stimulus perception (*i.e.* REA).

• **Support for conversational functions:**
  ◦ Systems lacking a representation of communicative actions, which treats the various modalities in alliance (*i.e.* Put-that-There; Peedy).
  ◦ System supporting conversational functions (*i.e.* Helper Agent; Cosmo; Gandalf; STEVE; REA).

• **Proactive behavior:**
  ◦ Event-driven systems, which only react to the users’ input and are not sufficiently independent to initiate autonomous behavior (*i.e.* Olga; Peedy).
  ◦ Systems that can take initiative and demonstrate sophisticated behavior without requiring explicit control (*i.e.* Helper Agent; Cosmo; STEVE).

### 2.3.6.2 Contextualization of our Approach

Like most of the previously described systems, the work being developed in the scope of this thesis contemplates the processing of multimodal input and output, that is, how information is intelligibly conveyed using synchronized verbal and non-verbal modalities.

The design of our platform combines many characteristics of the mentioned systems, integrating them in a way which will be described on latter chapters. These characteristics include:

• **Interface assistance:** The agent assists users to successfully execute tasks, which relate to the command and control of desktop applications.

• **Proactive behavior:** The ECA acts proactively and also initiates contact, with users, on specific situations.

• **Multimodal access:** The interaction can be made by speech, or by mouse and keyboard.

• **Speech recognition and synthesis:** The recognition and synthesis of speech is available. The target language of the employed technologies is the European Portuguese.

• **Talking head embodiment:** The agent is visually presented with an animated avatar face.
• **Conversational function representation:** The avatar’s reactive facial expressions and the mixture of the various modalities, allow explicitly portraying certain conversational functions.
PART III

MODALITY PROVISION: THE ENGINE LEVEL

Figure 13 - Block diagram of the overall system.

Figure 13 portrays a block diagram which illustrates how the multimodal interaction is done within our system, and how the users’ commands are recognized.

The platform incorporates speech recognition, speech synthesis, a reactive animated face, and direct manipulation interfaces. At the present moment the modalities are handled asynchronously and the interaction is guided by a basic dialogue manager.

The user can interact with the system via a keyboard, a mouse, or a microphone. The microphone is responsible for capturing the speaker’s voice and sending it to the sound card, which digitalizes the analog signal and transmits it to our ASR engine. Our engine asks for certain key specifications from the digital signal. Actually, the used sampling frequency of the digitalized signal is 16 kHz, and the number of quantization bits is stipulated to be 16.

A resembling, but inverted flow, occurs between our TTS engine and the playback devices. The engine converts text into audio wave data and, afterwards, the signal is converted into sound and outputted via the speakers.

All the interaction between the applications and the speech modality engines is done through software functions which are specified by engine interfaces. In our platform, the access to these functions is
centralized in a singleton runtime component, called Modality Server (MS), which will be described on a further chapter of this thesis.

The default speech engines, used for the exposed setup, are AUDIMUS ASR engine and DIXI TTS engine. Both these technologies were developed in our laboratory (NETO, J. P. et al., 1999)(CARVALHO, P. M. et al., 1998), and will be described, with moderate detail, in the following sections.
3.1 SPEECH RECOGNITION OVERVIEW

Automatic speech recognition is the process by which computers convert spoken words into machine-readable data, such as binary code for strings of characters. The goal of this technology is, essentially, the transcription of natural speech.

In this section, we are going to discuss the basics of speech recognition and the various characteristics of an ASR engine. The understanding of these concepts introduces some of the difficulties we had to overcome during the technological integration of this modality.

To finish, we will also present the ASR engine that was employed on the scope of this thesis: the AUDIMUS system.

3.1.1 ASR Engine Structure

ASR engines are software components, which have the ability to handle low-level speech recognition tasks. The most important elements of these engines are presented in figure 14.

![Figure 14 - Structure of an ASR engine.](image)

The foremost module inside an ASR engine is normally responsible for feature extraction. This procedure involves withdrawing a sequence of acoustic feature vectors from the digitized speech. The feature vectors contain information related to various attributes of the speech signal, and should be extracted with very high frequency.

Another fundamental stage during speech recognition is the hypothesis search. This process requires much linguistic knowledge at different levels. The lexicon delineates which words can be hypothesized, representing each acceptable word as a sequence of phonemes. The language model represents a linguistic structure, by computing the probability of occurrence of a specific combination of words. Finally, the acoustic model portrays the correlation between the feature vectors and the phonemes.
3.1.2 Dictation vs. Command and Control

The design and functionality of ASR engines generally reflect their intended usage conditions. In fact, they are normally developed regarding specific application areas, which can broadly be aggregated into the following categories:

- Dictation.
- Command and control.

Dictation is the most common area for which systems are targeted for. Applications that support this feature benefit from the engines’ ability to convert speech into text format. Their purpose is not to interpret the spoken input, but to simply display it as a printed sequence of characters. Dictation is particularly useful in active environments, allowing users to work with computers when their hands or eyes are busy. It may also be advantageous in scenarios where the voice input can significantly accelerate certain procedures. These scenarios include, between others, the creation of documents, the writing of emails, and the completion of forms.

Command and control concerns the users’ ability to operate applications through voice instructions. With this capacity, people become able to give orders to the computers and to control their functionality. Unlike dictation, when these commands are perceived, some kind of processing is typically executed based on them. Consequently, the precise recognition of the users’ speech becomes much more critical. Engines employed for command and control purposes can be found in diversified contexts, such as desktop browsers, voice surfing applications, etc.

The procedures by which speech recognition engines accomplish each one of these two application tasks are functionally very different. In dictation applications, the words that the engines are able to recognize, are not predefined and may come from a very large vocabulary. However, in command and control circumstances, the applications are actually aware of all the possible words that users may speak.

3.1.3 Vocabulary and Grammars

As explained earlier, the engines’ knowledge source is provided by the language model, which helps them to predict the succeeding words during recognition. Developers who wish to integrate ASR technologies into their applications should understand two basic concepts inherent to the language models. The first one is the vocabulary, which is defined by the lexicon, and the second one is the grammar.

The vocabulary of an ASR system contains the list of all possible words, or expressions, which may be encountered. Its dimension may vary from very few words, supporting the recognition of only one or two utterances, to many thousands of words. In general, smaller vocabularies will imply a less amount of computation for the engines. These will work well, for example, in systems that were design to provide command and control capabilities. Oppositely, very extensive vocabularies will be more difficult to process, and will introduce much more opportunities for recognition errors. However, these will be needed
to handle certain tasks such as the dictation of documents or messages. For all the referred reasons, a well designed ASR system should allow the adaptation of its vocabulary in respect to its current task.

The second important concept related to language models is the grammar. This element potentiates a reduction of the computer processing and an increase of the recognition accuracy. In fact, the grammar constrains the acceptable combinations of words, and the successive word choices. These constraints reduce the number of alternatives during recognition, boosting the system’s performance. Without the employment of grammars, every term from the vocabulary would have the same probability of occurrence at any point within an utterance. Consequently, continuous speech recognition would become impracticable.

The grammar’s content is formed by a structured set of words and expressions, which are tied together by a group of principles. With this information it is able to specify what that can be recognized at a given time, and to define the format of the outputted text. For these motives, grammars provide a flexible way for application developers to stipulate the expected speech input.

Grammars can be classified into three main categories according to their emphasis. These major subgroups include: dictation grammars; context-free grammars (CFGs); and limited domain grammars.

The dictation grammars are the ones which provide the most accurate results, when converting speech into printed text. Their performance relies on the lexicon and they are usually associated with very large vocabularies. However, despite its dimension, the quality of their vocabulary is defined by the balance between its deepness and the uniqueness of its terms. In fact, the more words on the lexicon, the bigger the probability of the system misinterpreting one term for another. On the other hand, the smaller the lexicon, the vaster the number of unknown terms which might appear. Thus, in order to increase their precision, dictation systems are normally tuned for a specific theme, such as the dictation of medical reports.

Context-free grammars are the most convenient when the spoken interaction is predefined and not casual. These grammars often come into play in situations which involve the recognition from a designated command set. They are able to restrain both the vocabulary and the syntax structure of the engine’s input expectations. In fact, CFGs work by evaluating the correlation of each term and phrase to an established set of grammar rules. By doing this, they are able to determine the most likely candidates for the next possible word at any given moment.

A major aspect of these grammars is their capability to deal with a wide variety of vocabularies. This is because their structure revolves around declaring lists and collections of context-free terms, which only are required to fit into particular patterns. The downside is that, since CFGs base their assumptions on the predefined rules, they are not adequate for dictation, where large vocabularies are the essential factor.

The CFGs’ rules are normally stored in text files, which, in many cases, can be edited in the Extensible Markup Language (XML) format. These files are loaded at run time by the applications, compiled into a
binary format, and then used by the engines to perform the actual speech recognition. As an example, the rules of a context-free grammar could be equivalent to the following:

```xml
<rule id="Element">
  <one-of>
    <item> document </item>
    <item> message </item>
  </one-of>
</rule>

<rule id="Operation">
  <one-of>
    <item> open new <ruleref uri="# Element"/> </item>
    <item> move <ruleref uri="# Element"/> </item>
  </one-of>
</rule>
```

The example above contains the definition of two grammar rules. The first one, ‘Element’, creates a list of possible terms. The second, ‘Operation’, depends on the first rule, and outlines a group of acceptable expressions. In order to extend the interaction possibilities to a wider range of tasks, it would only be necessary to add more rules and/or to expand the existing ones. This effortlessness to enhance the speech interface makes CFGs the most flexible of all grammars. It allows application developers to install systems that only possess essential components, and to expand them later in order to meet the needs of the different users.

Finally, the limited domain grammars are defined by a trade-off between the flexibility of CFGs and the precision of dictation grammars. They represent a viable option when the system’s vocabulary doesn’t have to be considerably large, and work well, for example, in tasks that involve filling-out forms or simple text entries.

### 3.1.4 Task-Independency Problem

Task-independence has to do with the ASR engines’ ability to handle free and unrestricted spoken input. Therefore, it denotes an extremely desirable attribute in a multitude of situations. The problem, however, is that the conception of a perfectly task-independent system still remains an unsolved challenge in the speech recognition area. There is an absence of generic models or vocabularies, which are able to deal with everything that can possibly be said by a user. For these reasons, developers often need to contemplate some kind of mechanism which guaranties the reliability of their applications. Otherwise, systems which have been tested under laboratory conditions will reveal much more inaccurate results in practical circumstances.

There have been a number of advances regarding the task-independency issue. These progresses result, today, in the availability of systems that can be efficiently used on particular tasks. However, the inevitability of having to develop solutions, for each task context, represents an overwhelming effort. It is
necessary to create large enough databases, so that each language model contains the whole phenomena that it must reflect. Still, only in this way will the model’s results be statistically valid and reproduce, as closely as possible, the future employment of the platform in an operational environment.

This task-oriented approach has been commonly used in the design of voice dictation systems, focusing the recognition accuracy on specific subjects. In these cases, the language models are usually created, with relative easiness, from huge amounts of text data available in various sources, such as newspapers, medical reports, etc.

