Software Toolkits for Supporting Voice in the Workstation

Barry Arons

Conversational Interface Consultant
360 Forest Avenue, #202
Palo Alto, CA, 94301

MIT Media Lab (after 9/1/90)
20 Ames Street
Cambridge, MA, 02139
+1 617-253-0300

Abstract

This paper explores the design of toolkits for handling voice and audio in a multimedia workstation, and how this toolkit will interact with the workstation's audio and window systems. It draws parallels from window systems and their graphical toolkits, as well as from the design of software-based audio servers. A set of voice presentation objects that can be used by a variety of interactive voice applications are also suggested.

Introduction

Support for interactive audio is required on new personal workstations entering the marketplace. Output capabilities range from low-quality beeps on IBM PC clones to high-fidelity audio on an Apple Macintosh. Sun SPARCstations provide telephone quality voice I/O, while NeXT machines also produce compact disc quality sound. Each workstation on the market has its own variety of audio hardware and an idiosyncratic software interface that discourages development of interactive applications that span hardware and software architectures.

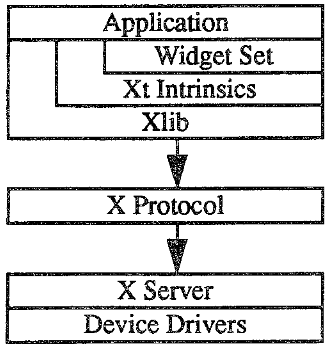
Device-independent application programming interfaces (APIs) are available for some environments, but this alone is not enough to make voice in the workstation widely supported by third-party software developers. A software server is superior to a conventional library for supporting a wide range of Desktop Audio applications and services as it allows a variety of resource sharing schemes. The VOX Audio Server, for example, provides a flexible network-transparent solution for integrating voice into the workstation, allowing multiple applications to share and configure audio processing modules. While such a server is flexible, it can be tedious to program, and therefore a simpler software interface is desired.

It is possible to build a variety of higher level software interfaces on top of an audio server, a subroutine library, or even a set of device drivers. This paper investigates a "toolkit" approach to designing such an interface on top of an audio server.

Graphical Toolkits

A toolkit, in the window system sense, is a high-level software interface for graphical interactions that also provides a set of user interface abstractions to applications. Toolkits hide the details of input, event, and screen handling from the application, allowing the programmer to develop an effective user interface without being concerned with tracking the mouse or updating the screen. Equally as important, a graphical toolkit provides a set of commonly used presentation objects such as scroll bars, text editors, selection menus, and command buttons.

For example, the software hierarchy of the X Window System is shown in the figure below. The X window server is directly accessible to applications through a library (Xlib). A toolkit (Xt Intrinsics) is built on this library, and several sets of presentation objects or "widget sets" are built on top of the toolkit. Note that a similar architecture exists for proposed audio servers and audio toolkits.



A toolkit is similar to a subroutine library, and is usually implemented as one. A toolkit provides functionality beyond a conventional library including memory management, a comprehensive set of options with default values that can easily be overridden at run-time, and mechanisms for input, output, and event handling. The X toolkit is built on an object-oriented framework with simple base classes and multiple levels of inheritance. This form of development environment (such as provided by C++) makes it exceedingly easy to create new specialized objects with properties inherited from existing objects.

The handling of input and output events are particularly important in a workstation environment with the potential for multiple voice applications to be active simultaneously. Unlike the single process nature of the MS-DOS and Macintosh operating systems, Unix provides true multitasking capabilities. The X toolkit is structured to gather input from the user and asynchronously call specific routines in the application program. A multithreaded applications environment is desired so that the toolkit and applications can be written without the use of complex dispatching routines or large numbers of very simple callback routines.

Audgets

In X Window System terminology, a "widget" is a graphical user interface object. The term "audget," for *aud*io wid*get*, will be used in this paper to represent an audio interface object analogous to a graphical widget.

For many applications the flexibility afforded by an audio server greatly increases the complexity of the application program. A toolkit built on top of an audio server provides high-level abstractions to hide the intricacies of the server. An audget set predefines common audio configurations, such as players, recorders, speakerphones, or even a complete conversational voice response system, to simplify application programs and to provide a uniform user interface.

Until a fully functioning audio server and a range of applications are developed it is difficult to envision a proper base set of voice interface audgets. However, from experience with audio server designs, and previous interactive voice-based applications, an initial set of audgets is proposed. Presentation audgets include interfaces to digital recorders, sound editors, or voice menus. Note that many of these audgets will have graphical interfaces in addition to their audio interfaces. A graphical interface to an audget, however, should be completely isolated from its voice interface so that a single audget can be used with more than one graphical widget set.

Look and Feel

A variety of widget sets are built upon the X toolkit (e.g., Athena Widgets, Open Look Widgets, Motif Widgets). These widget sets provide similar functionality, as they all contain a common set of user interface objects—scroll bars, buttons, text editor windows, etc. However, they all have a different "look" to their graphical presentations and a different "feel" to their interaction semantics. What is the audio domain equivalent to the look and feel of a graphical interface?

