Extended User Control over Multichannel Content Delivered over the Web

Nicolas Bouillot[1], Marcio Tomiyoshi[2], and Jeremy R. Cooperstock[1]

[1]*McGill University, Montreal, QC, Canada*

[2]*Universidade de São Paulo, São Paulo, SP, Brasil*

Correspondence should be addressed to Nicolas Bouillot (nicolas@cim.mcgill.ca)

ABSTRACT

We describe a server architecture that allows for inclusion of multichannel content in a website, supporting delivery to the user's web browser. With conventional content-delivery mechanisms, this is possible only by implementing browser-specific plug-ins for each platform. Our solution applies not only to any Flash-capable browser, but additionally supports per-channel volume control by the user while maintaining stream synchronization during playout. This capability is critical for end-user "mixing" of streamed audio/video content, which, while motivated by the requirements of our "Open Orchestra" music rehearsal system, is also useful for other potential applications.

1. INTRODUCTION

Our Open Orchestra project [7] aims to provide a simulated experience of ensemble rehearsal or performance, from home or music schools, with the convenience and flexibility of solo study. This builds on the theme of an immersive orchestral simulator and allows a musician to practice within the context of a group, as well as to refine his aural perception and interaction with the other instrumentalists. While traditional rehearsal restricts a musician to hear the ensemble only from his spatial position, preliminary experiments suggest that the capability of adjusting the mix of the other instruments, for example, to resemble the listening perspective of a different musician or the audience, is pedagogically valuable. To support this capability within Open Orchestra, the different instrument sections of a piece are recorded on separate channels so that they can be controlled independently by the musician during playout.

The Open Orchestra simulator is implemented as a Rich Internet Application. To achieve the flexibility of solo study, a lightweight version of the system is intended to be available to the student at home, but with reduced computational and network resource requirements relative to the school version. Open Orchestra includes an online library of several music pieces, each with audio and video recorded and pre-processed in order to simulate the perspective of different musicians within the ensemble. For any of these pieces, the student can replace one of the previously recorded musicians, and practice with the rest of the ensemble, experiencing the rehearsal sonically and visually from the same spatial position as the instrumentalist he is replacing. This is much like Music-Minus-One, but with the addition of immersive video and user-controllable multichannel audio. Other interactivity elements, not presented here, include automatic or manual page turning of the sheet music and messaging between instructor and students [7], as well as student performance feedback through visualization [5].

Audio/video content delivery in web applications, typically hosted in a browser-controlled environment, is commonplace. Both Adobe Flash streaming and HTML5 provide specifications for serving audio and video to most currently available web browsers. However, these technologies can only guarantee synchronized delivery of a single video channel with a stereo pair of audio channels. Delivery of more than two audio channels, synchronized both with each other and the accompanying video stream(s), to a web client that can control volume and spatial panning, is beyond the capability of existing tools. This paper describes our design and implementation of a server architecture overcoming these limitations, as needed for applications involving delivery of multichannel audio, possibly accompanied by one or more video streams, and in particular when requiring interactive control. In this architecture, multichannel handling is performed at the server side and mixed into

the appropriate number of channels to be served to web browsers.

The remainder of this paper is organized as follows. After describing in more detail the current status of audio/video delivery to web applications in Section 2, we propose our server architecture in Section 3. We then explain the mechanisms for control and monitoring in our custom streaming engine in Section 4 and conclude with a summary and plan for future and ongoing work in Section 5.

2. AUDIO/VIDEO DELIVERY TO WEB APPLICATIONS

In this section we review several protocols for audio/video streaming, along with their support in web applications when hosted in a browser-controlled environment.

2.1. Audio/video streaming protocols

The Internet Engineering Task Force (IETF) has provided a set of standards for data streaming over IP networks. Under these standards, various protocols have emerged, each targeted to specific tasks. Real-world applications typically require the use of several such protocols in parallel. For example, the Real-time Transmission Protocol (RTP) defines a standardized packet format for data transport of audio and video [8]. This is specified in conjunction with the RTP Control Protocol (RTCP), which provides synchronization information when multiple streams are transmitted. Another protocol, the Session Description Protocol (SDP), provides a format for describing streaming media initialization parameters [1]. SDP does not deliver media itself, but rather, is used for negotiation of media type, data format, and associated properties between end points. While SDP and RTP/RTCP support multichannel description, transmission and synchronization, the practical availability of multichannel handling is dependant on its support by receiver software.

A competing protocol, Adobe's Real Time Messaging Protocol (RTMP) [2] is used by many popular video broadcasting websites. RTMP integrates several communication features that operate independently of each other in the same TCP stream. It supports a single channel of compressed video in FLV or H.264 [4] format and a stereo pair of compressed audio channels, for example, in MP3 or AAC format [3]. In addition, RTMP enables Remote Procedure Call (RPC), allowing client application to call a specific server procedure.

2.2. Reaching the browser

Although the recent HTML5 specification includes a video element, no standard mechanism is defined for the handling of multichannel audio, while the only interactive controls provided are for audio volume (one only) and playhead position. In addition, the sets of supported video and audio formats are inconsistent across various browsers. Thus, with HTML5, interactive control over individual audio channels is not possible and synchronization of multiple channels or multiple media types is at best uncertain. In addition, RTP support is not imposed but depends on browsers and/or the specific plugin installed.