Besides the construction of specific models, the task- independency problem also bears other implications. Actually, this limitation deeply regulates the way in which application developers design their voice enabled solutions. We will present, in posterior chapters, the strategies that we adopted, at this level, with the objective of conferring robustness to our platform.

### 3.1.5 Speaker Dependency

The speaker dependency represents another key element in the designing of voice activated systems. In ideal conditions, this dependence should be minimal, which would denote that the same system could be verbally accessed by diversified users with the same satisfactory results. However, it is very complicated to guarantee recognition accuracy, in a generic way, since people speak rather differently from each other. Besides the existence of many different accents, the speaker speed and pitch inflection may also vary a lot. All these differences create the opportunity for a wide range of pronunciations.

From what was explained in section 3.1.1, it is easy to conclude that the knowledge concerning words’ pronunciations is hold by the acoustic models. These models are crucial in the recognition process because they are used, by engines, to convert the verbalized expressions to the data on which language models will be applied. In this way it is plausible to affirm that speaker dependency intimately relates to these acoustic elements.

Speech systems can be classified according to their speaker dependence, falling into one of following categories:

- Speaker dependent.
- Speaker independent.
- Speaker adaptive.

Speaker dependent systems are the ones that exhibit the most accurate results while consuming the lowest amount of computing resources. In these systems the speech patterns are created, or adjusted, regarding a specific user. Nevertheless, in order to achieve this adaptation, they require a great deal of training from every speaker. Each training session involves the recording of predefined terms, or expressions, and can be rather extensive. All the gathered information is analyzed and saved, providing the necessary knowledge to boost the recognition’s accuracy.
Despite their utility, these training procedures are not practical in many scenarios. This is why the speaker dependent approach is mostly employed in systems which are only used by a restricted group of people. In these cases it is viable to ask users to perform some hours of training in order to construct a precise acoustic model.

The second category that was mentioned corresponds to the speaker independent systems. These are usually chosen when it is not convenient to execute a training phase. Actually, any person can use them without having to accomplish an adaptation process. Therefore, speaker independency is a common option in those situations where numerous users need to work with the same machine. The drawback of these systems lies, however, in the great volume of computational resources that they require. It is also evident that this approach reveals much lesser accuracy than the speaker dependent one.

Lastly, the third category, which is the speaker adaptive, presents a compromise between the two previous methods. Speech systems of this type can perform appropriately without training and are capable of improving with continuous use. In fact, they can adjust the recognition parameters to a specific speaker without requiring tedious adaptation sessions. Their disadvantages include the increased computational requirements and the subsequent reduced performances.

3.1.6 AUDIMUS Recognition System

The nature of our work demanded for an accurate state-of-the-art ASR engine, able to handle vocal commands and considerable volumes of dictation input. Normally, an ASR engine is a substantial technological component, which involved much investment and human resources. Therefore, the emphasis of this particular project did not include engine development.

We made use of an available engine, the AUDIMUS ASR, which delivers speech recognition capabilities through a programmable low-level speech API.

AUDIMUS is hybrid speech recognizer that combines the temporal modeling capabilities of Hidden Markov Models with the pattern discriminative classification capabilities of multilayer perceptrons. This recognizer is being used for different complexity tasks based on a common structure but with different components.

In AUDIMUS, the lexical and the language models are dependent on the specific application domain. Due to the goal of task independency, they should be automatically extracted from a configuration XML file. Since there can be several subsequent uses of the system, there’s a pool of components for each appropriate model that is active in AUDIMUS according to the needs. These models can be associated to single domain or multiple domains. Having different components implies the availability of material collected for that domain.

The acoustic models of the AUDIMUS system are speaker independent.
3.2 **SPEECH SYNTHESIS OVERVIEW**

Speech synthesis, also known as text-to-speech, is a service that provides the capability of converting written text into human speech. Our system makes use of this technology for text reading purposes and also to provide users with an additional output modality.

 Integrating speech synthesis in our platform was rather straightforward, presenting much less challenges than the ASR technology. For this reason, and since the development of a TTS engine is not included in the scope of this thesis, we will only make a brief description of this technology's characteristics.

3.2.1 **Synthesizer Types**

In terms of quality, TTS engines are evaluated by the intelligibility of their output, and by its resemblance to the human voice. The clarity of this output depends mainly on the sophistication of the rules that identify and convert text into audio waveforms. TTS systems normally use one of the following techniques in order to accomplish the referred conversion:

- Concatenation of units.
- Rule-based synthesis.

The concatenative process is achieved by assembling segments of recorded speech which were previously stored. In this way, concatenative systems scan each word and then are able to know which segment combination is the correct one to pronounce it.

The dimension of the referred speech segments considerably impacts the quality of the spoken output. In fact, the storage of entire words or expressions may provide a high quality output. This approach is normally employed for specific usage domains. On the other hand, storing phones or diphones may lead to the synthesis of less comprehensible speech. In spite of that, this last approach is conventionally used since it covers a greater output range.

The second TTS technique that was mentioned refers to rule-based synthesis. This method bases itself on calculations about several human factors, such as the lip’s positioning and the tongue’s orientation. In this way, it is able to emulate a person’s vocal tract, creating an entirely artificial voice output.

Despite being less accurate than concatenative synthesizers, rule-based systems can be easily parameterized. Actually, they normally provide the ability to regulate the intonation, inflection, and speed of the synthesized speech, allowing users to create different voices at their own will. These systems also require fewer computational resources, and less storage capacity than concatenative ones.
3.2.2 TTS Engine Structure

Architecturally speaking, a TTS engine includes three major constituents, which are responsible for the following tasks:

- Text analysis.
- Linguistic analysis.
- Waveform generation.

Figure 15 shows the distribution of these components in a very elementary architecture model.

The front-end module has the important task of analyzing the inputted raw text and transforming it to a normalized version. This step is often referred as tokenization, pre-processing, or normalization. With its execution, certain symbols, such as numbers or abbreviations, become converted into the corresponding written-out words.

The second component is responsible for assigning the phonetic transcriptions for each normalized word. Additionally, it also tags and separates the text into prosodic units, such as phrases and propositions. The conversion procedure, which occurs in this module, is usually known as grapheme-to-phoneme or text-to-phoneme. The outputted information consists in symbolical linguistic representations which include the phonetic transcriptions and prosody related data.

Finally, the back-end module, normally named as the synthesizer, transforms the symbolical linguistic information into the waveform audio representing the speech.

3.2.3 DIXI Text-to-Speech Synthesizer

The task goal of our platform demanded for a high quality TTS engine, able to synthesize speech referent to various domains. Like it was mentioned above, this thesis does not focus on engine development. Therefore, and as with ASR technology, we used an available synthesizer, which also provides access to its services through a programmable API.

Text-to-speech synthesis is supported by DIXI, a concatenative-based synthesizer. This engine supports several voices and two different types of unit: fixed length units (such as diphones), and variable length units.
3.3 **THE FACE**

Besides spoken modalities, the agent underlying this thesis also provides the users with visual representation. Its embodiment is supported by a statically animated avatar face.

Since this work was not particularly focused on any other modalities than speech, we did not spend much time perfecting the agent’s representation. In this way, we suggest a simple approach, based on a reactive module that controls the face’s animation.

Future work might include the replacement of this reactive component by the 3D talking FACE that has been developed in our laboratory (VIVEIROS, M., 2004). Though, one of the major reasons why we haven’t yet used the 3D FACE is the huge amount of computational resources it requires.

The reactive module generates facial expressions, which should be read as natural responses for what the character has perceived (see figure 16). It reacts to specific occurrences, including the shortage of interaction from users.

![Figure 16 – The ECA’s facial expressions.](image)

The ECA’s expressions may reflect a number of conversational functions, such as: acknowledging command recognition; giving feedback about the perception of noise; requesting information to users; expressing the misunderstanding of spoken input; etc.

The conversational behaviors, which carry out the mentioned functions, are generated on the face through a series of still expressions that change in time. These static snapshots represent the frames of the animation. For simplicity reasons, our animated face does not show a lot of frames in a second. Because of this, we are unable to achieve the illusion of smooth movement; nevertheless, this basic approach fits the purposes of the present thesis.

When vocal responses are given, the face is synchronized with the spoken sentences. As an ultimate goal, the agent should move his lips according to the sounds synthesized during speech. However, for the sake of simplicity, we only employ still avatars, suggesting lip movement. The exhibition of these avatars is coordinated with the outputted speech, using real-time information requested from the TTS engine.
4.1 INTEGRATING SPEECH IN THE DESKTOP ENVIRONMENT

We have already given an overview of the speech technologies supporting our platform. The purpose of this chapter is, thus, to introduce the concepts concerning their integration in the context of our work.

Voice activating a desktop environment usually involves the incorporation of the following elements:

- Audio input and output peripherals.
- Speech engines.
- Engine integration middleware.
- Software for handling the engines’ results.

The message flow for the proposed framework is shown in figure 17.

As it was explained earlier, the speech services are provided by independent modules called engines, which can be either speech recognizers or synthesizers.

All the interaction between the speech engines and the applications is done through software functions which are specified by engine interfaces. These software functions are named Speech Application Program Interfaces (SAPI), and their access is generally managed by middleware components. In this way, speech-empowered applications and engines do not interact directly with each other. Instead each of them exchanges information via these software packages that stand in-between.
Another important requirement to successfully integrate speech in the desktop environment is the command handler software. The procedures inherent to this element can be divided by two major goals: the command recognition and the command dispatching.

The modules responsible for the voice command recognition use the SAPI to communicate with the ASR engine and get its results. The purpose of these modules is to make the association between the recognized spoken input and the respective handling logic.

Lastly, the command dispatching modules receive the detected commands and trigger the corresponding effectors on target desktop applications.

The following chapters will deal with the techniques that were followed in order to implement a framework similar to the one in figure 17. We will discuss the software supporting structure which allowed us to integrate the speech engine services within desktop applications.
4.2 Achieving Interoperability and Technological Transparency

Since our platform would integrate distinct technical components, one of our initial concerns was to guarantee software interoperability and technological transparency. Our objective was the achievement of a common way of communication, and data exchange, between the various functional units. This would translate in the employment of a standard set of data formats and exchange protocols. On the other hand, we also pretended to make a system which could be used in different OS environments and accessible from a vast range of applications. This means that our platform had to be capable of exchanging information with diversified technological sources.

4.2.1 Component Object Model Wrapping

We decided to base most of our strategy on the Component Object Model (COM), which is an object-based, Remote Procedure Call (RPC) based, and platform-independent technology. The primary objective of this technology is to provide a framework for integrating components, offering an object model that makes it possible for objects to interact with each other.

COM specifies a standard binary structure for the communication between components. This run-time binary representation provides the basis for software interoperability and helps to promote language-independence. In fact, as long as a compiler can convert the language structure into this binary representation, the implementation language for objects and clients can be arbitrary.

By applying COM to build most of our platform, we also hoped to reap benefits of maintainability and adaptability. We wanted to hide the internal implementation of each architectural module, allowing clients to abstract from its unique characteristics. With this conception, component integrators would only have to concern with the object interface details.

