

Building Speech Interactivity

Daniel Woo

School of Computer Science and Engineering

The University of New South Wales

Sydney NSW 2052

Australia

Email: danielw@cse.unsw.edu.au

Abstract

Speech technologies have reached sufficient maturity to enable a new generation of human-computer interaction based on spoken conversation, but this requires (i) architecture to control an operating system and applications by speech, and (ii) UI guidelines for spoken dialogs. New features of Mac OS 9 achieve this goal, by enabling users to control AppleScript by voice. Specifically, this paper demonstrates how to navigate a tree-structure of questions using only voice, and provides UI guidelines for goal-oriented conversational computing. Features and limitations of current technology are identified, and suggestions for future directions are prioritized.

Introduction

Speech control of a desktop environment is a major step in the evolution of human interaction with computers. Although the technology is not at a stage where a machine can replicate arbitrary human conversation, speech technology has reached a level of maturity that allows interactive voice applications within a constrained domain to be deployed on a desktop computer.

Speech-based applications require different design principles to be considered since unlike designing for mouse, screen and keyboard both the input and output consist only of verbal elements. Spatial and visible cues do not form part of the spoken domain.

The challenge is to design well-executed spoken dialogues that fulfil useful tasks for human users. In part, AppleScript already provides the component that can carry out useful tasks. The remaining issue lies with the design and deployment of the speech system.

A description of the system components required for speech applications is provided to introduce the speech functions available on the Macintosh. A new AppleScript addition that provides interactive speech capability is discussed in the context of simple examples to demonstrate the syntax of the commands. Several design issues are discussed to provide guidelines that highlight potential pitfalls that may be encountered when working in the voice domain. A typical tree-structure that represents an interactive dialogue of is presented to show how voice transaction flows can be designed and documented and to illustrate a mapping from the tree structure to AppleScript.

Architecture

Speech recognition and synthesis are integrated into the operating system using the Speech Manager and Speech Recognition extensions. These components provide the applications programming interface to the underlying algorithms to manage speech synthesis and recognition as well as interface to sound input and output system services.

In addition to speech services, the AppleScript extension provides a user-programmable scripting language that enables control of many aspects of the operating system services including opening documents and changing the visual appearance of the desktop. AppleScript relies on an interprocess messaging framework called Apple Events that facilitates message passing between applications to initiate specific actions.

A new feature of Mac OS 9 is the Speech Listener scripting addition that integrates speech synthesis, speech recognition and AppleScript. When this component is installed, all scripts may be enhanced with text-to-speech output and recognition of specifically defined phrases spoken by the user. Prior to this release, scripts could only output speech using the *say* command. Speech recognition from within an AppleScript was not available. Speech Listener provides all scripts with the potential to have speech interactivity.

The Speech Listener addition is only available for Mac OS 9 and cannot be used with previous releases of the operating system.

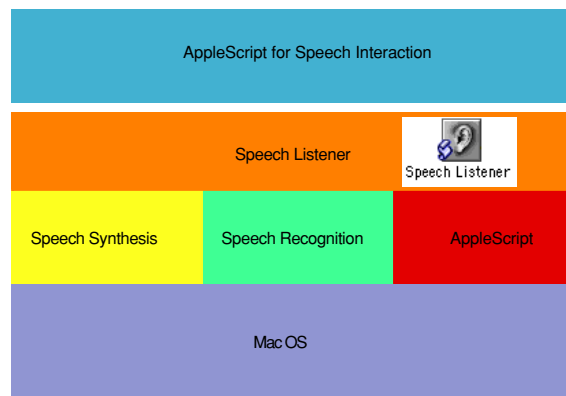


Figure 1. Components required to deploy speech interactive AppleScripts.

Development Environments

The combination of Speech Listener and AppleScript provides a simple and yet powerful environment to deploy speech applications. This means that speech can be used to control all Apple Event aware applications. A major advantage of AppleScript is that Script Editor, a tool for developing and testing scripts is installed in the Apple Extras folder thereby providing users and developers with a free and immediate path to produce scripts.

