An Adaptable Speech System for MS-DOS Based Personal Computers

Gwong Sun
Raymond Wisman
Khaled Kamel
Engineering Mathematics & Computer Science
University of Louisville
Louisville, Kentucky

ABSTRACT:

A software system to interface standard personal computer applications to speech synthesis hardware will be presented. It has been designed for use with applications utilizing the Microsoft Disk Operating System on International Business Machines Personal Computers and compatibles. A variety of phoneme driven speech synthesizers may be supported.

The system is composed of three main software elements: (1) an interface to the input and output of the standard application via the operating system, (2) a user interface providing control of the system, (3) translation of text-to-speech. Hardware consists of the host computer system and a simple speech synthesizer.

The speech system is adaptable to various speech synthesizer hardware. Adapting the speech system to another synthesizer requires two changes; to the IPA-to-synthesizer translation rule file to reflect that specific synthesizer's phoneme code representation, and to the software that drives the synthesizer hardware to reflect the manner in which the hardware interfaces to the computer system.

I. INTRODUCTION

Interest in artificial speech has existed for many years and has often been on the threshold of the current technology. Computer synthesized speech research was well under way by the early 1960's and the components needed to generate human speech from text were known by 1970 [1]. In current computer applications, three types of speech generation predominates: linear-predictive coding (LPC), waveform digitization and phoneme synthesis. These methods differ considerably in speech quality, vocabulary size and area of application.

Linear-predictive (LPC) coding is based on the frequencies found in speech. The vocal tract is modeled using stored filter coefficients, amplifier-gain settings, and excitation frequencies. Waveform digitization technology consists of recording a human voice, applying analog-to-digital conversion, storing the digitized parameters of speech then reversing the process to play it back under compute control. The speech quality produced is generally quite good but has large memory and data rate requirements.

Phoneme based speech technology electronically models the human vocal tract. Each of the approximately forty phonemes, the basic sounds of speech, is individually modeled and produced. By stringing together phonemes in the proper time and sequence, recognizable speech is produced. The principal advantage of this method is that unlimited vocabularies may be generated from combinations of these phonemes. Since phonemes are basic to all speech, foreign languages may also be produced by this method. However, phoneme based speech quality is often lower than preferable for many applications.

In general, there have been few successes in developing synthetic speech into a viable means of interacting with computer applications. The goal of this paper is to explore the problem of producing synthetic speech for standard computer applications. Techniques are utilized to enhance hardware independence, especially that of the speech synthesizer. This problem is approached within the general context of converting the usual text generated by the computer and the computer operator interaction into speech.

The general problem of converting standard textual computer output to spoken output has several stages that diagrammatically appear as in Figure 1.

(1)	(2)	(3)	(4)
Intercept text to console	=> Apply user constraints to text	=> Translate text resulting from for speech	=> Generate speech from translation

Figure 1 - Stages of Conversion from Textual Output to Speech Output

One may consider the processing of a given word from computer generated textual output into speech as essentially a series of sequential operations. A word enters stage 1 in text form, is processed at each step until reproduced in corresponding speech at stage 4. The first stage involves capturing any text being transmitted to the console from either the operator via the keyboard or as a result of functioning of the computer. This should not interfere with the normal textual output or overall functioning of the computer system. The second stage is that of collecting and applying user constraints to text previously captured before being translated to phonemes. This stage provides for control over certain aspects of speech production to allow adjustment to personal preferences. The third stage is that of translating form text to a form suitable for speech production by a synthesizer. The most desirable situation is where an unlimited vocabulary be produced, meaning that textual input be unrestricted in terms of words occurring within that text. The translation process must necessarily be rapid enough as to not slow the apparent

CH2674-5/89/0000-1163$01.00©1989IEEE

overall functioning of the computer system. A certain amount of speech reduction is to be expected as a by-product of speaking normal output, given it is naturally somewhat slower than normal video textual output.

The fourth stage is the actual production of the speech from phonemes resulting form the text-to-phoneme translation. For purposes of this paper, the final stage is implemented in hardware by a general purpose phoneme driven synthesizer. For completeness, a detailed design of the speech synthesizer circuitry is given in the following. It is however, part of this project's goal to develop a speech system that is largely independent of the device that produces the sounds.

II. HARDWARE ENVIRONMENT

A number of hardware options are available when implementing a speech system. One system that demonstrates the feasibility of replacing computer visual feedback with audio is the Total Talk system [2], which consists of a terminal, a special duty microprocessor system, a speech synthesizer and a host computer running standard software. This configuration is typical of systems that produce speech from the normal terminal-host computer interaction.