The way in which COM achieves the referred abstraction is by forcing component developers to provide well defined interfaces that are separate from the implementation. This framework makes use of a specific interface definition language (IDL), which we used to define the interfaces of our platform and their respective methods. The information stored, with this common IDL representation, is subsequently registered in the COM library, which maintains all the knowledge about available classes in the system registry.

From the block diagram shown in figure 1, the functional units that ended up being implemented with the COM framework were: the Modality Server; the ASR engine; and the TTS engine. These functional modules were defined as COM components, capable of returning objects when necessary. The interaction between these components and their users is based on a client/server model. Because of this, the COM strategy leads to greater robustness, since it allows clients and servers to exist in different process
spaces. If a server process crashes or is otherwise disconnected from a client, the client can handle that problem gracefully and even restart the server if necessary.

In terms of disadvantages from the COM framework employment, we emphasize its complexity. This technology required a deep knowledge of its specification. In fact, we devoted much effort to the understanding of such concepts as: reference counting; IDL; global unique identifiers; thread marshalling; connection points; monikers, etc. Some of these concepts will be explained latter, as well as certain implementation details of our COM servers.
4.3 CONNECTING SPEECH ENGINES WITH APPLICATIONS

There is a primary deficiency with the industry that concerns almost all available speech engines: the shortage of standards for fast and economical application development. With this problem, it becomes vital to create a layer of software which stands between applications and speech engines, empowering them to communicate in a standardized way.

The following sections will describe our effort to achieve the referred technological integration.

4.3.1 The First Approach: Microsoft Speech API

Our original strategy to integrate the speech engines within target OS environments was completely sustained by an existing API: the Microsoft SAPI. We devoted a significant amount of time to understand this technology and to apply it into a functional system. However, we ended up abandoning this path mostly because of its lack of flexibility and the consequent inclusion of unnecessary complexity.

In this section, we will briefly introduce Microsoft SAPI and explain how we used it to incorporate speech into desktop applications. To finish we will also present the reasons that made us give up of this approach.

4.3.1.1 Introduction to Microsoft SAPI

The Microsoft SAPI is part of a software development kit for building speech engines and speech-enabled applications for the Windows OS. It provides a software layer that can be used by applications to communicate with ASR and TTS engines.

This middleware technology works on COM architecture, providing access to its functions in the form of COM interfaces. In order for a speech engine to become a Microsoft SAPI engine, it must implement at least one COM object representing the engine instance.

Concerning its structure, the API is separated into the following categories, which provide access to different engine attributes and procedures: audio; events; grammar compilation; lexicon; resources; speech recognition; and speech synthesis.

The services are provided in two distinct sets of software routines. The first set is supposed to be exploited by applications which need ASR or TTS services. On the other hand, the second set is related to the low-level communication with speech engines.

Together, all the referred interfaces and procedures, aim at facilitating the interaction between speech engines and applications, concealing the inner functioning of the engines.
4.3.1.2 Our Microsoft SAPI Implementation

As it is shown in figure 18, our foremost concern was the integration of the engines under SAPI's middleware. To achieve that goal, we implemented the essential device driver-level interfaces that cover Microsoft SAPI's minimum requirements.

![Block diagram of the original system using Microsoft SAPI.](image)

Figure 18 - Block diagram of the original system using Microsoft SAPI.

After registering the COM assemblies, related to our engines' device driver wrappers, users became able to select the engines for employment. Figure 19 displays the configuration of Microsoft SAPI to employ our own custom ASR engine.

![Control panel for choosing among available SAPI engines.](image)

Figure 19 - Control panel for choosing among available SAPI engines.
With the engine selected, it was straightly expected the availability of speech services, through our base technologies. In fact, since numerous applications were already prepared to benefit from Microsoft SAPI’s services, much work related to the development of speech interfaces was suppressed.

The main advantage of Microsoft SAPI over any other approach, relates to these standard characteristics. In this way, after porting our engines, we automatically guaranteed dictation functionalities on SAPI-compliant applications. Figure 20 shows a spoken interaction with one of these supported programs.

![Figure 20 - The creation of a document using Microsoft SAPI provided speech services.](image)

**4.3.1.3 Drawbacks of Microsoft SAPI**

Despite having achieved a functional platform, we were incapable to implement most of the device driver interface methods due to incompatibility issues. We actually ended up dropping the Microsoft SAPI approach, opting for another one that will be described in the next section of this thesis.

The following disadvantages express the reasons that made us give up of this strategy:

- **Unnecessary Complexity**: The API from the device driver interface reflects much complexity of speech technologies, which should be hided inside engines’ logic. Speech recognition, per instance, possesses more than nine interfaces.

- **Redundancy**: Microsoft SAPI tries to control a number of low-level aspects which are already taken care within the Audimus ASR system and the Dixi TTS system, such as:
  - Audio data reading and noise recognition.
  - Audio storage and serialization of results for later analysis.
  - Low-level grammar and lexicon loading.
  - Grammar editing and importation.
• Management of grammar state changes.
• Etc.

• Incompatibility: Concerning speech recognition grammars, the following incompatibilities exist between Microsoft SAPI and the Audimus ASR engine:
  ▪ SAPI does not fully support the definition of CFGs using the W3C Speech Recognition Grammar Specification, which is employed by Audimus.
  ▪ SAPI administers CFG grammar rules and state changes, and the engines are supposed poll SAPI for that knowledge. However, in the Audimus case, the management of the referred information has to be done internally.
  ▪ SAPI takes full control of loading grammars, notifying the engine about their content through various device driver interface methods. Conversely, in Audimus system, the engine itself is responsible for that task.

• Inflexibility: The fact of having to employ a proprietary grammar format, because of the mentioned incompatibility issues, brought the inconvenience of the system becoming not application independent. In this way, all major advantages of using the Microsoft SAPI standard were lost.

• Not Developer Friendly: Using Microsoft SAPI involved an immense effort to understand which interfaces and objects it implements, and which interfaces the engines should implement. The absent of quality documentation, the lack of developer support, and the existence of innumerous inaccessible execution routines in the SAPI middleware made this endeavor near impossible.

4.3.2 The Modality Server

With Microsoft SAPI put aside, we needed to delineate an alternative method to interconnect engine-level services and application-level clients. In the present section we are going to give an overview of our strategy.

As illustrated in figure 1, a considerable part of our software approach aims at fulfilling the gap between our base technologies and the OS applications. In fact, we provide an interface which represents a common protocol on which clients can access the speech services.

We present a modality server (MS) that is supposed to be used by speech-enabled programs so that they can communicate with our engines. In this section we will describe the characteristics of this component whose architecture is displayed in figure 21.
4.3.2.1 Server Technological Details

The MS is defined as a COM out-of-process server, which means its code executes in a process space different from the one of its clients. Being out-of-process, the MS’s objects can be accessed through an inter-process communication mechanism based on a lightweight version of RPC (see figure 22). Because of this fact, the exchanged information has to be pre-formatted and bundled, in a procedure referred as marshalling.

In our platform, the marshalling is consummated by proxy objects and a stub object. These entities handle the cross-process communication details for the MS’s interfaces. The stub is created in the MS’s server process, being responsible for managing the real interface pointers. On the other hand, the proxies stand in the clients’ processes and connect them to the stub, supplying them with the interface pointers.

Clients call MS’s interfaces through their proxies, which marshal the parameters and pass them to the server stub. The stub un-marshals the parameters and makes the actual function calls. When the functions return, the stub marshals the results and passes them to the proxy, which in turn returns them to the clients.
4.3.2.2 Middleware Characteristics

The MS can be viewed as a piece of middleware which implements an application-level interface (ALI), and an engine-level interface (ELI). Modality engines should use the ELI layer to become accessible to the applications, which are served by the ALI layer. This ALI dramatically diminishes the code overhead necessary for the integration of the speech technologies. It makes them more accessible and robust for a broad variety of programs, which become able to use simplified higher-level objects rather than directly call methods on the engines. On the other hand, this architectural design enables the plugging in of additional modality engines without the need to modify any of the applications. In fact, every one of these modality capabilities can perfectly be used separately from each other.

Both layers of the MS service are implemented as collection of COM interfaces. In this way, they possess a set of entry points which are accessible from a variety of programming languages. Applications can use the ALI to instantiate an engine, and afterwards users can adjust its characteristics. Certain features become conveniently available, like the ability to set and get engine attributes, turn the services on or off, and perform direct text-to-speech and speech recognition functions.

4.3.2.3 Two-Way Communication

During the development of the MS we had to contemplate the definition of outgoing interfaces to the clients. We were interested in returning engine-specific information such as: the sentences recognized during the ASR process; the word boundaries of the synthesized speech; etc.
In order to design the MS’s notification mechanism we followed the Command design pattern, using subsequently, the COM’s connectable objects for the actual implementation. With this approach we successfully enabled two-way communication between clients and the MS component.

The Command design pattern is usually applied when an application receives different requests and needs to handle them separately. It decouples the object that invoked the operation from the one having the knowledge to perform its execution. The pattern’s key element is the employment of an abstract interface that declares the notifications. Every class deriving from this abstract interface overrides its methods in order to handle each request in their own specialized way.

Following the Command pattern, the MS’s ALI includes an abstract event interface implementation which programs can conform to. In fact, the applications can subscribe to a broad variety of events which are generated by the recognition and synthesis engines while processing. For example, when the speech recognition engine perceives a word it fires a recognition event to indicate that an utterance has been detected. This information channels in the reverse direction, from the engines, through the runtime modality server, and on to the event sinks on the subscribed applications.

4.3.2.4 Management of Shared Engines

Our initial motivation with the MS was to share the speech services across multiple clients. We wanted to provide single engine objects that could be globally accessed. This approach would allow the sharing of resources, the removal of microphone contention, and the existence of a global interface provider for speech applications.

In order to implement the MS’s engine objects we followed the Singleton design pattern. In this way, clients communicating the MS would access the same single instances. On the other hand, with the MS running on an independent COM process we became able to make its services generically available.

The Singleton pattern is useful to ensure that there is only a single instance of a class. In fact, it specifies the direct control over how many objects can be created. The following code shows a typical implementation of this design pattern:

```cpp
class SingletonEngine
{

protected:
    SingletonEngine();

private:
    static SingletonEngine* _engine = 0;

public:
    static SingletonEngine* Instance()
    {

        if (_engine == 0)
            
            _engine = new SingletonEngine();

        return _engine;
    
};
```
As displayed above, the data member ‘_engine’ is meant to point to the one and only instance of its respective class. In this way, after its creation, clients will always be involved into taking or making requests to this pre-existent object.

With the singleton design concept in mind, we had to focus on its concrete implementation in our COM out-of-process server. Nevertheless, COM presented us with some obstacles concerning the definition of this pattern.

An important aspect in the COM technology is that objects do not possess an identity. In fact, every time an object is requested, a new instance is returned. The purpose of this policy is to allow for object pooling.

We needed clients to connect to the exact same engine instance in different points in time. With this said, we required a mechanism to simulate the identity of the MS’s COM instances. The solution found was the binding of monikers to the server’s objects, at their respective class factories.