Alternatively, speech applications that directly interface to the appropriate speech recognition and synthesis APIs can be developed using a lower-level programming language (C/C++/Pascal) and appropriate software compilers [1]. This approach provides access to all available functions and but is considerably more complex. MacPerl [4], an incarnation of the Perl programming environment [5] provides an alternative scripting environment that facilitates access to speech recognition and synthesis services.

Basic Setup

To enable speech recognition, the Speech Recognition Manager must be explicitly installed as it is not included as part of the default installation process. Speech recognition is activated using either the Speakable Items option in the Speech Control panel or using the listen button in the control strip (Figure 2). When Speakable Items is available, a window appears with the feedback character.

As soon as the system is active you are able to issue commands that execute various scripts that are contained in the Speakable Items folder (contained in the Apple Menu items). In the default installation you can say “Computer, what time is it?” and a voice should reply with the current system time. Using the Speech Control panel other settings can be configured by the user including whether commands need to be preceded by “Computer” or some other user-defined name, the character used for both visual and audio feedback and the key to use if press to talk mode is activated.

Any file name in the Speakable Items folder is included in the list of potential speech phrases. If a file called “Connect to library” is contained in this folder, when a user says “Connect to library” the corresponding document is opened. In practice, this could be an alias to a URL that opens a browser at the appropriate page. The document can be a folder, document, script or alias.

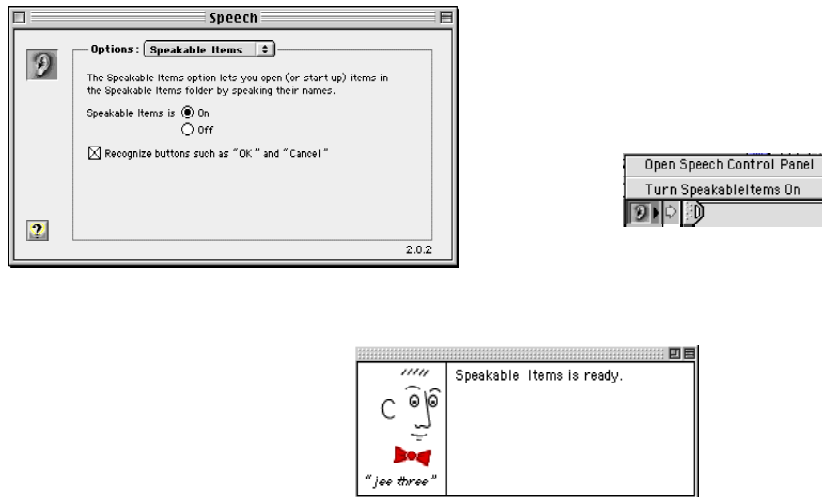


Figure 2 Speech Control Panel, Feedback window and Control Strip button

If the user has not enabled Speakable Items and a script is executed that contains a Speech Listener command, recognition will be automatically activated. When the task is complete, the speech service will close. It is therefore not always necessary to have Speakable Items enabled.

Audio Input

It is recommended that a PlainTalk microphone be used regardless of whether the system has a built-in microphone. The sound input should be set to External mic. The distance of the microphone should be around a metre from the user. Alternatively, headset microphones can be used for better accuracy but they may require a microphone adaptor.

Commands for AppleScript

AppleScript is an extensible scripting language built into the operating system that facilitates the addition of new elements to the language [2]. This aspect of the language has been used to provide speech input and output capability for all AppleScripts. Only two commands are required: *listen for* and *say*. *Listen for* is included as part of the Speech Listener Scripting Addition and is new to Mac OS 9 *say* is defined in the Standard Additions and was originally available as a single scripting addition. These files must be included in the Scripting Additions folder to perform recognition and synthesis actions.

```
listen for: Listen for a spoken phrase
listen for a list of string -- list of possible phrases to listen for
    [with prompt string] -- text computer will speak as a prompt
    [giving up after integer] -- how many seconds to wait before giving up
    [filtering boolean] -- whether to skip phrases with special characters
Result: string -- the recognized phrase
```

Listen for provides complete interaction functionality providing both a text-to-speech prompt and list of phrase items to recognise. The list of phrases constrains the language model to a limited number of possible responses. This helps improve the accuracy of the recognition task but also limits the flexibility of the spoken reply.

The number of seconds allowed for the user to respond can be defined to avoid circumstances of waiting indefinitely for a response or until the next sound is detected.