The system chosen for this paper consists of a single computer executing standard application software plus all special software necessary for speech production. A wide variety of computer systems, including personal computers, can support such a configuration. The synthesizer can be contained within the computer attached to the main system bus or as a stand alone device connected to an external port. The specific hardware components used are:

1. International Business Machine Personal Computer Model 5150 (IBM-PC) with 256 kilobytes random access memory, 2-360 kilobyte diskettes, and graphics video interface.
2. Custom internal speech synthesizer board using Silicon Systems Inc. SSI-263A speech synthesizer chip.
3. 8 Ohm speaker.

A number of initial considerations are required. Several are imposed by the goals of the project, others are by nature expedient. Those of main importance to the synthesizer design are:

phoneme driven synthesizer - allows production of unlimited vocabulary

bus connection - simplifies overall design, no independent power supply or inter-system communication devices such as UARTs

simple interfacing - reduce the complexity of synthesizer board design.

The circuit that best satisfies the initial requirements and the one selected is the SSI-263A. The SSI-263A synthesizer circuit is phoneme-based, TTL (transistor-transistor logic) compatible throughout and designed for ease of interfacing with micro-processors. It allows for unlimited vocabulary with very low data rates for monotone speech (less than 100 bits per second)[3]. The circuit contains 5 eight bit registers that allow software control of speech rate, pitch, pitch movement rate, amplitude, articulation rate, vocal tract filter response and phoneme selection and duration.

The synthesizer circuit consists of a single 24-pin CMOS chip (Complementary Metal-Oxide Semiconductor) powered by a single 5 V source. Interfacing the device via the system bus to a computer system involves the decoding of the appropriate addresses and control signals from the computer bus to select the device, generation of the proper clock signal and connection to the data lines. The synthesizer board connects to the computer bus in any available slot. All signals and power requirements are supplied via the computer bus.

The synthesizer device requires a clock signal to be synchronized with the computer system clock. However, in this case the clock signal is derived from a system clock having a rate of 14.318 MHz as its base. Clocking for the device is derived from the system clock after dividing the pulse by 16 to produce a .89 MHz clock with a 50% duty cycle. The analog output of the circuit is filtered and amplified to directly drive an 8-ohm speaker.

III. Software Overview

In order to convert textual output of standard software into equivalent spoken audio output, project software must accomplish the three stages outlines:

1. Intercept textual output to computer console - character capture.
2. Provide user control over speech process - user interface.
3. Translate resulting text into form appropriate for synthesizer - text to speech translation.

3.1 Stage 1 - Character Capture

As the first stage of the process that leads to speech production from the textual output of standard computer software, capture of that text is also the part most subject to the idiosyncracies of the software/hardware environment. The general problem is first to detect when output is being made to the console or the keyboard is being exercised by the user. These may be detected at the hardware level or through the operating system software. The MS-DOS 2.0 operating system offers several possible methods within the structure of a standard software environment. Using MS-DOS the problem becomes one of interfacing MS-DOS console input/output functions to those of the speech system character capture functions.

The software interface between the MS-DOS operating system and all other project software has a rather simple requirement, that of trapping either keyboard input or output directed to the screen. Characters trapped are first processed by project software and either returned to calling routines when finished or used only internal to the speech system for control purposes such as cursor movement.

Using the available functions in MS-DOS, it is rather simple to store speech system addresses into the interrupt vector to transfer control of a MS-DOS system interrupt to an interfacing routine address is replaced by the address of the character capture routine. An input interrupt causes processing control to transfer to the character capture routine. The character capture routine calls the BIOS (Basic Input and Output System) routines [4] normally used by the MS-DOS. When control is returned to the capture routine interface, the input is processed and returns control to the interrupting software. Both keyboard and video functions may be treated in a similar manner.

When an application requires keyboard input or a video function, a software interrupt is issued requesting service. Normally, the request is vectored directly to the BIOS routines, serviced and results returned to the application that made the initial request. In order to capture the textual output resulting

from the keyboard and video interaction, the capture routines are inserted into the flow of control between the application and the BIOS operation.

The character capture portion is necessarily designed for a specific operating system environment using a non-portable language, Assembler. The remaining major portions of the speech system software, the user interface and text-to-speech translation, are coded in the C language and are mainly operating system independent.

3.2 Stage 2 - User Interface

In general terms, the user interface provides overall system control to the user via commands entered from the keyboard or otherwise displayed to the console. commands are separated in two groups by the criteria of immediate or delayed response to the commands. some commands, such as cursor positioning, must be performed immediately. Others, such as conversion of the abbreviation "Mr." to "Mister," would be delayed until the "Mr." occurred in text being processed. The group of delayed commands controls such functions as:

1. Turn text-to-speech translation on/off.
2. Add/delete/replace stored commands that control character replacements or skipping of characters sent to text-to-speech translator.
3. Switch modes to character or word spoken at a time.
4. Turn effect of stored commands on/off.
5. List (speak) stored commands.