Figure 23 exposes how the access to MS’s objects is processed, using the referred method.
As can be seen from above, monikers are objects that know how to create and initialize the content of single COM object instances. They can be asked to bind to the COM objects they represent, identifying them with unambiguous names.

With the use of monikers, the MS’s clients became able to acquire pointers to its running named objects. Implementing the singleton pattern at the object’s class factory, we restricted the existence of more than one named instance. In this way, we guaranteed the sharing of services from each particular engine.

### 4.3.2.5 Concurrency Control

Appropriate synchronization had to be applied during the implementation of methods provided by MS’s objects. In fact, since the MS was defined as a multithreaded server, we had to handle the concurrent access modification to the objects’ states.

We applied locking mechanisms to negotiate mutual exclusion among client threads. With these synchronization tools, we managed to develop a system capable of avoiding race conditions.

These locks became particularly useful on certain procedures, such as language model activation, which are continuously invoked. They serialized the access, by different clients, to these engine’s resources, preventing errors and inconsistencies.
4.4 ACCESSIBILITY AUGMENTATION: SIMULATING TASK INDEPENDENCY

Desktop application software can be characterized by its ample functional diversity. This multiplicity instigates the existence of a large-scale number of operational contexts. In fact, even a single application may possess numerous operating methods and interaction states, each one connoting to its own domain.

Because our shared ASR engine was supposed to stand simultaneously accessible in several distinct applications, a number of limitations have emerged from the task-independency problem, which was exposed in section 3.1.4.

We contemplated a strategy to deal with the mentioned issue, giving priority to the system’s robustness. The goal was to achieve reliable enough performances, in the actual context of the applications.

Our approach was based on the intelligent runtime administration of engine resources and configurations. We defined mechanisms to accurately manage these technologies, with the purpose of simulating task-independency. At the same time, we tried to minimize the memory and the processing load, needed to adequately control the engine’s possible states.

In the following sections we are going to describe the architectural decisions that allowed the accomplishment of the above delineated strategy.

4.4.1 Context Management

In order to handle functional domain multiplicity, we have created an architectural module which is responsible for context management and awareness. This module is represented in figure 1 by the name of Context Manager (CM).

The CM detains domain awareness on the desktop environment. This knowledge becomes essential to identify if the interaction with the current interface demands for dictation or command and control capabilities. Additionally, it also allows the correct configuration of each one of these engine operating modes.

This module acquires the necessary information to detect dictation occasions, and to choose the appropriate models for each one of them. On the other hand, it is responsible for detecting variations between the different domains, and for performing the consequent command and control grammar changes.

Finally, the CM also features screen reading capabilities and UI focus awareness, which combined together, provide the ability to dynamically create grammars that can represent unforeseen interaction opportunities. We will explain on section 4.4.2.2 how this dynamical process is done.
4.4.2 Language Model Administration and Functional Handling

With the goal of providing a vast range of interaction possibilities, we employed a diversified set of language models, ensuring their correct administration with the CM’s support. In this way, we were able to make the overall speech interface as complete as possible.

In respect to dictation, we have created specific models that were built from huge amounts of text data. In opportune occasions, the user is presented with the possibility to choose a subject for his dictation purposes. This subject is associated with a given language model which, in turn, is loaded on the engine at runtime.

Regarding the command and control application area, we implemented a hybrid approach that combines the usage of constant grammars, which can be generic or application specific, with dynamic generated ones. Figure 24 exposes this grammar sub-division.

![Diagram](image)

**Figure 24** - Classification of the platform’s spoken commands.

The handling of these commands is done in separate places, as can be seen in figure 1. The reactive module may receive the recognition results and, following a general supervision logic, perform the respective associated actions. By the other hand, particular application add-ins may also obtain the recognized spoken commands. These add-ins are responsible for specialized cover of certain application needs.

Concerning the commutation between the different handling routines, it is controlled by the MS’s application supporting module. This component queries the CM for focus information and then directs the MS’s recognition notifications to the proper client.

The next sections will describe more extensively this command handling distribution, which promotes simultaneously robustness and extensibility.
4.4.2.1 Base Spoken Interface: The Invariable Command Sets

Our platform employs a group of command and control grammars that remain invariable in time. They include the supporting commands of the overall spoken interface.

As displayed in figure 24, we divided these grammars in two subgroups, corresponding to: generic commands; and application-specific commands.

The generic grammars are always loaded when the platform initializes. They represent the commands that make sense in virtually any native application. In fact, they can be easily handled in a generic manner for every window.

The generic commands offer a reliable collection of functionalities that are always available. Still, if we were just to rely on them, important features would be lost. Users would experience extreme frustration when trying to interact with the computer only using speech recognition. Most importantly, specific command and control functionality would be extremely difficult to achieve.

Fortunately, with appropriate integration, most of that frustration was avoided. We employed application-specific grammars that contemplate particular requirements of certain UIs. In this way, we became able to provide precise and extensive accessibility support in non-standard environments.

4.4.2.2 Attaining Portability with a Dynamic Command List

While some characteristics of OS interfaces are invariable, many others can be customized by users, resulting in a huge disparity among the available interface elements of different environments. Several factors may influence this diversity, such as: the organization of files and directories; the variety of installed applications; the personalization of interfaces; etc.

We were interested to accomplish a system capable of adapting to unforeseen interaction opportunities. In this way, we had to consider some method to guarantee the consistency of unpredictable command and control alternatives.

With the above purposes in mind, we recurrently employed the accessibility technologies of the underlying operating systems. With their usage, we pretended to gather information concerning the available control options.

We used an accessibility API, called UI Automation, which provided programmatic access to most desktop interface elements. In this API, every piece of interface, such as a window or a folder, is represented by an object that supplies information about innumerable UI properties. By the other hand, the API also uses control patterns to categorize and expose the interface elements’ behavior.

All items covered by UI Automation are part of a tree structure in which the root element is the desktop. Through this root it was possible to navigate and obtain references to any interface child node. In figure 25 it is displayed a small fraction of the alluded tree.
We wanted to periodically access the Automation tree in order to locate potentially interesting interface controls. However, since this structure can comprise many thousands of nodes, we had to delineate an efficient search routine.

We implemented a tree-walker algorithm and defined a heuristic function to rank the alternatives at every branching step. The heuristic evaluation was based on the various UI properties of each node, conditioning the search to merely follow tree branches with potential interest for user interaction.

After ensuring the gather of crucial information about available UI, we had to develop mechanisms to correctly handle that knowledge. Figure 26 displays our overall strategy.

---

**Figure 25** - Control view of the UI Automation tree and properties of a specific UI element.

**Figure 26** - Dynamical creation of command grammars and storage of respective control patterns.
As shown in the above diagram, when an application is opened or switched, our system performs a query to determine the supported accessibility features, and then works through those. The platform locates the interface elements, taking a controlled amount of time, and dynamically produces the corresponding grammar.

The control patterns of the created commands are also stored, using an appropriate data structure. Actually, we employ a hash table using the UI elements’ names as keys. An example of its content can be seen in figure 27.

![Figure 27 - Hash table structure containing the interface control patterns.](image)

The Automation tree is an extremely variable structure. Parts of it are built as they are needed, and it can undergo changes as UI elements are added, moved, or removed. Because of this reason, we had to subscribe for focus-changing events in order to know when to update our dynamic command and control grammar.

With the generated grammar loaded at the ASR engine, users become able to access several interface elements via speech. The way in which our platform deals with these spoken commands is exposed in figure 28.

![Figure 28 - Command recognition followed by the invocation of the corresponding action.](image)
As displayed, the CM receives the recognized strings and transforms them into a hash table index, locating the corresponding control pattern. This operation is performed in amortized O(1) time, which makes the access to a huge hash table very efficient.

To finish, our platform employs the control pattern to programmatically manipulate the interactive item, automating the desktop environment.

4.4.2.3 Granting Extensibility: The Add-In Approach

While designing our system we took into consideration the need of future growth and the level of effort which would be necessary to implement extensions. We were interested in the continuous development of application level software responsible for specialized employment of the MS's services. The goal was to bridge the gap between what the modal technologies have to offer and what is needed for the many different applications to accomplish.

With the motivations referred above, we suggest an approach based on add-in integration, which brings parts together into a whole. Add-ins consist of computer programs able to interact with a host application with the objective of providing specific functionalities. They allowed us to potentiate interfaces, with spoken interaction, when this feature was unforeseen. On the other hand, this strategy gave us the freedom to separate our logic from the applications' one, allowing us to surpass the incapability to access the manufacturer's private source.

We contemplated an approach where the platform's functionality could be expanded, or modified, without the demand for the alteration or recompilation of the whole original code. With this idea in mind, we included in the MS's API, specific public interfaces accessible by add-in developers who don't have access to the system's source. The objective was to provide them with the necessary means to adjust the ECA's behavior and to extend overall speech accessibility.

The mentioned interfaces allow the binding of add-ins without the system having prior knowledge about them. On the other hand, our framework possesses mechanisms that guarantee seamless transitions between their distinct nonstandard domains. In fact, when applications with add-ins initialize, they register themselves on the MS, which subsequently provides the CM with all necessary information to handle these managed domains. Figure 29 gives an overall picture of this binding process.
As displayed above, after having obtained a pointer to the MS’s singleton object, the add-in binds to its notification mechanism. In this manner, two-way communication becomes enabled between them.

The application has then the opportunity to load engine-specific resources, such as acoustic or language models. These resources provide the necessary means to extend the spoken interface, adding, for example, extra recognizable command sets.

Finally, the application registers itself within the MS, providing information concerning its running process. This information is subsequently passed to the CM which processes and stores it for latter employment. In fact, this intelligence will allow commutating between all existing functional modules depending on the desktop’s current focus.
5.1 DESIGNING THE SYSTEM'S SPEECH INTERFACE

Speech interfaces possess some properties not usually encountered in most mature interface technologies:

- They often produce errors.
- Their state is often unclear to the user.

These characteristics impose the incorporation of a number of specific features into the UI.

The present chapter will expose the principles which conducted the planning of our platform's generic speech interface. Their aim was guiding decisions and criteria regarding the maximization of robustness and usability. These general guidelines are supported by several researches, which presented testing results contemplating several measurable objectives (MANE, A. et al., 1996)(YANKELOVICH, N. and Lai, J., 1999 )(LAI, J. and Yankelovich, N., 2002). They should be followed by a wider range of speech interfaces in order to make them more useful.

In the following sections we present the strategies that were followed to fulfill to the various interface necessities.

5.1.1 Feedback

Feedback is generally used to survey the behavior of a system. This feature becomes convenient, for example, when users are attempting to interact with the speech system, but nothing happens. In this particular case it may not be clear to the user what might be the problem.

In order to prevent these situations, well-designed speech interfaces should intelligibly and constantly display their state to the user (SCHURICK, J. M. et al., 1985). Strategies that were pursued in order to address this issue include:
• **Providing graphic state information.**

The speech recognition’s state is constantly displayed to the users. This modality may possess up to four different states, which are: ready; blocked; listening; and recognizing. However, in some situations we only present two states (blocked and listening) which are absolutely necessary to keep the user informed.

