The Broadway Audio System

Ray Tice, Mark Welch†

Abstract

This paper describes the X Audio System, a proposed Consortium standard for application access to network-transparent audio services. These services include the ability to play, generate, and record audio clips. The system also allows audio services to be coordinated with other services, such as video or graphics. The X Audio System owes much of its heritage to the X Window System, Digital's AF, NCD's Network Audio System, and many other prior systems.

Goals

Simply put, the X Audio System provides applications with access to audio services. These include the ability to play, generate, and record audio clips. The system also allows these services to be coordinated with other services, such as video or graphics.

The X Audio System shares many goals with the X Window System. Network transparency allows an application to use audio devices on the same machine, or on any other machine on the network. Hardware independence allows programs to be written once, but usable for a wide variety of audio hardware. Device sharing allows multiple applications to use the audio hardware simultaneously. A C compatible common application programming interface (API) allows programs to be portable across different platforms. And extensibility allows vendors to add additional capabilities.

There are other goals for audio services that are not shared with the core X protocol. For example, notions of security, compression, and cooperation with other media have been built into the core audio protocol. These allow better integration into a larger infrastructure, currently known as

---

†*Ray Tice (ray@x.org) is the architect of the X Audio System, and is a member of the technical staff at the X Consortium. Mark Welch (mw@x.org) is a member of the technical staff of the X Consortium.*

Broadway. Please see Scheifler, *Broadway: Universal Access to Interactive Applications over the Web*, also in these Proceedings, for more information on the overall Broadway infrastructure.

Finally, the X Audio System has been designed to make writing simple programs simple, with the remainder of the system learnable on an incremental basis. The programming model has been designed to fit well with toolkits, so that a single programming style can be utilized throughout an application.

Targeted Applications

In order to ship in a timely manner, version 1.0 focuses on support for the following applications:

- Basic record and playback
- Audio on the web
- Playing synchronized audio/video clips
- Teleconferencing
- Support needed for NT audio device drivers

In addition, the architecture was selected to allow future growth.

Applications Not Targeted

There are also application areas for which advanced capabilities have been omitted from the X Audio System, or have been intentionally omitted from version 1.0 of the core protocol. In most cases it is felt that such capabilities are best layered on top of the audio system, designed as an extension to the core audio system, or deferred until after the first release. Non-goals for version 1.0 of the core protocol include the following:

- A generalized digital signal processing or filtering environment.
  The system focuses on handling audio for human consumption, rather than providing signal analysis tools.
- Post-production or studio production.
  It is not the goal of the audio system to provide a full sound studio environment within the core server.
- Internal provision of sophisticated multimedia synchronization paradigms.
  The current audio system provides low level support for synchronization of other media to audio and a virtual time model. Its architecture allows clients or extensions to provide higher level synchronization, but the core protocol does not provide these directly.
- Control of generalized analog signal routing and processing.
- MIDI support in the core server.
- Full game support.

Note that since the audio system architecture is designed to be very extensible, these services can be added at a later date.

Overview

The audio system meets the needs of targeted applications with the following features:

- Record and playback of audio clips
- Temporary storage of audio clips
- Encapsulation of audio hardware services in a server.
- Rate and format conversion.
- Explicit time model for audio data streams and devices.
- An extensible programming model and interface compatible with toolkits.

The X Audio System uses a client-server architectural model, where audio hardware is abstracted into the server, and the application becomes a client of that server to obtain audio services. The application becomes a client of the server by opening a connection to the server.

The X Audio System defines three components: the API that the client uses to interact with the library, the protocol that the library uses to interact with the server, and the objects that the application manipulates via the library and protocol. Objects exist on both the client and server sides, depending on what services they abstract.

The object model uses the notion of classes. A class defines a list of values called attributes and defines the meaning of each of these attributes for that class and what happens when these attributes change. Unlike some object models, the X Audio object model defines only a few methods (or requests) on the object: create, destroy, get, set, and (for some objects) read and write.

The protocol and C API are relatively small, since they provide a generic mechanism to create, destroy, modify, and query objects. The complete client visible state of the server and library is presented as a collection of objects. The classes of these objects are defined in the protocol and library specifications. The system provides pre-created instances of some of these classes, and the application may create instances of some classes. It is not intended for applications to subclass from these classes.

Server Classes

A client uses an instance of a server side “port” object to move data into and out of the server. The application uses the port to access the buffer of an output “device” or input device in the server. A simple example may help explain this.

A simple case is where an application has samples in its memory and would like to play them. To do this, the application takes the following steps:

- Opens a connection to the server.
- Obtains a “format” object in the server that describes the sample rate and other characteristics of the samples.
- Creates a port object in the server to accept samples from the client for output to the default output device. (The port will use the format created in the previous step to decode the audio samples sent to it.)

- Sends the audio samples to the port.