These functions would not normally be used a great deal, only to set initial configurations of the speech system and for occasional alterations. Consider that one wanted the word "Mister" spoken when "Mr." occurred in text. This would be accomplished by storing a command to convert "Mr." to "Mister" before text-to-speech translation.

Stored commands control elements of the interface that, from the perspective of the user, are not dynamic. Their effect is automatic, once initiated, requiring no action by the user. Stored commands are used primarily to set conditions and tailor spoken output to personal requirements. The following Table I lists the stored commands and their functions.

Table I
Stored Command Codes

(Form: $COMMAND[string1]$[stromg2]$)
Note: $ = Ctrl A is default and COMMAND either upper or lowercase

s	toggles speech on/off
w	puts system in word mode
c	puts system in character mode
a	adds an abbreviation
d	deletes an existing abbreviation
i	ignores text from string1 to string2
l	lists and speaks stored commands

All commands that require storage of their parameters make use of this database. The stored commands and speech system settings may be listed (spoken) or written to a file for later retrieval. It allows the creation of libraries of commands to control speech functions tailored for a particular application or taste.

The remaining requirement of the user interface is to allow the cursor to be moved about the screen and characters on the screen processed in the same manner as keyboard/video output characters and corresponding speech produced. This allows text appearing on the video screen to be reviewed. Two modes of cursor movement and speech production are utilized, moving and speaking either a character or a word at a time.

Cursor control is accomplished by shifting the standard cursor control keys. This is to allow the use of the speech software in conjunction with software that utilize the same keys in the more standard, unshifted state. The shifted numeric keypad is therefore not useable by other applications in conjunction with the speech software. The cursor control keys and their function are illustrated in Table II.

Table II
Control Key Definitions

^	Moves up 1 line
v	Moves down 1 line
->	Speaks character/word to right of cursor
<-	Speaks character/word to left of cursor
Home	Speaks entire line to left of cursor
End	Speaks entire line to right of cursor
Pg Up	Speaks entire screen to left of cursor
Pg Down	Speaks entire screen to right of cursor
ESC	Escapes from command entry mode
Ctrl Break	Aborts speech started by cursor control

After text has been processed through the user interface, it is then passed to the final, major stage, that of translation from the text output by the standard operation of the computer, to phoneme codes, serving as input to a speech synthesizer.

3.3 Stage 3 - Text-To-Speech Translation

The method of translation of text into phonemes used in this paper is the Rule Translation - by matching individual or patterns of letters with stored rules to produce corresponding phonemes. These rules determine the phonemic result of a given letter pattern. Each letter of the alphabet has it's respective list of rules and resultant phonemes. The most complex rules are considered first with the very last rule being the rule for the individual letter. In that way letter patterns that do not match any rule, such as random letters, would be spelled out letter by letter.

The advantages of such a method are: small size, only about 350 to 400 rules are necessary. Simplicity, all words are dealt with using the same method of translation. And adaptability, to change pronunciations or synthesizer support, only the rules need be altered, no program coding change is needed since rules are stored in a standard text file.

The basic approach requires two distinct steps, that of rule formulation and that of rule application. In the rule formulation stage, which is done by hand, letter combinations

are assigned a particular phonetic equivalent. As a simple example, the OOK combination as it occurs in COOK is given the IPA (International Phonetic Alphabet) [5] equivalent of 'U k'. Rule application requires that the text OOk to be matched with the appropriate rule.

Rules are generally made up of four parts: left part, the letters preceding the specific letter combination being translated; exact part, the letter combination exactly identical to the text being translated; right part, the letters following the letter combination being translated; translation result, the corresponding phonemes of the translation. The left and right parts may be empty. The Table III lists the rule syntax.

Table III
Translation Rule Form

All rules have the form of:
left [exact] right = result

where
letter ::= ASCII character
vowel ::= A|E|I|O|U|Y
consonant ::= B|C|D|F|G|H|J|K|L|M|N|P|Q|R|S|T|U|V|W|X|Z
::= B|D|V|G|J|L|M|N|R|W|Z (voiced consonant)
% ::= ER |ED |E |ES |ING |ELY (suffix)
& ::= S|C|G|Z|X|J|CH|SH (sibilant)
@ ::= T|S|R|D|L|Z|N|TH|CH|SH (long u)
+ ::= E|I|Y (front vowel)
# ::= 1 or more vowels
* ::= 1 or more consonants
$ ::= 1 consonant followed by E or I
^ ::= 1 or more vowels
: ::= 1 consonant
left ::= combination of letters|&|@|+|#|.|*|$|^|:
exact ::= letters
right ::= comb. of letters|&|@|+|#|.|*|$|^|:|%
result ::= IPA phoneme equivalent