The state indicators were put in a space where they do not conflict with the current task and, at the same time, where users are able to continuously monitor them. For these reasons they were localized on a gadget which can be moved outside or to the frontier of the active task region (see figure 30).

With the objective to facilitate the perception of a status alteration, we adopted a color convention. In fact, intuitiveness was supported by redundancy in respect to both location and color. Polychrome coding was preferred in order to raise the visual distinctiveness of the state displays.

![State display and VU meter](image)

**Figure 30** – Controller gadget which returns feedback about the ASR state and the spoken input.

• **Providing information about the speech input.**

We provide feedback related to the spoken input parameters. A VU (volume unit) meter is used, allowing continuous supervision of whether the platform is live. In fact, there is always some reaction to any audible input.

The existence of this continual sound-level information helps users to accurately shape their vocal behavior, which, in turn, results in more successful recognition.

Additionally, the ability to monitor the background noise facilitates the determination of whether the environment is adequate for speech recognition. It also suggests a possible reason why the platform may be malfunctioning.

• **Providing auditory feedback.**

We employ auditory feedback to complement the graphic information. This feedback mode becomes very useful when graphic response is impracticable. In this way, events like certain state changes or key recognitions are notified by easily differentiated tones.
The interpretation of the auditory information could result in some confusion for the inexperienced users. Therefore we limited the diversity of auditory signals to make their meaning easy to distinguish from their context of occurrence.

On the other hand, the auditory feedback was defined brief in order to not distract users from the task at hand.

- **Providing feedback about long blocked intervals.**

Certain tasks related to speech technology involve considerable processing and allocation of resources. This normally translates in long blocked intervals which may lead users to feel that some problem has occurred. To prevent this from happening, we made the users aware of these situations, whenever is possible to predict them.

We employed graphic feedback, such as progress bars or loading messages, in order to keep the users informed (see figure 31).

![Loading... Please Wait](image)

**Figure 31** – Interface loading message.

- **Elucidating users about what can be spoken.**

While interacting with speech applications, inexperienced users commonly bring assumptions based on the traits of human-human communication. However, speech systems only possess knowledge about a restricted language and therefore will not be able to interpret expressions which are not included in the currently active vocabulary. In this context, inexperienced users need to learn the interface in order to have much less difficulty staying within the bounds of the language. They should be guided into the acceptable language through several methods.
One employed strategy, concerning the mentioned problematic, was providing hints about the system's expectation at specific times. Actually, the use of prompts, phrased in certain ways, supplied guidance on expected inputs (e.g., “yes” or “no”).

Another way to deal with this problem was the on-screen display of all the legal utterances.

- **Not distracting users from task-related activities.**
  Status displays should be rapidly interpretable, without taking away the users’ attention from their actual task. With this in mind, peripheral displays were chosen over small displays which require explicit attention to be noticed. On the other hand, feedback was defined to be provided, in some situations, by separate modes, such as sound.

- **Not confusing users with excessive feedback.**
  Despite the fact that feedback is necessary, too much information can become overwhelming. Actually, feedback displays shouldn’t require users to search through them to identify the relevant information for their performance.

  In this way, certain information about recognition results, like decoding scores or utterance durations, was hidden for the application users. In fact, all the provided feedback clearly promotes the increase of interface usability.

### 5.1.2 Interface Input

The interface input supplies the necessary means to insert information and manipulate applications. Speech system developers should pay attention to certain aspects while designing how the access, through the interface, is processed. In this way, the following rules were considered:

- **Allowing continuous speech control.**
  Users generally possess a mediocre capability to coordinate speech with button presses. Due to their unfamiliarity with this skill, people may feel uncomfortable when having to combine these two forms of interaction. Furthermore, interface-specific manipulation may detract from the task-specific activity. For all these reasons, robust continuous listening interactions were always preferred to ones that require explicit user action.

- **Using toggle interface input items.**
  Interface elements that involve high cost tasks, like the allocation of engine resources, should clearly display their current state to the users. This is why push-and-hold protocols were preferred to two-click protocols. In the first case, the states are mechanically mingled while in the latter case users might not be well aware of what state the system is in.
5.1.3 Error Recovery

Speech systems are likely to produce recognition errors sometime in the foreseeable future. For this reason, there should always be mechanisms designed to handle the occurrence of those errors and provide some way to recover from them. We list some practices which were followed:

- **Providing the possibility to undo an action.**

  The undo capability can be perceived as the disconfirmation of an input. In fact, this is a feature of the applications for which speech serves as interface. In these applications, some states may not be reversible while others may require elaborate action history management.

  In certain situations, such as stateless applications, the ability to undo may not be required. In these cases, the cost of a single transaction regulates the usability of that interface.

- **Using a confirmation protocol when convenient.**

  Undoing an action may not be practical at all times. When the cost of a single operation is too high we employ a confirmation protocol which insures the users' intents.

  This input verification is done by an additional utterance, such as “Ok” or “Cancel”, after the examination of a candidate recognition result. The speech recognition accuracy is very high during this validation phase. In fact, the engine’s decoder is in a sub-state that guarantees correctness while confirming or disconfirming.

  In some situations, a different approach to validate recognized commands is used. We employ an action in some other mode, such as a button press. Despite the fact that it is less desirable, this alternative offers much more reliability.

5.1.4 Error Correction

Error correction should be supported by all applications that allow dictation (BABER, C. et al., 1990). This task involves the creation of sentences using speech, and, therefore, has to deal with a certain degree of recognition error.

Besides being embedded in the spoken interface, the error correction should also make use of the mechanisms provided by underlying applications. Strategies that were pursued to support this feature include:

- **Providing alternative input methods.**

  Users are never put in a situation where they can only rely on spoken interaction. An alternative input method which overrides speech is always available. Only in this way, are users constantly able to regain control, either to reset the system’s state or to introduce information by an alternate mode.
• **Allowing the selection between alternative recognition results.**

In ambiguous situations, users are able to choose among alternative decoding results. Although the recognition engine normally acts immediately on the best scoring hypotheses, it can also provide this kind of information. The alternatives being shown correspond to close higher scoring results.

Similarly to confirmation protocols, high accuracy is guaranteed during alternative choosing. In order to ensure a quick convergence to the right decoding result, a controlled selection mode is provided. We employ constrained language models and optional interface elements, such as menus.

• **Providing an interactive recovery procedure.**

Speech interfaces should assist users to identify and correct errors, which emerge from inaccurate recognition, with the objective of making error recovery more efficient.

With this purpose, we defined innumerable spoken commands to correct dictation errors. These commands will be described ahead.

### 5.1.5 Application Integration

Integrating speech interfaces into existing applications is not a simple assignment since every scenario is unique. This task requires a great amount of experience with spoken interfaces and with the actual application domain. We present the practices that were employed to deal with this issue:

• **Handling elementary applications with a generic approach.**

Simple applications use a speech interface that does not need to track their sub-domains. This interface is generic, being used in all applications that allow addressing through emulation mechanisms.

• **Providing specific mechanisms to deal with more complex applications.**

The complex applications are directly accessed, through their functional level, in order to accomplish the speech input commands. Despite all the additional work, this approach reveals much more efficiency and potentiality than the emulation of an existing generic interface.
5.2 THE SUPERVISED APPLICATIONS

We were interested on specific integration of the MS’s services, within the context of word processing and e-mail applications. Therefore, much effort was devoted on that direction by the creating add-ins for such applications as Microsoft Word 2007 and Microsoft Outlook 2007.

Add-in development is not a simple task. There is no generic add-in integration package because every scenario is unique. In fact it was required to analyze each specific application scenario in order to estimate the cost of integration. This task demanded for considerable knowledge about the actual application domains.

We produced complementary command sets to make the speech recognition more helpful on the referred environments and also defined additional ECA behavior. Nevertheless, incorporating speech technology was not just about plain application control and dictation capabilities. The ability to enter data into templates or custom applications efficiently was also taken in account. Highly attention was paid to the integration of speech into the user’s workflow.

With the deployment of add-ins, we have also made the platform’s interface more practical by creating a set of customized shortcuts to easily operate the features of overall system. Figure 32 displays some of these shortcuts which allow changing the platform’s state, and choosing the following features:

- The ASR’s language model for dictation purposes.
- The ASR’s acoustic model.
- The voice to be used by the synthesizer.

![Figure 32 – Interfaces for the selection of the: a) ASR models; b) TTS voice.](image)

In the following sections we will describe the interaction’s singularities on each one of the referred supervised applications.

5.2.1 Word Processing Application

Figure 33 shows a user interaction in the context of one of the platform’s supervised applications: the Microsoft Word 2007.
A group of spoken commands, which are listed in appendix B, enable users to naturally control this application and complete innumerable tasks, such as: formatting documents; efficiently fixing incorrectly recognized words; copying and deleting text; etc.

Document management is also an important part of the referred command set, providing functionalities like opening, saving or transitioning between documents.

Still, most attention has been given to the dictation task. In order to allow a quick creation of documents, the recognition results are extensively processed, and the punctuation, capitalization, and spacing issues are automatically resolved. Users may actually define which words should be translated into symbols during dictation.

![Screenshot of dictation interface](image.png)

**Figure 33** – The dictation of a document using our system.

### 5.2.2 E-Mail Application

In the same way as with Word, we have developed specific logic to adapt the MS’s services to an e-mail client application: the Microsoft Outlook 2007.

A number of commands have been associated with customized interfaces towards the definition of efficient workflows. We were particularly interested in the accomplishment of key tasks related to the creation and management of e-mails.

We added commands which allow navigating between the various panes of the application and selecting the appropriate options. With this support, users may, for example, select, open, delete, or reply to messages, without needing to use the mouse or keyboard.

Figure 34 exposes one of the many possible interactions with the system.
Concerning the dictation of messages, the strategy that was adopted is very similar to the one which was used within Word. In fact, since the interoperability APIs provided by Outlook and Word are analogous, the text processing routines could be reused.

All the commands supported by Outlook 2007 can be viewed on appendix B.

Figure 34 – The creation of an email using our system.
5.3 **The Interaction’s Characteristics**

5.3.1 **Interaction’s Overview**

Once the system launches, users are prompted with the gadget interface that can be seen in figure 30. From this point on, the spoken interaction can be started or interrupted, through the gadget, according to the users’ needs.

With command and control capabilities, the speakers are able to give orders to the applications. However, when dictating, the speakers’ input is not interpreted and is exclusively converted to text format. Figure 32 shows the interface which allows the selection of the dictation task model, on a supervised application.

The system provides automatically a generic set of commands which allows the navigation through the desktop environment (see appendix A). Users can say, for example, something like ‘open calculator’ or ‘switch to word’ to alternate between applications.

They can also say commands like ‘recycle bin’ in order to access a specific interface element. These elements are maintained on the dynamically generated command set, which is updated on a regular basis. When speech is used to open or switch to an application, the system queries it, to determine which accessibility features it supports, and then works through those.

Certain applications possess add-ins which hold extra command sets and the respective accessibility support (see appendix B). Microsoft Word and Outlook are examples of these supervised applications which also allow dictation. In these particular situations the AUDIMUS engine maintains four active language models regarding: the generic constant commands; the dynamically generated commands; the application specific commands; and the dictation language model. The engine is responsible for choosing the results, according to the model which returns the most statistical valid recognition.

