Design of a Speech Interface for Augmenting Desktop Accessibility

David F. Rodrigues
L²F - Spoken Language Systems Laboratory, INESC ID Lisboa / IST
R. Alves Redol, 9, 1000-029 Lisboa, Portugal
http://l2f.inesc-id.pt/
{David.F.Rodrigues}@l2f.inesc-id.pt

Abstract. The goal of this paper 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 (ASR) and text-to-speech (TTS) engines, and also gives visual feedback through an animated face. The target language intended for the voice modality is the European Portuguese. In this paper we expose the opportunities created by these technologies in order to enhance user interaction with application interfaces. To do this, we will use mainly our command, control, and dictation system, designed to operate in the scope of a desktop environment.

Keywords: Multimodality; embodied conversational agent; interaction; speech recognition; speech synthesis; animated face; user interface.

1 Introduction

During recent years computers have witnessed an emerging paradigm in user interfaces. 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, which support more challenging applications, and more difficult usage conditions by combining the strength of multiple and sometimes complementary communication channels.

The interpretation of speech and gesture started to be studied with Bolt’s innovative work on deictic gesture, in a project named Put-That-There [1]. This system was one of the first multimodal systems, and took advantage of the conjoint use of voice-input and gesture-recognition to command events. Since this work much research on multimodal systems has been presented: Koons [2], Bolt and Herranz [3], Wahlster [4], Johnston [5] and others.

The concept of multimodality has proved to introduce considerable design and development overheads to the implementation of such systems. Actually, the
designing phase, of some complex systems, must consider support for context sensitivity, synchronization of animation and TTS output, and dialogue capabilities.

The research developed in the scope of this paper focuses on a particular type of multimodal systems, which are embodied conversational agents (ECAs). These virtual agents represent animated anthropomorphic characters that can engage a real-time multimodal interaction using speech, gesture, posture, or other verbal and non-verbal behaviours, emulating the experience of human face-to-face dialogue [6].

ECAs are being increasingly employed in the development of a new family of user interfaces for a wide diversity of applications. These applications are trying to take advantage of the referred sophisticated technologies, to facilitate the accomplishment of different tasks. There are many researchers who are advancing in this new generation: Cassell [7], André and Rist [8], Bates [9], Lester [10] and others.

The work underlying this paper tries to present a step forward, in respect to the introduction of multimodal interaction, through an ECA, in the context of an operating system. Our initial driving force was to empower desktop environment with the ability of command and control speech recognition and voice dictation. Later we got interested in expanding the system with speech synthesis capability, ending up with a platform that is extensible in respect to input and output modalities. In the final stage our main focus was to maximize the quality, utility, and usability of the modalities, provided by our speech engines.

In order to describe and analyze our command, control, and dictation system we refer mainly to its implementation on Microsoft Windows Vista operating system. We will use this work to expose the possibilities that our speech 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 spoken interface.

In section 2 we will present the implementation of our platform by the command, control, and dictation system on Windows Vista environment. Section 3 gives a brief description of the platform’s architecture, and in section 4 we describe the interaction’s characteristics. The conclusions are presented in section 5.

Fig. 1. The dictation of a document using the command, control, and dictation system.


2 The Command, Control and Dictation System

In our laboratory, we have applied our base technologies, such as automatic speech recognition, natural language understanding, and speech synthesis, in a wide variety of environments. We have been researching speech interfaces in different scenarios including home automation and telephone interactive voice response systems [11]. These interfaces provide user-friendly means to handle information systems, since speech is probably the most natural of all interaction modalities.

The command, control, and dictation system empowers computer users to interact with their desktop environment by voice. It was designed to allow a significant reduction of mouse and keyboard use, while preserving or even boosting the overall productivity of human users.

The system was specifically developed to work under Microsoft Windows Vista. In order to improve the computer accessibility we have made a general effort to explore and understand the interaction opportunities which exist in the context of this particular environment. In this way we were able to present a usable system which guides the user to complete the tasks at hand.

The system allows people to dictate to their computers and have their speech automatically transcribed into their email or word processing applications. It also enables innumerable ways to control the operating system with voice. Starting and switching between applications, selecting portions of text, correcting sentences, synthesizing documents, and selecting menu options are just some of the interaction possibilities.