There is no control over which voice is used to play the prompt. The voice that is currently set in the Speech Control Panel is used for all feedback.

```
on run
    set theApplication to "Speech Listener"
    set speechListenerPath to ((path to scripting additions as string) & ¬
theApplication)
    using terms from application "Speech Listener"
        tell application speechListenerPath
            listen for {"e-mail", "fax", "word processor document"} ¬
with prompt ¬
"Would you like to create an e-mail, fax or word processor document?" giving
        up after 10
            end tell
        end using terms from
    end run
```

If the phrase is not recognised, the string returned by the *listen for* command is '???' and an error is thrown (code -5120). As an example, if the speaker said telephone call instead of e-mail, fax or word processor document, an error would be returned by the *listen for* command. Such errors should be captured by a *try/on error* statement block and handled so that the speaker has another opportunity to say one of the expected phrases. If the error is not handled then the script will exit with an ungraceful execution error dialog (Figure 3).

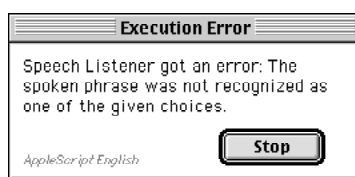


Figure 3 Execution Error dialog box

If the user does not respond within the specified time out period this can be detected using the *try/on error* mechanism available in AppleScript. To detect the timeout you will need to detect the error code -1712.

```
repeat
try
    tell application "Speech Listener"
    ...
    end tell
    exit repeat
on error number error_number
    if the error_number is -1712 then
        say "You did not respond. Please try again."
    Else if the error_number is -5120 then
        say "I did not recognise what you said. Please try again."
    Else
        -- something else and undocumented happened
    end if
end try
end repeat

say: Speak the given text
say anything -- the text to speak, which can include intonation characters
[displaying string] -- the text to display in the feedback window (if different).
```

```
-- Ignored unless Speech Recognition is on.  
    [using string] -- the voice to speak with  
    [waiting until completion boolean] -- wait for speech to complete before returning (default is  
true).  
-- Ignored unless Speech Recognition is on.
```

Text-to-speech (TTS) output of any string can be produced using the *say* command. In its most simple form, a single parameter consisting of the desired text to be spoken is provided. The default behaviour is to output the text string using the default voice and display the text string in the feedback window.

```
tell application "Finder"  
    say "Any string may be spoken using text-to-speech"  
end tell
```

Should the feedback window be different from the text string, a specific display string can be defined as one of the parameters using the *displaying* parameter.

The speech synthesis manager offers control over various parameters to help fine tune the naturalness of the speech and correct mispronunciations. Prosodic and phonemic controls are available to change the emphasis and pronunciation of text-to-speech output [3].

```
tell application "Finder"  
    say "iMac"  
    say "[[emph -]] eye Mac"  
    say "[[inpt phon]] _1AYm=1AEk"  
end tell
```

Design Considerations

There are many factors that will influence the success of an interactive speech application that require attention to human interface issues, software engineering and usability testing. The focus in this paper is “question-answer” style dialogues but this may not be the only way to deploy a speech application. Other elements may be incorporated into the application that provide visual cues in addition to a verbal prompt.

Questions to consider at the beginning of the design are the target audience and the frequency of use of the application. Novice users may require considerably more assistance to navigate through a speech driven system, especially since this mode of input is a relatively new technology. More experienced users may benefit from shorter prompts to enable faster navigation through the system. For applications that are not used often, users may not remember the correct phrase in the list of available options, they may remember the concept but the exact word may temporarily escape them. Some form of verbal assistance may be necessary to provide feedback to list all possible options.

The success of the Macintosh GUI has been based on consistency of user interface between applications. Similar rules of familiarity will be of benefit for speech applications. An example, always include a graceful exit strategy that allows the user to say phrases that are consistent with the visual environment such as “cancel” or “quit”. Provide a “help” option that provides users with information specifically relating to the current prompt. Other possibilities include navigation actions typical of web browsers such as “go back” or phrases that are application specific.

The prompts presented should guide the user to say a phrase from the expected set of responses. In some cases, the desired response could be incorporated into the question. It is important to remember that delivery of the prompt will influence the user response. Minor variations to the wording of the prompt can produce different outcomes.

