Y-Windows: Proposal for a Standard AUI Environment

Martin Kaltenbrunner

Music Technology Group, IUA – Universitat Pompeu Fabra
Passeig de Circumval·lacio 8 – 08003 Barcelona, España
mkalten@iua.upf.es

ABSTRACT

This paper introduces a draft framework for a shared auditory user interface (AUI) environment. Y-Windows, similar to the approach of the X-Windows GUI framework [1] in the Unix world, aims to provide common functionality for the easier development and design of AUIs. This initial publication of these ideas, originally roughly developed within a diploma thesis [2], should encourage researchers and developers from the auditory interfaces community to contribute to the further development and possible future implementation of this concept.

1. INTRODUCTION

Today there exists a variety of libraries, APIs and applications ([3],[4],[5]) focused on the development of auditory interfaces, there even exist complete auditory desktop environments [6]. While each of these components provides its specific functionality it is often impossible to incorporate them together within a single application – because of their exclusive use of the sound hardware for example or simply their monolithic design. During his initial work involving the design of auditory user interfaces [7] the author noticed a lack for a common framework allowing these components to work together.

The Y-Windows concept therefore was designed to fill in this gap. Usually GUI application developers don't have to spend their time implementing basic interface design concepts, because they are already provided with a variety of libraries which simplify the construction of an average GUI. Only few GUI-developers might still attempt to develop the code for their buttons or progress-bars "manually", but this is still the case for their equivalents in the AUI world. Therefore the implementation of some AUI widget libraries will be one of the tasks for the Y-Windows project.

Due to the known limitations in audio programming there is also a need for a central instance which manages the resources for various applications requiring simultaneous access to the audio hardware. In the Unix world there already exist several so called *sound servers* which have been designed exactly for that purpose. Such a sound server, extended and optimized for the special AUI requirements, will be the central component of Y-Windows.

While the examples in this document are mainly borrowed from the Linux platform context, Y-Windows ideally should be platform-independent and network–transparent. An application developed for Y-Windows should be easily portable to - or executable on - any platform implementing the framework. One possible approach to achieve this is the use of the Java programming language or easily portable code for the high-level interfaces, while using optimized native code for lower layers.

2. GENERAL OBJECTIVES

The general objective of this project is to create an adequate development and operating environment for primarily pure acoustic user interfaces. This adds an additional *auditory mode* to the existing *text* and *graphical modes* of current operating systems. Linux users might be familiar with the concept of run-levels, where the operating system boots directly into such an operating mode, therefore a pure auditory mode is easily to implement for those systems. Of course, the Y-Windows system should also be usable for multi-modal interfaces if needed.

Such a system must provide a *shared environment* where multiple auditory applications can share simultaneously the audio hardware resources and also can exchange audio data with each other. In the simplest case this should allow additional audio playback in the foreground while for example an acoustic background monitoring task is running. On the other hand it should be possible to record audio from the microphone while a speech recognition process is listening to the same port. And finally several independent applications should be able to use the in- and outputs of others just like ordinary devices. This routing functionality also should allow the construction of application chains out of basic components.

A central engine should provide basic audio signal processing functions which can be easily used by the connected applications, without the need of redundant re-implementation of known concepts. This *rendering engine* should provide optimal implementations of basic sound generators and filters and other similar components equivalent to generic graphical operations for visual interfaces. Additional rendering tasks such as advanced sound, music or speech synthesis and advanced signal processing should be realized as plug-ins and therefore as extendable or replaceable parts of this engine.

Graphical operating systems already come with the necessary libraries for the creation and execution of standard conformant GUI applications. This guarantees an uniform look-and-feel and handling of all different kinds of applications. The widely used classical desktop metaphor also allows inexperienced user the quick understanding of the interface principles. Therefore Y-Windows has to create a common *hear-and-feel* by providing common acoustic user interface elements which stay consistent over all applications using the provided AUI libraries. Those libraries and interfaces not only have advantages for the developers of auditory applications, they are also crucial for the actual users of the framework which are not forced to get used to various interface metaphors while using different applications. The adequate metaphor for such an auditory "desktop" environment is still missing though.