Our system has already been presented to some beta testers revealing 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.

Fig. 2. The creation of an email using the command, control, and dictation system.
3 The Overall Platform

This section presents a general overview of the platform’s design and implementation. First we will explain how the multimodal access is provided, then we are going to introduce the speech engines, and finally we will examine the system’s architecture.

3.1 Multimodal Input / Output

Figure 3 portraits a block diagram which illustrates how the users’ commands are recognized, and how the multimodal interaction is done within the system.

The platform incorporates spoken conversation, speech synthesis, a reactive animated face, and direct manipulation interfaces. 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.

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.

The default speech engines used for this setup are AUDIMUS ASR engine and DIXI TTS engine. These engines and their features will be described with moderate detail in the next section.

All the interaction between the applications and the speech modality engines is done through software functions which are specified by engine interfaces. These software functions are usually named Speech Application Program Interfaces (SAPI). In our platform, the access to these functions is centralized in a singleton runtime component, called Modality Server (MS), which will be better described on a further section of this paper.
3.2 The Speech Engines

Both speech engines, which are supporting the spoken modalities in our platform, were developed in our laboratory [12].

The automatic speech recognition is provided by AUDIMUS, a hybrid speech recognizer that combines the temporal modeling capabilities of Hidden Markov Models with the pattern discriminative classification capabilities of multilayer perceptrons. This same recognizer is being used for different complexity tasks based on a common structure but with different components. The acoustic models of the AUDIMUS system are speaker independent. The vocabulary and language model, due to the goal of task independency, should be automatically extracted from a configuration XML file.

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 Architecture Design Overview

![Diagram of platform’s overall architecture](image)

**Fig. 4.** The platform’s overall architecture.
Figure 4 exhibits a general view of the platform’s architecture and component arrangement.

As illustrated, our software approach aims at fulfilling the gap between our base technologies and desktop applications. The objective is technological transparency and a more efficient management of engine resources. On the other hand we provide an interface which represents a common protocol on which programs can interface with the modality engines.

Another fundamental aspect, in this architecture, lies mainly on the fact that we want to elaborate a generic multimodality providing platform, which can be used in several and different applications. In this way, context management and awareness become crucial necessities in order to potentiate all the interaction possibilities within a desktop environment.

In order to promote simultaneously robustness and extensibility, the handling of commands is done in separate places, as can be seen in figure 4. A generic reactive module receives the recognition results and, following a general supervision logic, performs the respective associated actions. By the other hand, particular application add-ins also receive the recognized commands that the user has spoken. These add-ins are responsible for specialized cover of certain application needs.

**Middleware Characteristics**

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. Therefore, it becomes vital to create a layer of software which stands between applications and speech engines, empowering them to communicate in a standardized way.

We present a modality server that is supposed to be used by speech-enabled applications so that they can communicate with our speech engines.

The modality server can be viewed as a piece of middleware which implements an application programming interface (API), and the engine driver interface (EDI). Modality engines should use the EDI layer to become accessible to the applications, which are served by the API layer. This API 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. Every one of these modality capabilities can perfectly be used separately from each other.

Both layers of the modality server service are implemented as collection of component object model (COM) interfaces. In this way, the application-level interface possesses a set of entry points which are accessible from a variety of programming languages. Users are able to instantiate an engine, and afterwards applications 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.
The API also 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 applications.

Context Management

Our shared ASR engine object will be supposed to stand simultaneously accessible in several distinct contexts. The primary obstacle in achieving this goal emerges from the functional diversity of application software, causing the existence of a large-scale number of operational domains.

The designing of a perfectly task-independent system would be the most desirable solution. However, this still represents an unsolved challenge in the speech recognition area. In fact, there is an absence of a general model or vocabulary, which is able to deal with everything that can possibly be said by the user.

In the AUDIMUS recognizer, the lexical and the language models are also dependent on the specific application domain. 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.

Our platform’s strategy to deal with the task-independency problem gives priority to the system’s robustness. The goal is to achieve reliable enough performances, in the actual context of the applications. At the same time, we try to minimize the memory and the processing load, needed to adequately manage the engine’s possible states. We propose an architectural module which is responsible for context management and awareness. The module is represented in Figure 4 by the name of Context Manager (CM).