In other situations reading all the possibilities may not be practical. In cases where reading back a list of options in the prompt is deemed necessary, consider the frequency of use and whether experienced users would dislike having to review these items. Another design issue to explore is whether some questions could be broken down into a series of smaller, perhaps simpler questions.

Phrases may be non-unique so be prepared to allow synonyms for some branches of the menu tree. Be cautious of similar sounding option phrases as this will reduce the reliability of the system as one option may always be confused with another.

Short phrases or one-word responses should be used to reduce the complexity presented to the user. If response phrases contain several words then users may replace or swap one or more words in their response. The recognition system reports exact matches to the phrases. Variations are reported as an error or incorrectly recognised.

Design Process

The design of a speech application could be considered similar to designing a form-based process although the flow is constrained to be linear rather than spatial. A good starting point is to use an *n-ary* tree structure to describe the speech transaction sequence. Each node corresponds to a voice prompt and each branch to a spoken option. In the deployed application, each branch will have an associated action that records some result (Figure 4).

Once several iterations of the overall flow have been considered global functionality such as help and exit strategies can be included. Help and graceful exit should not impact on the overall dialogue flow.

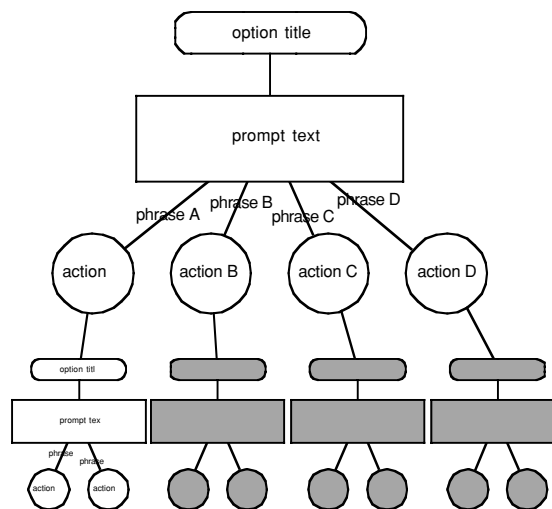


Figure 4 Tree-structure representation of dialogue.

The following pseudo AppleScript implements the expanded level of the tree in Figure 4.

```

set gSpeechListenerApp to ((path to scripting additions as string) & "Speech Listener"
...
-- option title 1
set thePromptText to "Question to ask user"
set theOptions to {"phrase A", "phrase B", "phrase C", "phrase D" }
using terms from application "Speech Listener"
tell application gSpeechListenerApp
    set bResponseOK to false
    set theAnswer to ""
    repeat while bResponseOK is false

```

```

        set bSuccess to true
        try
            set theAnswer to ¬
            listen for theOptions prompt with thePrompt ¬
giving up after 10
            on error errorString
                -- utterance did not match anything in list theOptions
                set bSuccess to false
            end try
            if bSuccess is true then
                exit repeat
            end if
        end repeat
    end tell
end using terms from
if theAnswer = “phrase A” then
    tell me to doAction_A()
else if theAnswer = “phrase B” then
    tell me to doAction_B()
else if theAnswer = “phrase C” then
    tell me to doAction_C()
else if theAnswer = “phrase D” then
    tell me to doAction_D()
end if

```

In the previous example, global functions such as exiting the script by saying “cancel” or requesting help have not been shown. Changes to this script would require adding the exit and help phrases to the options and handling them in the same way that the actions have been handled.

Most nodes will have similar structure requiring a main text prompt and options to listen for. In addition, more detailed prompts could be added that explain to the user the options that they are able to say. These could be presented when the user has not provided an expected utterance. Global help and exit phrases should be appended to the options. This common structure could be implemented using object-oriented principles provided by script objects. Each node could inherit the global functions from a parent script.

Possibilities

With the current technology, directed dialog applications can be developed that use a bounded set of responses for a given set of questions. Speech in conjunction with AppleScript can be used to develop a replacement or an enhanced interface for “wizards”. Wizards usually consist of a series of dialog boxes that allow the user to type a limited amount of information or select from known set of parameters. Saying a word or phrase from a closed set of options is straightforward using the *listen* command under AppleScript.