3. BASIC DESIGN

According to these requirements the Y-Windows approach is roughly split into four major layers: A hardware layer should define an abstracted interface to the various parts of audio hardware. The central layer – the Y-Server – provides the shared acoustic workspace and performs the basic rendering of virtually all acoustic interface elements. A third level allows abstract access to the rendering layer through a collection of libraries providing speech APIs, synthesis APIs and so on. Additionally it should provide libraries for higher level AUI elements, such as parameterized auditory icons and common auditory widgets. Finally the actual auditory applications using the Y-Windows server and its libraries are part of the fourth layer. A vertical direct rendering interface (DRI) as shown in the diagram additionally allows direct access to all the lower levels for those applications which rely on time-critical operations.

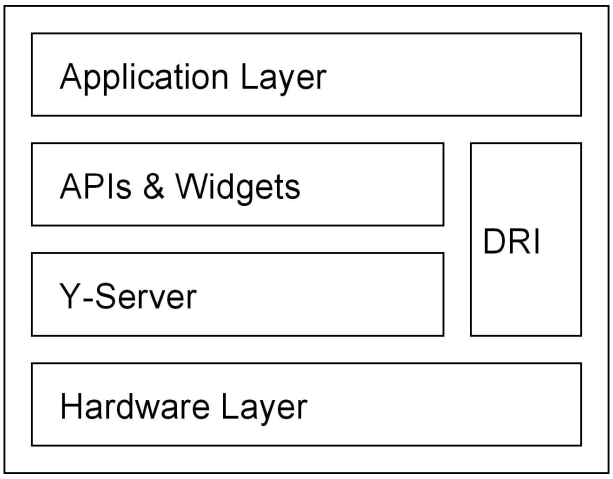

Figure 1. *Basic Y-Windows Structure*.

3.1. Hardware Layer

This lowest-level layer mainly intends to create a common abstract interface to the various audio hardware. This not only should include the typical sound cards but also should consider all additional hardware relevant for auditory user interfaces:

- internal, external sound cards
- MIDI equipment
- hardware speech synthesizers
- 3D speaker systems
- head trackers
- additional multimedia hardware

Projects such as PortAudio [12] already provide such a common interface for the different audio programming interfaces such as OSS, ALSA, CoreAudio, DirectSound etc. which we know on the various operating system platforms. It has to be evaluated if such a system conceptually can be extended for additional hardware as listed above. Although the implementation of this lowest layer is not really crucial for the overall concept, it generally improves the portability of the higher layers, specially the server component.

3.2. The Y-Server

The Y-Server is the central instance of the Y-Windows architecture. It is in charge of the generation & rendering, composition & management and finally the actual display of the complete auditory scene.

As already mentioned above, the server process should already internally provide some basic built-in rendering operations, such as simple waveform generators and the most common filter operations, providing fast access to simple sound synthesis methods. A modular design and plug-in architecture should allow the versatile extension of this rendering layer with additional components. This modular approach permits the easy replacement of specific components, such as different brands of speech recognition or synthesis engines for example. It should also encourage the co-existence of both commercial and open-source components. A collection of standard plug-ins should include at least some of the following functionality:

- advanced sound and music synthesis, such as spectral or physical modeling
- speech synthesis
- further filters and sound effects
- 3D sound spatialisation

The second major server task is the connection and management of the various applications, which are simultaneously using the sound server infrastructure. This basically includes the routing of the audio in- and outputs of all the connected components, which concatenates chains of internal components, external plug-ins and actual applications. All this components should be able to actuate equally as sound generators, filters and consumers within this system.

Finally, the server has to render the actual output of the complete auditory scene. While the output stream of a single task, which is simply played back via the speakers is no difficult issue, the rendering and presentation of multiple streams of independent applications could use a spatial display using headphones, where the various applications are placed at different positions within the virtual user space. Other possible final presentation methods to distinguish multiple tasks could be different volume levels or finally applied sound effects which should be definitely part of this presentation layer.