The CM is responsible for domain awareness, which becomes essential to identify if the interaction with the current interface demands for dictation or command and control capabilities. In addition, this knowledge also allows the correct configuration of each one of these engine operating modes. The procedures by which the speech recognition engine accomplishes each one of the two applications tasks are functionally very different. In dictation circumstances, the words that the engine is able to recognize, are not predefined and may come from a very large vocabulary. However, in command and control applications, the speech recognition engine is, actually, aware of the words that the user may speak.

In respect to dictation, we have created specific language models that were built from huge amounts of text data. In opportune occasions, the user is presented with the possibility to choose the respective subject for his dictation purposes. This subject is associated with a given language model which, in turn, is loaded on the engine at runtime. The context manager possesses the necessary knowledge to detect these dictation occasions, and to choose the appropriate models for each one of them.
In respect to the command and control application area, we propose a hybrid approach that combines the usage of constant grammars, which can be generic or application specific, with dynamic generated ones. The context manager is responsible for detecting variations between the different domains, and for performing the consequent grammar changes. Additionally, this component features focus awareness and screen reading capabilities, which, combined together, provide the ability to dynamically create grammars that can represent unforeseen interaction opportunities.

**Granting Extensibility**

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. By the other hand, we were interested in an approach where the platform's functionality could be modifiable without the demand for the alteration or recompilation of the whole original source code. With this purpose in mind we included in our platform a public API which allows the extension of the overall spoken interface, by software developers who don't have access to the system’s source code.

We pretend to customize and integrate speech technologies into a broad variety of mainstream applications. The goal is achieve a vast range of interaction possibilities, making the overall speech interface as complete as possible.

It is desirable to create a general implementation which is able to deal with virtually any native application. Dictation, for instance, can be easily handled in a generic manner for every window. However, with a generic approach, important features will be lost, like the ability to select text and correct it. Most importantly, specific command and control functionality will be extremely difficult to achieve in this way. Users will experience extreme frustration when trying to interact with the computer only using speech recognition.

Fortunately, with appropriate and specific integration, most of this frustration can be avoided. We suggest an approach based on add-in integration, which brings parts together into a whole. The strategy’s purpose is to bridge the gap between what the speech technology has to offer and what is needed for the many different applications to accomplish.

The add-in consists of a computer program able to interact with a host application with the objective of providing a specific functionality. It allows us to potentiate systems, with spoken interaction, when this feature was unforeseen. On the other hand, this strategy gives us the freedom to separate the code of an application because of the incapability to access the manufacturer’s private source.

Our implementation of this add-in framework offers the advantage of allowing functional modules to be added without the system having prior knowledge about them. At the same time, it does not constrain the capability to create seamless transitions between the distinct nonstandard application domains.

We have been interested on the integration of spoken interaction within the context of word processing and email applications. Much effort has been devoted in this direction by the creation of add-ins for such applications as Microsoft Word (see Figure 1) and Microsoft Outlook (see Figure 2). We have produced complementary
command sets to make the speech recognition software more helpful in these environments. Nevertheless, incorporating speech technology is not just about plain application control and dictation capabilities. The ability to enter data into templates or custom applications efficiently has also been taken in account. Highly attention has been paid to the integration of speech into the user’s workflow. We have also made the speech recognition technology more practical by creating customized commands to easily operate the features of the speech software (see Figure 5).

4 The Interaction’s Characteristics

Once the command, control, and dictation system launches, the user is prompted with a gadget interface, which can be seen in figures 1 and 2. From this point on, the spoken interaction can be started or interrupted, through the gadget, according to the user’s needs.

With command and control capabilities, the speaker is able to give orders to the applications. However, when dictating, the speaker’s input is not interpreted and is exclusively converted in text format. Figure 5 shows an 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. 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 a 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. 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 system displays the animated face, which engages the user in a dialog to complete certain key tasks, like the creation of emails.

![Fig. 5. Interfaces for the selection of the: a) ASR models; b) TTS voice.](image)
5 Conclusions

The system has been submitted to a usability evaluation process and the feedback we have been receiving from users has been substantially positive. They 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.

Future work on this project might involve 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 command, control and dictation system.

References