In particular situations the ECA engages users in a dialog to complete certain key tasks, like the creation of emails. At those times the system displays the animated face which provides the agent’s visual feedback.

5.3.2 **Use Case: Creating an E-mail**

This section demonstrates a very simple scenario on which an e-mail is dictated and subsequently formatted through command and control functionality. The ECA assists the user on the process of creating the e-mail, reading it at the end to review the message’s content. Finally, the e-mail is sent to the respective recipient.

- **Task Overview**

The creation of e-mail messages is one of the most common scenarios while interacting with our platform. This task usually involves the accomplishment of the following steps:
• Starting the e-mail client application.
• Creating the message.
• Dictating and formatting the text.
• Synthesizing the text, with an artificial voice, in order to review its contents (optional).
• Sending the e-mail message.
• Closing the e-mail client application.

**Task’s Steps**

**Start the application:**
1) Execute our system’s main process (this will make the gadget control show up).
2) Start the recognition.
3) Say “abrir outlook”.

**Creating, dictation, and formatting the message:**
4) Say “escrever correio electrónico” (this will make the ECA show up, and engage in a dialogue).
5) Insert the receiver’s address, which will be asked by the agent.
6) Say “próximo campo”, in order to select the text box referent to the email’s subject.
7) Insert the e-mail’s subject.
8) Say “próximo campo”, in order to select the text box referent to the message’s body.
9) Say “olá vírgula parágrafo esta mensagem é um teste ao sistema de ditado ponto ponto ponto ponto”.
   **Note:** It is necessary to pronounce the punctuation.
10) In order to correct or modify the message’s body, say “selecionar palavra anterior” or any other command that allows selecting the text block to be changed. Finally, dictate the desired correction.
11) To format the text, it is possible to give commands such as “formatar parágrafo justificado” or “formatar tamanho vinte”, between many others.

**Synthesizing the message’s content:**
12) Say “sintetizar mensagem”.

**Sending the e-mail and closing the application:**
13) Say “enviar mensagem”.
14) Say “sair”.


6.1 HEURISTIC EVALUATION

A heuristic evaluation was conducted in order to find the usability problems of our UI design. In this way, the interface issues were attended as part of an iterative design process.

A group of four evaluators was used. They individually examined the interface and judged its compliance with recognized usability principles. Each heuristic evaluation session lasted about one hour.

Since the evaluators were fairly naive with respect to the system’s characteristics, it was necessary to assist them to use the interface. However, hints were not given until they were clearly in trouble regarding usability problems.

The evaluators’ actions and verbal comments were interpreted while they were interacting with the interface. From their reactions we inferred potential usability issues of our system.

The heuristic evaluation finished with a debriefing session which provided us with additional design advices. This session was carried out on brainstorming mode and focused on possible interface redesigns.

6.1.1 Heuristic Evaluation Results

In this section we list the usability problems that were found during the various iterations of the heuristic evaluation. We also present the solutions that were proposed to deal with the identified problems.

Most of the referred solutions were implemented in the most recent prototype of our system. Because the heuristic evaluation aims at explaining each observed problem with reference to established principles, it was fairly easy to generate a revised design.

The usability issues were divided into ten principles for UI design, the so-called heuristics, and are exposed in the following table:
<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Usability Problem</th>
<th>Proposed Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility of system status</td>
<td>The speech recognition status is not clear in the gadget interface.</td>
<td>Creating a toggle button with shifting colors that reflect the speech recognition status.</td>
</tr>
<tr>
<td></td>
<td>The system does not inform users about what is going on after a command is spoken.</td>
<td>Creating a feedback window that shows all the recognized commands in order.</td>
</tr>
<tr>
<td></td>
<td>While dictating, the text is only written when the speaker makes a pause.</td>
<td>Enabling the return of hypothesis, by the ASR engine, and showing them while users dictate.</td>
</tr>
<tr>
<td></td>
<td>The users are unaware of what is going on while the system is loading resources.</td>
<td>Showing a splash screen that keeps users informed about loading periods.</td>
</tr>
<tr>
<td>Match between system and the real world</td>
<td></td>
<td><em>No violations were found for this heuristic.</em></td>
</tr>
<tr>
<td>User control and freedom</td>
<td>The user has no control about the models that are activated when the system starts.</td>
<td>Creating a dialog box that allows users to choose the default models.</td>
</tr>
<tr>
<td></td>
<td>In order to change an acoustic or language model, users must go to a supervised application (e.g. Word). If no supervised application is installed, users have no control over the models.</td>
<td>Creating context menus, on the notify icon, that allow users to choose the models on any occasion.</td>
</tr>
<tr>
<td>Consistency and standards</td>
<td>The text messages are not consistent in respect to language. Some appear in Portuguese and others in English.</td>
<td>Employing the Portuguese language for all text messages.</td>
</tr>
<tr>
<td></td>
<td>The spoken command that saves a document ('guardar') is different from the one that saves a message ('salvar').</td>
<td>Allowing both spoken commands in all appropriate occasions.</td>
</tr>
<tr>
<td>Error prevention</td>
<td></td>
<td><em>No violations were found for this heuristic.</em></td>
</tr>
<tr>
<td>Recognition rather than recall</td>
<td></td>
<td><em>No violations were found for this heuristic.</em></td>
</tr>
<tr>
<td>Flexibility and efficiency of use</td>
<td>To achieve certain goals, an oral command needs to be repeated several times in a row. E.g., to delete five words the user needs to speak the 'apagar palavra' command five times.</td>
<td>Complementing commands with numeric variables that specify repetitive behavior. E.g., 'refazer quatro vezes', 'apagar duas palavras', etc.</td>
</tr>
<tr>
<td></td>
<td>Some frequent tasks involve too many spoken commands. E.g. to activate another opened document the user must say at least three commands.</td>
<td>Defining new commands that speed up the interaction for the expert user. E.g., 'alternar entre documentos'.</td>
</tr>
<tr>
<td></td>
<td>In order to specify the recipient of an email message the user needs to know the email address or search throughout the address book.</td>
<td>Defining a command for each name on the address book to directly insert the related email address.</td>
</tr>
<tr>
<td><strong>Aesthetic and minimalist design</strong></td>
<td><strong>Help users to recognize and recover from errors</strong></td>
<td><strong>Help and documentation</strong></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>No violations were found for this heuristic.</td>
<td>Several error messages are expressed in engines error codes which have no meaning for common users.</td>
<td>Creating text messages, in plain language, for each existent error code.</td>
</tr>
<tr>
<td></td>
<td>Some error messages are inconclusive, not guiding users towards a possible solution. E.g. ‘Invalid config file’.</td>
<td>Creating messages that are more specific towards the problem. E.g. ‘Invalid config file: Wrong license’.</td>
</tr>
<tr>
<td></td>
<td>There’s no way to be acquainted with the available spoken commands, besides reading the user manual.</td>
<td>Defining a ‘what can I say’ command which shows the list of all available commands.</td>
</tr>
<tr>
<td></td>
<td>The user manual lists all the available commands but does not specify the actual steps to be carried out.</td>
<td>Including, in the user manual, the ordered list of actions necessary to complete a given task.</td>
</tr>
</tbody>
</table>

**Table 1** – The heuristic evaluation’s results.
6.2 **Evaluation with Users**

Besides being analyzed by experts, the system was also evaluated with common users. A field study was conducted to test if usability had been impacted, either positively or negatively. With that goal, a couple of tasks were defined as well as a number usability measures. We then observed users completing those tasks, using either our system or traditional UIs, and compared the results.

Findings indicate significant usability improvements between the exclusive usage of our software and the employment of traditional UIs.

In this section we will present and discuss the methodology of the field study. We will also expose its results and explain how challenges associated with the field evaluation may have impacted the findings.

6.2.1 **Methodology**

- **Subjects:**
  The measures were collected from a total of 10 participants. Half of the participants were female and the other half were male. The age range was between 21 and 51, with an average of 29 years old.

- **Instruments and settings:**
  - Laptop computer;
  - Intel Centrino Duo, 2.00 Gb of RAM;
  - Windows Vista;

- **Procedure:**
  We defined two tasks including control commands, dictation, navigation, and interaction with the ECA.
  We then delineated usability specifications for performance with regard to the error rate and time on task.
  Before the evaluation, users became familiar with the available spoken commands in the course of a training session.
  Finally, we spent time observing users and registering the usability measures. We also collected subjective user reactions with a questionnaire using a Likert scale.

6.2.2 **Tasks**

- **Task 1: Creating and formatting a document.**
  1) Run the Microsoft Word application.
  2) Create a document.
3) Transcript / dictate a given text containing 230 words and punctuation.
4) Format the text in a specified way.
5) Save the document.
6) Exit the application.

- **Task 2: Creating and sending an e-mail**.
  1) Run the Microsoft Outlook application.
  2) Create an e-mail.
  3) Fill the e-mail’s header.
  4) Transcript / dictate a given message containing 105 words and punctuation.
  5) Send the e-mail.
  6) Exit the application.

6.2.3 Tests and Usability Measures

The following tables expose the registered usability measures for each task. The presented values correspond to the averages and standard deviations from the results of all users.

The significances were determined using the T-test assuming unequal sample variances. The P-values above 0.05 were considered statistically insignificant, indicating no difference between the groups.

- **Task 1**:

<table>
<thead>
<tr>
<th>Usability Measures</th>
<th>Our System</th>
<th>Traditional UIs</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arithmetic mean</td>
<td>Standard deviation</td>
<td>Arithmetic mean</td>
</tr>
<tr>
<td>Number of Deletions</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Number of Corrections</td>
<td>12</td>
<td>10</td>
<td>17</td>
</tr>
<tr>
<td>Transcription Time (seconds)</td>
<td>327</td>
<td>151</td>
<td>618</td>
</tr>
<tr>
<td>Formatting Time (seconds)</td>
<td>36</td>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>Total Completion Time (seconds)</td>
<td>380</td>
<td>148</td>
<td>655</td>
</tr>
</tbody>
</table>

Table 2 – The usability measures for the document creation task.
• **Task 2:**

<table>
<thead>
<tr>
<th>Usability Measures</th>
<th>Our System</th>
<th>Traditional UIs</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arithmetic mean</td>
<td>Standard deviation</td>
<td>Arithmetic mean</td>
</tr>
<tr>
<td>Number of Deletions</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Number of Corrections</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Transcription Time (seconds)</td>
<td>121</td>
<td>27</td>
<td>292</td>
</tr>
<tr>
<td>Editing Time (seconds)</td>
<td>41</td>
<td>5</td>
<td>36</td>
</tr>
<tr>
<td>Total Completion Time (seconds)</td>
<td>180</td>
<td>30</td>
<td>340</td>
</tr>
</tbody>
</table>

*Table 3 – The usability measures for the e‐mail creation task.*

• **Questionnaire:**
  - ✓ 80% of the users rated the experience a four or five on a one to five scale where five is the best.
  - ✓ For both tasks, 70% of the users found our system better in terms of efficiency and ease of use.
  - ✓ 100% of the users expressed the desire to use our system in the future.