A modified thumb-index technique (6) is used for rule storage and access. Rules are grouped together by the letter starting the letter combination to be translated (e.g. #(AE)S=a z would be grouped with the letter A rules and would translate AE) in the order the rules are to be examined. This grouping and ordering of the rules is reflected in the data structure used to access the rules during their application to text. The rules are read from a file and stored using the starting letter as an index into an array of linked lists. The rule would be stored at the end of the linked list, maintaining the order in which the rules are read from the file. The four components would be: left=#, exact=AE, right=S and result=a z. Due to the difficulty in representing the IPA symbols, in practice they are equated to an ASCII character representation.

Rule application of the text-to-speech rules to computer generated text produces IPA phoneme results. To apply the rules to text, the rule list for the beginning letter of the text to be translated is searched until either a match is found for the right, exact and left components of a rule, or the end of the rule list for that letter is reached. In the former case the resultant phonetic translation is produced. In the latter, the default (the last rule of the list) is used as the translation which has the effect of spelling some words.

Some commercial synthesizers will not accept phonemes in that form. Necessarily, a similar translation of the phonemes from the IPA form to that acceptable to a specific synthesizer is required. Most synthesizers are generally capable of producing the sound corresponding directly to the IPA phonemic representation through actual phoneme code representation varies. It is this two step process of translation to IPA then to synthesizer phoneme representation that provides the adaptability of this method to function with different speech synthesizers.

In many situations the synthesizer is capable of parameter control not defined by the IPA codes. Often a single IPA code may be represented by several synthesizer codes, such as the IPA code of UH has three Votrax representations of UH1, UH2 and UH3. Each synthesizer code produces a UH sound of differing duration. The second stage of translation for IPA-to-synthesizer codes can utilize synthesizer properties that account for the fact that certain phonemes are voiced differently when in conjunction with other phonemes. Selection of the proper synthesizer codes by the translation process results in more understandable speech.

The translation software is written in the C language which consists of two parts, one part that acts as a translation rule preprocessor and a second that applies the translation rules to text. Numbers are pronounced as series of single digits (134 would be one three four) and punctuation marks as their spelled equivalent (* would be pronounced as asterisk). Rules are stored on a text file and read in at program initialization. Words to be translated are input from keyboard or console with the phonetic translation codes output to the synthesizer.

IV. RECOMMENDATIONS

The following recommendations are submitted as possible directions for future development.

Develop software drivers as defined within MS-DOS to interface to speech synthesizer hardware. Used in conjunction with synthesizer translation rules, these would allow greater portability to different synthesizers.

Polish the user interface. Provide on-line help. Study ways that user could store portions of screen, recall and replay at will. Also restrict portions of screen to be spoken in a manner similar to video windowing.

Consider use of speech in multi-tasking personal computer systems as an alternative to console output from background jobs. The foreground function could be monitored primarily via video output while other background functions could be monitored via audio.

Improve speech intelligibility. A continuing problem for all applications of synthetic speech.

V. REFERENCE

1. Sherwood, Bruce, The Computer Speaks, IEEE Spectrum, August 1979, pp. 18-25.

2. Blazie, Deane B., Total Talk a Computer Terminal for the Blind, Proceedings of the First National Search for Applications of Personal Computing to Aid the Handicapped, October 1981, pp. 251-253.

3. SSI 263A Phoneme Speech Synthesizer Data Sheet by Silicon Systems, 14351 Myford Road, Tustin, CA.

4. Roskos, J. Eric, Writing Device Drives for MS-DOS 2.0, Byte Magazine, February 1984, Vol 9, No. 2, pp. 370-380.

5. The Principles of the International Phonetic Association, International Phonetic Association, Department of Phonetics, University College, London, England.

6. Knuth, donald E., The Art of Computer Programming Volume 3/Sorting and Searching, Addison-Wesley Publishing Company, Reading, MA.

KHALED KAMEL

Dr. Kamel is currently a professor of the Engineering Computer Science at the University of Louisville. He joined the Engineering School in 1979. Prior to this, he worked with the Aircraft Engine Group at General Electric Co. as an Instrumental Engineer. He also worked part time with IBM Information System Division, IBM University Program Division, General Electric Appliance Park, the Naval Ordnance Station in Louisville, and others. He completed the Ph.D. degree in Electrical and Computer Engineering at the University of Cincinnati. Dr. Kamel is a senior member of IEEE