The Y-Server as well manages the various user input by providing features such as continuous speech recognition, speech command and control, or DTMF decoding for the use in telephony systems. Additional features could add authentication methods such as speaker recognition.

3.2.1. Evaluation of existing sound servers

The implementation of the Y-Server most likely will follow the design of already existing sound-server solutions. Possible candidates are *aRTs* [8], a sound server and synthesis engine used by the *KDE* desktop, and *Jack* [9], a relatively new sound-server optimized for real-time performance with advanced routing features. aRTs also provides network transparent operation via MCOP, a CORBA-like interface adapted for multimedia. Although aRTs already provides more than the core features one would expect from a versatile sound server, the concept of Jack promises better performance: Instead of using a communication protocol, applications for Jack plug-in directly into the server engine using a callback interface. It has to be evaluated if the two approaches can be combined and extended with the most important features required for the Y-Server, such as speech recognition, advanced sound synthesis and transformation, or sound spatialisation. The lean and fast design of Jack is quite appealing, and offers better performance and expandability over the rather large and slow aRTs server.

There also exist several other sound server solutions for the Unix platform, such as the Enlightenment Sound Daemon [10] used by the GNOME desktop, or the Network Audio System [11], which focuses on network transparency for the use of audio in X-Terminals. It is remarkable though that there exist that many solutions for the same problem, which obviously all don't seem to satisfy the basic needs for such a system. Since too many sound-servers raise the same problems as concurrent audio applications, .it is necessary to find a common sound server solution, which is generally accepted.

3.3. Interface Layer: Widget Libraries & APIs

Based on the possibilities offered by the Y-Server, which provides the pool of basic technologies necessary for the rendering of an auditory display, the interface layer enables the access to this rendering layer through a series of libraries. Regarding the modular architecture of the Y-Server, there is also a need to define a set of common programming interfaces (APIs) in order to provide the same interface for the different possible modules. This then enables the replacement of the speech synthesis engine manufacturer for example, while maintaining the same interface. Existing APIs, such as the Linux Audio Plug-In Architecture LADSPA [13] should be included, where on the other hand still not existing APIs, such as a common Speech API need to be developed entirely from scratch.

Additionally, this layer should implement or adapt interface libraries of the various existing basic auditory design elements, such as parameterized auditory icons [14], Earcons [15], common auditory interface widgets (acoustic progress-bars [16], etc.) or further experimental designs. Using these basic components, higher level embeddable components such as sonification tools or voice browsers can be constructed. With a growing set of those tools brought together in a collection of AUI widget libraries - which then can be extended or modified - developers of auditory applications are not forced to invent the wheel again and again.

Basically such widget libraries need to provide suitable solutions and templates for the most common dialogs in interactive auditory systems. This should include basic menus, forms and even some direct manipulation interfaces. On the output side this should provide known sonification and data auralisation methods in order to facilitate auditory feedback design. This could also include some algorithmic composition methods to exploit music for auditory feedback as well.

Another crucial task is the choice or definition of the required communication protocols between Y-Server and applications, the required file-formats and interface construction languages. For the speech interface components, there already exist approaches such as VoiceXML [5], which might be also extended for the use in not purely speech based auditory interfaces. Similar XML based formats might be suitable for most of the other interfaces.

4. IMPLEMENTATION & APPLICATIONS

An initial implementation of the Y-Windows framework should be realized on the Linux platform, mainly because of two complementary reasons: First of all Linux already offers a variety of open-source components, which can be modified for the use within this environment. On the other hand Linux still lacks of some common audio, multimedia and speech APIs, which could be developed as part of this project. Of course the framework should be ported to other mainstream operating systems such as MacOS or Windows as well, where it can adopt native interfaces such as QuickTime or DirectX for an initial implementation.