### 6.2.4 Discussion

Time on task was significantly higher with our system than with traditional interfaces. This is mostly explained because of the superior efficiency of dictation over text transcription.

On the other hand, our platform showed a slightly lower error rate, however, no significant differences were found at this level.

We compared error rates, time to complete task, and subjective responses against targets in our usability specification. For 70% of the tasks, the subjective rating was higher than the observed performance, indicating that users did not perceive the errors and longer times as problematic. One explanation may be that we underestimated the target goals in the usability specification. Another explanation could be that users did not mind making mistakes since the experience of interacting with the character is fun enough to counterbalance the longer time spent on the tasks.

### 6.2.5 Evaluation’s Conclusions

Our findings suggest that our platform compares favorably to traditional UIs, however, some caution must be taken while interpreting the results. In fact, the conducted field study is limited by its reduced sample size and testing time.

Nevertheless, the tests that were made revealed important critical factors affecting the system’s usability: speech recognition accuracy; program reliability; ease of error correction; and environmental noise level.
7.1 CONCLUSIONS

The work described throughout this thesis covered, with success, most of the objectives that were proposed.

The major goal was the technological integration of our speech technologies within Windows environment, on which we suggested a middleware approach that was successfully implemented. This strategy dealt with a range of issues that underlie the integration of speech engines, particularly, such problems as task independency and resource distribution.

With the base technologies available through a well-defined structure, we focused on the practical challenges of designing speech based interfaces. We created and deployed interfaces which provide access to the various possibilities within the OS and its applications. Additionally, we defined custom commands and handling routines that supply additional specific features to word processing and e-mail applications, such as dictation and screen-reading capabilities.

Finally, the interface has been successfully empowered with ECA interactive capabilities, allowing the system to capitalize from the advantages of face-to-face dialogue and proactive assistance.

The system has been submitted to a usability evaluation process and the feedback we have been receiving from users has been substantially positive. People prefer to use their speech over traditional interfaces in tasks which involved dictation, due to its naturalness and efficiency. On the other hand, command and control tasks were proved to be done with more effectiveness when speech interaction takes place alongside direct manipulation interfaces.

Through this work, we have expressed one vision of what speech interaction between humans and computers may be like in the foreseeable future. In fact, the resulting platform possesses characteristics which could lead to its massive employment in future OS interfaces.

In respect to interface designing it was proved, above all, that as the application of the ECA interface expands, it is crucial that we work closely with users. Only in this way, we will be able to understand the variety of features that match the users’ preferences in respect to their perception of the interface.
7.2 Future Work

Future work on this project might involve a great amount of endeavors since the deployment of the modality engines can create innumerable interaction opportunities.

A great amount of effort could regard the expansion of the interaction possibilities and the maximization of the usability. The goal would be the reduction of any kind of frustration experienced while interacting with the system.

Also, the experience with email and word processing applications could, per instance, motivate the expansion of the supervised interfaces to more challenging areas, where information is difficult to supervise, such as web browsers.

On the other hand, the ECA behavior could equally be expanded in order to contemplate new situations where users might need proactive help.

Regarding the speech engines, a number of improvements could also be presented in the future:

- Employment of language models correlated with subjects specific to e-mails, including a generic model.
- Development of a mechanism which would allow the speakers to adapt the ASR engine to the characteristics of their voice, consequently improving the speech recognition.
- Creation of a wizard that would permit users to add or edit words from the available language models.
- Improve the efficiency of the speech engines and of the acoustic / language models.


APPENDIX A: INVARIABLE GENERIC COMMANDS

In this appendix we present the list of generic commands supported by the system. These commands are constantly available in every single application domain.

<table>
<thead>
<tr>
<th>Navigation through the Desktop</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Navegar entre janelas”</td>
</tr>
<tr>
<td>“Ambiente de trabalho”</td>
</tr>
<tr>
<td>“O meu computador”</td>
</tr>
<tr>
<td>“Localizar arquivo”</td>
</tr>
<tr>
<td>“Subir”</td>
</tr>
<tr>
<td>“Descer”</td>
</tr>
<tr>
<td>“Esquerda”</td>
</tr>
<tr>
<td>“Direita”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Menu Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Activar barra de menus”</td>
</tr>
<tr>
<td>“Menu iniciar”</td>
</tr>
<tr>
<td>“Menu da janela”</td>
</tr>
<tr>
<td>“Menu do item”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Confirmar” or “Ok”</td>
</tr>
<tr>
<td>“Cancelar”</td>
</tr>
<tr>
<td>“Anterior”</td>
</tr>
<tr>
<td>“Seguinte”</td>
</tr>
<tr>
<td>Open/Close Applications</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>“Abrir Paint”</td>
</tr>
<tr>
<td>“Abrir Calculadora”</td>
</tr>
<tr>
<td>“Abrir Notepad”</td>
</tr>
<tr>
<td>“Abrir Wordpad”</td>
</tr>
<tr>
<td>“Abrir Word”</td>
</tr>
<tr>
<td>“Abrir Outlook”</td>
</tr>
<tr>
<td>“Sair” or “Fechar janela”</td>
</tr>
</tbody>
</table>
APPENDIX B: INVARIABLE APPLICATION-SPECIFIC COMMANDS

In this appendix we present the list of application-specific commands supported by the system. These commands are only accessible from contexts or sub-contexts inherent to particular application domains. Their availability is conditioned by the deployment of the respective add-ins.

- Microsoft Word 2007

<table>
<thead>
<tr>
<th>Document Management</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“Novo documento”</td>
<td>Opens a new document</td>
</tr>
<tr>
<td>“Abrir documento”</td>
<td>Opens a saved document</td>
</tr>
<tr>
<td>“Guardar como”</td>
<td>Saves the document on a specific folder</td>
</tr>
<tr>
<td>“(Salvar/Guardar) documento”</td>
<td>Saves the document</td>
</tr>
<tr>
<td>“Fechar documento”</td>
<td>Closes the document</td>
</tr>
<tr>
<td>“Alternar entre documentos”</td>
<td>Alternates between opened documents</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Document Review</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“Desfazer”</td>
<td>Undoes an action</td>
</tr>
<tr>
<td>“Refazer”</td>
<td>Redoes or repeats an action</td>
</tr>
<tr>
<td>“Verificar a ortografia”</td>
<td>Checks the spelling and grammar of the document’s text</td>
</tr>
<tr>
<td><strong>Switching View</strong></td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;Vista esquema de impressão&quot;</td>
<td>Alternates between the document’s views</td>
</tr>
<tr>
<td>&quot;Vista de destaques&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Vista de rascunho&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Navigation through the Document</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;(Saltar/Transitar) para princípio do documento&quot;</td>
<td>Moves the cursor to a specific place on the document</td>
</tr>
<tr>
<td>&quot;(Saltar/Transitar) para fim do documento&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(Saltar/Transitar) para princípio do parágrafo&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(Saltar/Transitar) para fim do parágrafo&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(Saltar/Transitar) para princípio da linha&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(Saltar/Transitar) para fim da linha&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(Saltar/Transitar) para princípio da palavra&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(Saltar/Transitar) para fim da palavra&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Caracter anterior&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Caracter seguinte&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Text Deletion</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Apagar&quot;</td>
<td>Deletes the previous character</td>
</tr>
<tr>
<td>&quot;Apagar à direita&quot;</td>
<td>Deletes the next character</td>
</tr>
<tr>
<td>&quot;Apagar palavra à esquerda&quot;</td>
<td>Deletes the previous word</td>
</tr>
<tr>
<td>&quot;Apagar palavra à direita&quot;</td>
<td>Deletes the next word</td>
</tr>
</tbody>
</table>
### Text Selection

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Seleccionar (tudo/documento)&quot;</td>
<td>Selects all text</td>
</tr>
<tr>
<td>&quot;Seleccionar parágrafo anterior&quot;</td>
<td>Selects previous paragraph</td>
</tr>
<tr>
<td>&quot;Seleccionar parágrafo&quot;</td>
<td>Selects current paragraph</td>
</tr>
<tr>
<td>&quot;Seleccionar parágrafo seguinte&quot;</td>
<td>Selects next paragraph</td>
</tr>
<tr>
<td>&quot;Seleccionar linha anterior&quot;</td>
<td>Selects previous line</td>
</tr>
<tr>
<td>&quot;Seleccionar linha&quot;</td>
<td>Selects current line</td>
</tr>
<tr>
<td>&quot;Seleccionar linha seguinte&quot;</td>
<td>Selects next line</td>
</tr>
<tr>
<td>&quot;Seleccionar palavra anterior&quot;</td>
<td>Selects previous word</td>
</tr>
<tr>
<td>&quot;Seleccionar palavra&quot;</td>
<td>Selects current word</td>
</tr>
<tr>
<td>&quot;Seleccionar palavra seguinte&quot;</td>
<td>Selects next word</td>
</tr>
<tr>
<td>&quot;Seleccionar caracter anterior&quot;</td>
<td>Selects previous character</td>
</tr>
<tr>
<td>&quot;Seleccionar caracter seguinte&quot;</td>
<td>Selects next character</td>
</tr>
</tbody>
</table>

Selects text blocks from the document

### Copying and Moving Text

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Mover parágrafo para cima&quot;</td>
<td>Moves the current paragraph up</td>
</tr>
<tr>
<td>&quot;Mover parágrafo para baixo&quot;</td>
<td>Moves the current paragraph down</td>
</tr>
<tr>
<td>&quot;Cortar texto&quot;</td>
<td>Cuts the selected text</td>
</tr>
<tr>
<td>&quot;Copiar texto&quot;</td>
<td>Copies the selected text</td>
</tr>
<tr>
<td>&quot;Colar texto&quot;</td>
<td>Pastes the selected text</td>
</tr>
</tbody>
</table>

### Text Alignment

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Aumentar avanço&quot;</td>
<td>Creates a hanging indent</td>
</tr>
<tr>
<td>&quot;Diminuir avanço&quot;</td>
<td>Reduces a hanging indent</td>
</tr>
<tr>
<td>&quot;Adicionar marcas&quot;</td>
<td>Applies bullets to the current paragraph</td>
</tr>
<tr>
<td><strong>Text Formatting</strong></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>“Formatar fonte”</td>
<td>Opens the Font dialog box</td>
</tr>
<tr>
<td>“Formatar fonte verdana”</td>
<td></td>
</tr>
<tr>
<td>“Formatar fonte times”</td>
<td>Changes the font of the selected text</td>
</tr>
<tr>
<td>“Formatar fonte arial”</td>
<td></td>
</tr>
<tr>
<td>“Diminuir tamanho da fonte”</td>
<td>Changes the font size of the selected text</td>
</tr>
<tr>
<td>“Aumentar tamanho da fonte”</td>
<td></td>
</tr>
<tr>
<td>“Formatar tamanho (seis - trinta)”</td>
<td></td>
</tr>
<tr>
<td>“Bold” or “Negrito”</td>
<td>Applies bold formatting</td>
</tr>
<tr>
<td>“Itálico”</td>
<td>Applies italic formatting</td>
</tr>
<tr>
<td>“Sublinhado”</td>
<td>Applies an underline</td>
</tr>
<tr>
<td>“Maiúsculas” or “Capitalizar”</td>
<td>Changes the case of letters</td>
</tr>
<tr>
<td>“Limpar formatação”</td>
<td>Removes all manual formatting</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Paragraph Formatting</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“Formatar parágrafo”</td>
<td>Opens the Paragraph dialog box</td>
</tr>
<tr>
<td>“Formatar parágrafo centro”</td>
<td>Aligns the selected paragraphs as centered</td>
</tr>
<tr>
<td>“Formatar parágrafo justificado”</td>
<td>Aligns the selected paragraphs as justified</td>
</tr>
<tr>
<td>“Formatar parágrafo esquerda”</td>
<td>Left aligns the selected paragraphs</td>
</tr>
<tr>
<td>“Formatar parágrafo direita”</td>
<td>Right aligns the selected paragraphs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Screen Reading</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“Sintetizar documento”</td>
<td>Synthesizes speech from the document’s text</td>
</tr>
<tr>
<td>“Sintetizar seleção”</td>
<td>Synthesizes speech from the selected text</td>
</tr>
</tbody>
</table>
• **Microsoft Outlook 2007**