Multiple choice style questions can be developed that allow the user to answer with a simple phrase instead of saying “A”, “B”, “C” or “D”¹. A speech interface for multiple choice questions could be complemented with a visual clue of what to say would help restrict the user’s response (see example in Appendix).

When combined with an interface that displays images, applications can be developed that allow simple navigation of kiosk style technologies. Such an interface is analogous to the early text based adventure games that were developed for the Apple II or simplified interactive books as a more recent example. The most challenging part will be to design an interface that makes the audience aware that these technologies are speech enabled.

AppleScript has been recognised as a significant productivity technology that enables the control of many aspects of the MacOS and Apple-event aware applications. This has created opportunities for inter-application communication and automation of repetitive tasks.

Speech control of AppleScript provides a mechanism to initiate and control such operations. Examination of the standard set of Speakable Commands will indicate some of the other applications.

Limitations

The system is not as robust as human speech communication and cannot deal with many of the situations that human listeners would take for granted such as high levels of background noise. Some of the limitations remain technology hurdles that will eventually be overcome. In the meantime, it is important to educate users of the limitations of the technology to avoid frustration by setting overly high expectations.

The first point to make is that the speech recognition system is not a dictation system and cannot respond to arbitrary speech utterances. The current focus is for command and control applications. The system not designed for natural language queries.

While text-to-speech is being played, speech will not be recognised when recognition is in continuous mode. This means that when a prompt is being presented, the user cannot anticipate the prompt and answer the question before the end of the prompt. This is a good case for keeping prompts brief. However, if push to talk is enabled, the user can interrupt the prompt by pushing the appropriate key (the default is the Escape key).

The speech samples are derived from North American English accents for adult speakers. Other accents, such as Australian may not be recognised with high levels of accuracy. The system will not work as well with children's voices.

Similar sounding words can be easily confused so choose phrases and words that are meaningful to the user but also sound distinct. As an example choosing "crochet" and "OK" as possible answers would not be recommended. Even "no" and "none" seem to be confused for Australian accents.

Speech recognition does not work well in dynamically noisy environments. If other people are talking in the background or opening and closing cupboards system performance will be degraded. The system does perform some level of adaptation to the background noise but is only suitable for constant noise conditions such as consistent air conditioning background noise. Don't try to use speech recognition with a CD playing in the computer.

A PlainTalk microphone is very important. Other microphones are not recommended ⁱⁱ.

Say does not allow a different feedback voice to be used if speech recognition is active. The feedback voice that is currently set will override subsequent attempts to change the voice using the *say...using voice* command.

Pronunciations are derived from an internal dictionary and for cases like proper names may not match expectations. "iMac" is a good example. Use the voice playback function in SimpleText to verify pronunciations. This is useful for explaining why some Australian English utterances are poorly recognised. For instance, "tomato".

Limit the number of phrases to listen for. The performance of the system will degrade for large numbers of options.

Testing

Usability testing of the system using a number of different speakers will reveal problems with the structure of the dialogue, the scope of the response phrases and the accuracy of recognition. Some of these aspects can be identified prior to coding by simulating the system in the form of interviews or role-playing with potential users, independently of a speech recognition system. Such activities can provide the extent of the wording of questions and expected phrases for specific problem domains.

Test scripts can be easily generated that assess the level of confusion between the set of option phrases. This will provide an indication of which phrases are well recognised and those that are not. The results of these experiments should suggest other phrase combinations to try.

In addition to the speech side, scripts that perform the actions can be developed and tested independently of the speech application. Action functionality should be contained within subroutines for modular construction.

Like any software development process, test often with potential users and observe their reaction to the dialogues and solicit their comments about the system.

Other New Features

Two other features are noted here but are not discussed at length. Provided that user interface elements such as check boxes, radio buttons and standard buttons have been implemented in a conventional manner, users can speak the name associated with the element. For instance, the buttons on the alert that prompts the user to save a file before closing can be spoken (Figure 5). In this case, the user can say “Computer, save”, instead of clicking on the default button.