The first development steps need to focus on the refinement of the Y-Windows concept itself. Then the major requirements and tasks need to be defined, which includes the definition of the central API interfaces and communication protocols and a prototype implementation of the Y-Server.

The main application areas of the Y-Windows framework in its initial phase might be basically the research & development of auditory interfaces themselves, since it should allow the faster prototyping of test setups. From a user's point of view though, the main application areas will be auditory interfaces for mobile devices, acoustic monitoring applications, or auditory desktop applications for the sight-disabled. Especially mobile devices, such as PDAs or SmartPhones with their limited screens and keyboards would profit definitely of such a scalable system. The pure acoustic nature of the framework might also be interesting for various embedded applications within industrial or home devices, which increasingly are designed using the embedded Linux platform. Of course, the framework can also be used for the advanced acoustic enhancement of GUI desktop applications.

A possible future distribution of the Y-Windows framework should also include a set of demo applications such as a voice-browser, sonification tools, acoustic background monitors and an auditory "desktop" shell. While such applications initially will be probably mainly reference implementations for developers, the final goal should be the creation of a complete auditory environment based on the Y-Windows framework.

5. CONCLUSIONS

The Y-Windows concept is far from being completely thought through and there are obviously several odds and ends in some details of its design, but it is mainly meant to initiate a discussion in that direction. The advantage though of an implementation of such a framework would be the creation of a common pool of current knowledge in AUI research and design, allowing the easier access to and extension or improvement of this knowledge. The author therefore hopes that some researchers and developers from the auditory interfaces community will join this effort

Together with the presentation of this article there will be created an open-source project page in order to provide the necessary collaborative tools for this task. This includes the installation of a mailing list, CVS code repository and additional web-based documentation including the further development of this concept paper.

6. REFERENCES

[1] Open Group Inc.: X11R6 URL, http://www.x.org/last_release.htm

[2] Kaltenbrunner, M.: "Auditory User Interfaces for Desktop, Mobile and Embedded Applications". Diploma Thesis, FH Hagenberg, 2000

[3] Cook, P. & Scavone, G.: "The Synthesis Toolkit", URL, http://ccrma-www.stanford.edu/software/stk/

[4] Sun Microsystems: Java Speech API v1.0. URL, http://java.sun.com/products/java-media/speech/

[5] VoiceXML Forum, W3C: VoiceXML 2.0 Specification, URL, http://www.w3.org/TR/voicexml20/

[6] Raman, T.V.: "Emacspeak – A Speech Interface", Proceedings of the Conference on Human Factors in Computing Systems. p66ff, ACM Press, 1996
URL, http://emacspeak.sourceforge.net/

[7] Kaltenbrunner, M. & Huxor, A.: "Marvin: Supporting Awareness through Audio in Collaborative Virtual Environments", In: Earnshaw, R. & Vince J. (eds.): "Digital Content Creation" p294ff, Springer Verlag, Hamburg 2001

[8] aRTs, analog realtime synthesizer, URL, http://www.arts-project.org/

[9] JACK audio connection kit, URL, http://jackit.sourceforge.net/

[10] Enlightenment Sound Daemon, EsounD. URL, http//www.tux.org/~ricdude/EsounD.html

[11] Network Audio System, URL, http://radscan.com/nas.html

[12] PortAudio – An Open-Source Cross-Platform Audio API. URL, http://www.portaudio.com/

[13] Linux Audio Developer's Simple Plugin API, LADSPA, URL, http://www.ladspa.org/

[14] Gaver, W.: "Synthesising Auditory Icons", Conference proceedings on Human Factors in Computing Systems, Amsterdam 1993

[15] Brewster, S.: "Using Non-speech Sounds to Provide Navigation Cues." ACM Transactions on Computer-Human Interaction, p224ff, 1998

[16] Crease, M. & Brewster, S.: "Making Progress with Sounds. The Design Evaluation of an Audio Progress Bar", Proceedings of the International Conference on Aditory Display, 1998