✓ *Outlook’s Primary Interface*

<table>
<thead>
<tr>
<th>Generic Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Enviar e receber mensagens”</td>
<td>Sends and receives messages</td>
</tr>
<tr>
<td>“Anular”</td>
<td>Undoes an action</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Creating / Replying / Forwarding Messages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Escrever correio electrónico”</td>
<td>Creates a new message</td>
</tr>
<tr>
<td>“Responder”</td>
<td>Replies to the sender of the message</td>
</tr>
<tr>
<td>“Responder a todos”</td>
<td>Replies to the sender and all other message’s recipients</td>
</tr>
<tr>
<td>“Reencaminhar”</td>
<td>Forwards the message to someone else</td>
</tr>
<tr>
<td>“Reencaminhar como anexo”</td>
<td>Forwards the message as an attachment</td>
</tr>
</tbody>
</table>
### Navigating through the Application

<table>
<thead>
<tr>
<th>Portuguese</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Alternar entre painéis”</td>
<td>Moves between the navigation pane, the main Outlook window, the reading pane, and the to-do bar</td>
</tr>
<tr>
<td>“Mudar para correio”</td>
<td>Switches to mail</td>
</tr>
<tr>
<td>“Mudar para calendário”</td>
<td>Switches to calendar</td>
</tr>
<tr>
<td>“Mudar para contactos”</td>
<td>Switches to contacts</td>
</tr>
<tr>
<td>“Mudar para tarefas”</td>
<td>Switches to tasks</td>
</tr>
<tr>
<td>“Mudar para notas”</td>
<td>Switches to notes</td>
</tr>
<tr>
<td>“Mudar para lista de pastas”</td>
<td>Switches to the folder list in navigation pane</td>
</tr>
<tr>
<td>“Mudar para atalhos”</td>
<td>Switches to shortcuts</td>
</tr>
<tr>
<td>“Abrir livro de endereços”</td>
<td>Opens the address book</td>
</tr>
<tr>
<td>“Ver caixa de entrada”</td>
<td>Switches to inbox</td>
</tr>
<tr>
<td>“Ver caixa de saída”</td>
<td>Switches to outbox</td>
</tr>
<tr>
<td>“Mensagem anterior”</td>
<td>Switches to previous message</td>
</tr>
<tr>
<td>“Mensagem seguinte”</td>
<td>Switches to next message</td>
</tr>
</tbody>
</table>

### Message Manipulation

<table>
<thead>
<tr>
<th>Portuguese</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Apagar mensagem”</td>
<td>Deletes the selected message</td>
</tr>
<tr>
<td>“Marcar como lida”</td>
<td>Marks the selected message as read</td>
</tr>
<tr>
<td>“Marcar como não lida”</td>
<td>Marks the selected message as unread</td>
</tr>
<tr>
<td>“Marcar como não sendo lixo”</td>
<td>Marks the selected message as not junk</td>
</tr>
</tbody>
</table>
**Outlook’s E-mail Reading Window**

- **Message Management**
  - “Nova mensagem” Opens a new message
  - “(Salvar/Guardar) mensagem” Saves the message
  - “Fechar mensagem” Closes the message

- **Replying / Forwarding Messages**
  - “Responder” Replies to the sender of the message
  - “Responder a todos” Replies to the sender and all other message’s recipients
  - “Reencaminhar” Forwards the message to someone else
  - “Reencaminhar como anexo” Forwards the message as an attachment

- **Message Manipulation**
  - “Marcar como não sendo lixo” Marks the selected message as not junk
## Navigating between Messages

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Mensagem anterior”</td>
<td>Switches to previous message</td>
</tr>
<tr>
<td>“Mensagem seguinte”</td>
<td>Switches to next message</td>
</tr>
</tbody>
</table>

## Screen Reading

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Sintetizar mensagem”</td>
<td>Synthesizes speech from the message’s text</td>
</tr>
</tbody>
</table>
✓ Outlook’s E-mail Writing Window

<table>
<thead>
<tr>
<th>Generic Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Enviar mensagem”</td>
</tr>
<tr>
<td>“Abrir livro de endereços”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Nova mensagem”</td>
</tr>
<tr>
<td>“(Salvar/Guardar) mensagem”</td>
</tr>
<tr>
<td>“Fechar mensagem”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Desfazer”</td>
</tr>
<tr>
<td>“Refazer”</td>
</tr>
<tr>
<td>“Verificar a ortografia”</td>
</tr>
</tbody>
</table>
### Switching View

<table>
<thead>
<tr>
<th>Switching View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Vista esquema de impressão”</td>
<td>Alternates between the message’s views</td>
</tr>
<tr>
<td>“Vista de destaque”</td>
<td></td>
</tr>
<tr>
<td>“Vista de rascunho”</td>
<td></td>
</tr>
</tbody>
</table>

### Navigation through the Message

<table>
<thead>
<tr>
<th>Navigation through the Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Campo anterior”</td>
<td>Moves the cursor between the message’s fields</td>
</tr>
<tr>
<td>“Campo seguinte”</td>
<td></td>
</tr>
<tr>
<td>“Saltar/Transitar para princípio da mensagem”</td>
<td>Moves the cursor to a specific place on the message</td>
</tr>
<tr>
<td>“Saltar/Transitar para fim da mensagem”</td>
<td></td>
</tr>
<tr>
<td>“Saltar/Transitar para princípio do parágrafo”</td>
<td></td>
</tr>
<tr>
<td>“Saltar/Transitar para fim do parágrafo”</td>
<td></td>
</tr>
<tr>
<td>“Saltar/Transitar para princípio da linha”</td>
<td></td>
</tr>
<tr>
<td>“Saltar/Transitar para fim da linha”</td>
<td></td>
</tr>
<tr>
<td>“Saltar/Transitar para princípio da palavra”</td>
<td></td>
</tr>
<tr>
<td>“Saltar/Transitar para fim da palavra”</td>
<td></td>
</tr>
<tr>
<td>“Caracter anterior”</td>
<td></td>
</tr>
<tr>
<td>“Caracter seguinte”</td>
<td></td>
</tr>
</tbody>
</table>

### Text Deletion

<table>
<thead>
<tr>
<th>Text Deletion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Apagar”</td>
<td>Deletes the previous character</td>
</tr>
<tr>
<td>“Apagar à direita”</td>
<td>Deletes the next character</td>
</tr>
<tr>
<td>“Apagar palavra à esquerda”</td>
<td>Deletes the previous word</td>
</tr>
<tr>
<td>“Apagar palavra à direita”</td>
<td>Deletes the next word</td>
</tr>
<tr>
<td>Text Selection</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>“Seleccionar (tudo/mensagem)”</td>
<td>Selects text blocks from the message</td>
</tr>
<tr>
<td>“Seleccionar parágrafo anterior”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar parágrafo”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar parágrafo seguinte”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar linha anterior”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar linha”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar linha seguinte”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar palavra anterior”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar palavra”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar palavra seguinte”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar caracter anterior”</td>
<td></td>
</tr>
<tr>
<td>“Seleccionar caracter seguinte”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Copying and Moving Text</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“Mover parágrafo para cima”</td>
<td>Moves the current paragraph up</td>
</tr>
<tr>
<td>“Mover parágrafo para baixo”</td>
<td>Moves the current paragraph down</td>
</tr>
<tr>
<td>“Cortar texto”</td>
<td>Cuts the selected text</td>
</tr>
<tr>
<td>“Copiar texto”</td>
<td>Copies the selected text</td>
</tr>
<tr>
<td>“Colar texto”</td>
<td>Pastes the selected text</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text Alignment</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“Aumentar avanço”</td>
<td>Creates a hanging indent</td>
</tr>
<tr>
<td>“Diminuir avanço”</td>
<td>Reduces a hanging indent</td>
</tr>
<tr>
<td>“Adicionar marcas”</td>
<td>Applies bullets to the current paragraph</td>
</tr>
</tbody>
</table>
### Text Formatting

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Formatar fonte”</td>
<td>Opens the Font dialog box</td>
</tr>
<tr>
<td>“Formatar fonte verdana”</td>
<td>Changes the font of the selected text</td>
</tr>
<tr>
<td>“Formatar fonte times”</td>
<td></td>
</tr>
<tr>
<td>“Formatar fonte arial”</td>
<td></td>
</tr>
<tr>
<td>“Diminuir tamanho da fonte”</td>
<td>Changes the font size of the selected text</td>
</tr>
<tr>
<td>“Aumentar tamanho da fonte”</td>
<td></td>
</tr>
<tr>
<td>“Formatar tamanho (seis - trinta)”</td>
<td></td>
</tr>
<tr>
<td>“Bold” or “Negrito”</td>
<td>Applies bold formatting</td>
</tr>
<tr>
<td>“Itálico”</td>
<td>Applies italic formatting</td>
</tr>
<tr>
<td>“Sublinhado”</td>
<td>Applies an underline</td>
</tr>
<tr>
<td>“Maiúsculas” or “Capitalizar”</td>
<td>Changes the case of letters</td>
</tr>
<tr>
<td>“Limpar formatação”</td>
<td>Removes all manual formatting</td>
</tr>
</tbody>
</table>

### Paragraph Formatting

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Formatar parágrafo centro”</td>
<td>Aligns the selected paragraphs as centered</td>
</tr>
<tr>
<td>“Formatar parágrafo justificado”</td>
<td>Aligns the selected paragraphs as justified</td>
</tr>
<tr>
<td>“Formatar parágrafo esquerda”</td>
<td>Left aligns the selected paragraphs</td>
</tr>
<tr>
<td>“Formatar parágrafo direita”</td>
<td>Right aligns the selected paragraphs</td>
</tr>
</tbody>
</table>

### Screen Reading

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Sintetizar mensagem”</td>
<td>Synthesizes speech from the message’s text</td>
</tr>
<tr>
<td>“Sintetizar selecção”</td>
<td>Synthesizes speech from the selected text</td>
</tr>
</tbody>
</table>