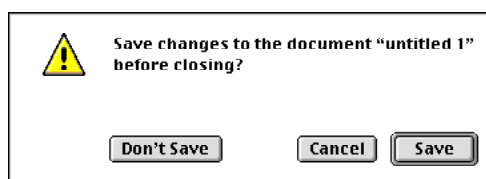


Figure 5 Standard alert where buttons are speakable

Techniques are available to develop custom animation sequences that can be installed into the feedback window. Details of how to create and add these animations will be described in a forthcoming Apple technical note.

Conclusion

With AppleScript, interactive speech applications can be quickly implemented, tested and deployed. This means that a majority of functions available through the Finder and many other Apple Event-aware applications can be controlled by voice. Speech dialogues require a different approach compared with screen layout. By taking into consideration some simple guidelines, applications can be designed that allow users to speak responses to directed questions. The conversations cannot be unconstrained so a significant part of the design process must define the scope of the options and assess how to handle unexpected utterances.

The success of the desktop environment has been based on the consistency of the “look and feel” of the GUI. One of the design challenges for speech is to provide an enhanced user experience that extends the functions currently available from a point and click interface.

See Also

There is a new AppleScript Guide Book for “Scripting Speech” available at <http://www.apple.com/applescript>

References

- [1] Pallakoff, M. and A. Reeves, “The Speech Recognition Manager Revealed”, develop, issue 27, April 1999.
- [2] AppleScript Language Guide, Addison-Wesley Publishing Company, 1993.
- [3] Apple Computer Inc., “Speech Manager”, Chapter 4, “Inside Macintosh: Sound Speech Manager, 1996.
- [4] Wall, L. and M. Neeracher, MacPerl 5.2.0r4, April 1998.
- [5] Wall, L and R.L. Schwartz , “Programming Perl”, O’Reilly and Associates, Inc, California, 1991.

Endnotes

- ⁱ Short utterances such as “A”, “B”, “C” and “D” are not wise choices since they can be easily confused by speech recognisers.
- ⁱⁱ Universal Serial Bus (USB) microphones maybe be supported in the future.
- ⁱⁱⁱ Experience with this interface will demonstrate that “Don’t Save” is not as well recognised as “Save” and “Cancel”.

Appendix

The following example illustrates uses a simple question that has four possible responses. To assist in user feedback, PictureViewer is used to display a visual representation of the valid response words.

```
set theGraphicsPath to "replace with path to folder containing jpegs" – use colons to separate folder
names
set allPics to theGraphicsPath & "all.jpg"
set theBaseballPic to theGraphicsPath & "baseball.jpg"
set theCricketPic to theGraphicsPath & "cricket.jpg"
set theSoccerPic to theGraphicsPath & "soccer.jpg"
set theTennisPic to theGraphicsPath & "tennis.jpg"
set thePrompt to "Which sport would you like to select?"
set theNoResponse to "You did not respond"
set theNoUnderstand to "I did not recognise what you said. Try again."

-- display the picture
ShowPicture(allPics)
set theApplication to "Speech Listener"
set speechListenerPath to ((path to scripting additions as string) & theApplication)
using terms from application "Speech Listener"
    tell application speechListenerPath
        repeat
            try
                set theResult to listen for {"baseball", "cricket", "soccer", "tennis"} –
                    with prompt thePrompt –
                    giving up after 8
                exit repeat -- successful
            on error number theErrorNum
                -- check for no response
                if theErrorNum = -1712 then
                    say theNoResponse
                else if theErrorNum = -5120 then
                    say theNoUnderstand
                else
                    say "Error " & (theErrorNum as text)
                    tell me to ClosePicture()
                    return "Unhandled error"
                end if
            end try
        end repeat
        tell me to ClosePicture()
        set theImage to theGraphicsPath & theResult & ".jpg" -- using the result to formulate
the response
        tell me to ShowPicture(theImage)

        say "You have selected " & theResult
        delay 2
        tell me to ClosePicture()
    end tell
end using terms from

on ShowPicture(thePicture) -- Display the picture using PictureViewer
    tell application "PictureViewer"
        activate
```

```
        open the file thePicture
    end tell
end ShowPicture
on ClosePicture()-- Remove the picture from view
    tell application "PictureViewer" -- PictureViewer does not understand window events
        quit
    end tell
end ClosePicture
```