A pure voice interface would use only speech input and output, eliminating keyboards, mice, and graphics displays. A conversational user interface paradigm can be based on a keyword vocabulary that is always active, or a context sensitive vocabulary that is changed dynamically by the state of the application. The exact vocabulary, phrasing of system prompts, and the overall interaction semantics thus would embody the "feel" of an audio interface. The specific mapping of keystrokes and voice commands to actions in an application is a critical component of the feel of a user interface. The speech quality, timbre, pacing, and rhythm of a voice interface are perhaps analogous to the "look" of a graphical interface.

Voice Menu Audget

Menus are commonly used in voice mail and interactive voice response systems. A voice menu is similar to a graphical pull-down or pop-up menu; it presents a set of choices to, and accepts input from, the user. Menu items are presented with recorded or synthesized speech rather than in textual form, and input comes from a speech recognizer or telephone keypad instead of a mouse or keyboard.

The programmatic interface to a voice menu minimally consists of a list of text strings that represent the desired voice prompts. A voice menu audget, however, also can be set up to handle more sophisticated interactions. It can accept additional lists of strings for second or third round of prompts if the user fails to respond. The audget can have a built-in help facility that is derived directly from the prompt strings and the menu semantics. The audget also may be customizable as to how the interactions are structured so that a variety of user interfaces can be rapidly prototyped.

An application can create a multimedia interface by encapsulating both a voice menu and a graphical menu. A "multimedia menu" created in this fashion could simultaneously generate voice and graphical menus, and accept multimodal input from a variety of sources. Note that while such a menu is highly desirable for many applications, audio will always remain a different medium than graphics. Some types of menu items, such as graphical texture patterns or musical motifs, cannot be converted from one form to the other. Even the translation of simple text menus to the audio domain may be difficult. It is not practical, for example, to have a user respond to a list of 17 synthesized items. A menu audget can, however, be programmed to break a list into smaller components, and present them as a set of two or three smaller voice menus.

Synthesis Audget

A synthesis audget can put an insulating wrapper around a core text-to-speech engine. The toolkit adds a higher level software interface to functions such as user defined exception dictionaries, or the handling of touch tone events in interactive voice response systems. An audget may combine a text-to-speech synthesizer's indexing function (indicating the sythesizer's current location), touch tone input handling, and a set of application defined callbacks into a concise audget routine. The application writer is thus freed of low-level interrupt and device handling, and can concentrate on writing highly interactive telephone-based applications.

Extensible Audgets

Other audgets of interest include interfaces to stand-alone players and recorders. A recording audget, for example, can notify the application after reaching a maximum length, upon detection of silence, or upon receiving other user input. The application controls the audget—it can set a file name associated with the recording, pause the recorder, or set the silence detection sensitivity. The audget can be queried for its current status, position within a file, recording parameters, etc.

A basic record or play audget can be extended via the inheritance mechanism to operate on a sequence of sounds rather than individual files. Such an extension may increase the interactivity of the application and simplify the applications programming interface to the audio system. The audget can be programmed to analyze the pending events in its output queue, and pre-fetch the next sound file while another event is active, to reduce latencies in the operating system or audio system.

Editor Audget

Applications often need to manipulate and edit audio buffers or sound files. The application may view editing as high-level cut, copy, and paste functions, while internally this is accomplished through the manipulation of linked lists of pointers to audio buffers, reference counts, etc. An editing audget manages these internal data structures, and provides a device-independent interface to a range of sound formats and encoding styles. A single editing audget thus operates on highly compressed speech signals as easily as digitally encoded music.

By merging simple audgets, it is possible to create an audio only, or a combined audio and graphical, interface. With a sound editing audget, it is possible to create a variety of user interfaces to an audio editor. Editing functions can be controlled by touch tones, mouse interactions, or possibly voice commands. The graphical interface can be a mouse-based selection mechanism, or keystroke-oriented if integrated with a conventional keyboard-based text editor. The audio editor audget can be connected to a range of graphical widgets to create interactive editors with distinct looks and feels to blend into the existing window system user interface paradigm.

Summary

The following points summarize many of the important design goals and current software engineering practices needed to develop software toolkits for supporting voice in the workstation:

1) Design and use audio servers and audget toolkits.
2) Keep the audgets separate from the graphical widgets.
3) Support a well-defined object system and multithreaded applications.

This paper raises as many questions about toolkit design as it answers, but it is intended as a thought experiment to lead future research and development efforts in this area. The development of window system toolkits has accelerated the pace of applications development by allowing programmers to quickly and efficiently harness the power of the graphical interface. Toolkits for audio, and easy-to-use user interface audgets, should help bring about a similar revolution in the design of future voice interfaces.

Acknowledgments

Many of the ideas presented in this paper evolved from lively discussions with Don Jackson's Audio Group at Sun Microsystems and the founders of Arosa Systems, Inc. Special thanks to Wayne Yamamoto, Chris Schmandt, and Rosanne Wagger for their support in the production of this paper.

References

B. Arons. Desktop Audio Applications for a Workstation-Based Voice and Audio Server. In *Proceedings of the 1989 Conference of the American Voice I/O Society*, Sep. 1989.

A. Nye and T. O'Reilly. X Toolkit Intrinsics Programming Manual. O'Reilly & Associates. 1990.

R.W. Scheifler and G. Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79-106, Apr. 1986.