Figure 1 below shows the resulting setup, with client-created objects to the left of the vertical dashed line:

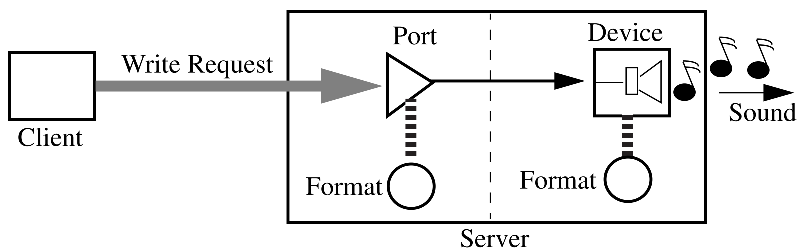

Figure 1: An application writing samples to an output device.

In the above figure, the port object receives the audio samples in the client's format and timeline. The format of the client's audio samples is defined by the format object attached to the port. The port object converts the samples to the format of the device and schedules the samples to the timeline of the output device for playback.

To record samples, the process is very similar, except that the client creates a port object that makes the audio samples from the default input device available for reading, and then the client fetches samples from the port. In Figure 2 below, client created objects are shown to the right of the vertical dashed line.

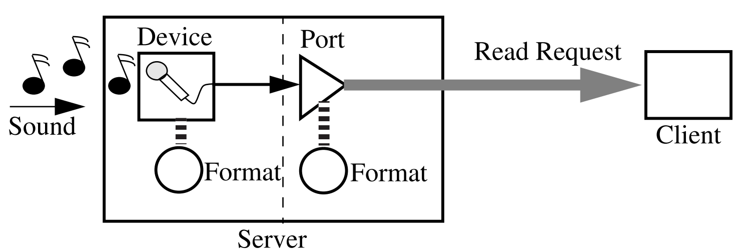

Figure 2: An application reading samples from an input device.

There are several other classes of objects in the server. For example, bucket objects temporarily store audio clips in the server, waveform objects generate synthetic audio signals, and other classes exist which are used for access control. Triggers provide notification to client applications whenever a targeted set of attributes change or an error occurs. In fact, the entire client-visible state of the server is presented as attributes on instances of the various classes.

Client Classes

There are several classes of objects which exist only in the client. File objects represent audio files on disk, and contain information parsed from the audio file header, such as the file and data formats. Reader objects provide a spooling mechanism by which a client application can read samples from an audio file and automatically send the data to an audio server. Finally, event handler objects exist within the client as a sort of “handle” for manipulating triggers within the server, while at the same time encapsulating callback information in case the trigger sends an event message back to the client.

Example Application

One of the primary design goals of the X Audio System is to enable developers to write simple, often-used applications with a minimal amount of code. Here is an example which demonstrates the relative simplicity with which X Audio applications may be written. The following code, given a buffer of µlaw-formatted data, opens a connection to an audio server, creates a port on an output device (speaker) and writes the buffer’s contents to the port.

```
XaErrorCode playUlawBuffer(void *buf, int numSamples)
{
    XaAudio      aud;
    XaTag        outputPort, fmt;
    int          numBitsToProcess = numSamples * 8;
    int          numBitsProcessed = numBitsToProcess;
    XaErrorCode  err = XaEsuccess;

    /* Open a connection to the audio server. */
    aud = XaOpenAudio();

    /* Get a ulaw format object so that we can specify what kind
       of data we wish to send */
    fmt = XaFind(XaCFormat, "ulaw");

    /* Create a port onto the default output device.
       Setting the input device to XaTclient tells the server
       that the client will be writing to the port.
       The output device will be automatically set to the
       default output. */
    output = XaCreate(aud, XaCPort,
                      XaNinputBuffer, XaTclient,
                      XaNformat, fmt);

    while((numBitsToProcess > 0) && (err == XaEsuccess))
    {
        err = XaWrite(aud, output, (XaTime) 0, XA_LATEST_TIME,
                      buf, numBitsToProcess, 0, &numBitsProcessed);
        numBitsToProcess -= numBitsProcessed;
    }

    XaDestroy(aud, outputPort);
    XaCloseAudio(aud);
    return err;
}
```

Status

The protocol and API specifications for the X Audio System are expected to go to consortium review shortly. The X Consortium implementation of the X Audio server and client library will be included as part of the Broadway release.

Further Reading

Slides of the XTECH '96 presentation accompanying this paper may be found at ftp://ftp.x.org/contrib/conferences/XTech96/audio_slides.ps.

See also Scheifler, R., *Broadway: Universal Access to Interactive Applications over the Web*, elsewhere in the XTECH '96 Proceedings (slides at ftp://ftp.x.org/contrib/conferences/XTech96/).

Acknowledgments

The X Audio System is the result of efforts by many people. The authors wish to thank David Rivas of Sun Microsystems, Peter Derr of Digital Equipment Corp., and Mike Patnode and Shawn McMurdo of SCO.

Author Information

Ray Tice and Mark Welch may be reached at the following address:

X Consortium Inc.
201 Broadway, 7th floor
Cambridge, MA 02139
USA
Phone: (617) 374-1000