Despite the limitation in number of channels, the RTMP protocol, as part of the Adobe Flex development framework, is handled by the Adobe Flash plugin. It is thus available for almost all desktop computers, operating systems, and browsers. Importantly, its RPC feature allow the client to control specific computation done by the server, as required for server-based mixing. Moreover, it provides additional functionality that may be useful or necessary in media-based applications, such as audio recording or video display across multiple screens.

3. ARCHITECTURE

To address the previously described shortcomings, we designed an architecture and implemented its components. In this architecture, multichannel audiovisual content is handled at the server side, mixed in real time and forwarded as a live video stream with stereo audio to the clients. Accordingly, the server maintains synchronization between channels and supports content processing operations, such as volume and panning, which can be controlled by the client in real time.

3.1. Description

The server consists of two components, the custom streaming engine, which handles multichannel content, and the RTMP session manager, which maintains sessions with clients, and translates streaming protocols to the RTMP format. As described below, these communicate with each other by means of popular protocols.

The streaming engine is dedicated to content processing and stream serving. Its features include play/pause/seek

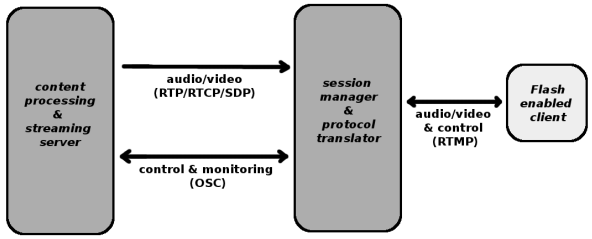

Fig. 1: Server architecture with protocols involved in end-to end delivery and control of multichannel content

controls in addition to volume and interactive panning control over individual channels of audio. Internally, it handles simultaneous reading of multiple files, mixing of the various audio channels into stereo audio and synchronization with video, AAC encoding of audio at 320 kbps, stream marshaling, and transmission.

The RTMP session manager re-packs the RTP/RTCP stream provided by the streaming engine, converting it to RTMP without the need to re-encode the content. This component also manages sessions of multiple, possibly simultaneous, clients. Accordingly, each client is given a dedicated session, allowing for content serving and control forwarding to the streaming engine without affecting other client sessions. Our Java implementation of this component extends the default functionality of the Wowza Media Server 2.[1]

A typical session starts with the client requesting a piece from the session manager, using the RPC provided by RTMP. This request launches a dedicated streaming engine process with parameters enabling communication between the two server components. At initialization, it extracts the streams' parameters and generates an SDP file, describing the specific stream configuration, and makes this available to the second component. This file is then handled by the protocol converter that converts and forwards data to the client. Obviously, these steps require non-trivial set-up, discouraging their repetition each time the user wants to navigate the stream. Accordingly, this initialization process is achieved during session creation and maintained during the entire session. The streaming engine has been implemented in C, largely using the GStreamer framework [6], allowing for the construction of graphs of media-handling components.

[1] `http://www.wowzamedia.com/`

3.2. Synchronization

Our custom streaming engine is built as a pipeline of components, building on the GStreamer open source multimedia framework. To maintain synchronization, the system clock is used to slave all elements in the pipeline, including file reading and RTP/RTCP marshaling. The primary task of this clock is to control the progress of time among the GStreamer elements, potentially running in parallel, in a synchronized manner. GStreamer derives several times from the system clock and the playback state, e.g., the running time of the pipeline and the current position in the media. This "stream time" is handled by the RTP payloader that determines header timestamps, allowing the marshalling element to specify in an RTCP packet the synchronization information related to the stereo audio stream and the video stream.

When our protocol translator receives the RTP streams described in the SDP file, it decodes synchronization information from RTCP and converts it into the RTMP format, maintaining a constant temporal relationship between audio and video at the client.

3.3. Control and feedback

Interactive control over volume, panning of individual audio channels, and playhead (play, pause, and seek) as well as feedback of streaming parameters between the client and streaming engine are coordinated through control names, associated with particular media content. In our Open Orchestra project, this information is obtained by the client during "discovery", i.e., requesting a list of available content from the database. When the user changes a parameter, the client forwards the parameter name and its new value to the session manager through the internal RTMP remote procedure call feature. The session manager then translates these calls to messages handled by the streaming engine so that they can be applied, and provides the client with confirmation of successful updates.

4. ENABLING CONTROL AND MONITORING IN OUR STREAMING ENGINE

We now provide further details regarding the control and monitoring of the GStreamer element.

Typical GStreamer-based applications are controlled and monitored using a custom interface such as a graphical user interface that operates on *properties* of the *elements* composing the pipeline. For instance, the volume element has a boolean property called "mute" that can be controlled (written) during the pipeline action. These properties may be *readable* and/or *writable* by other elements or by the application itself. Some properties are also *controllable*, which allows for automation of parameter control using interpolation between the specified values positioned in time; this is a function available in most media editing applications.

In the case of a server application, however, control and monitoring are performed remotely, requiring an entry point to control such properties. For a streaming engine, control should be independent of content, since the number of channels and the processing applied is application dependant. For instance in Open Orchestra, the number of audio channels may vary according to the instrumentation of a musical piece.

These requirements motivated us to develop a GStreamer element that provides remote control and monitoring of a pipeline, independent of its composing parts. Our element supports the following requests:

- **set** the value of a property for a specified element. Required arguments are the element instance name, property name, and the value to apply
- **get** the value of a property. Required arguments are the element instance name, property name, the IP address and the port required for the response
- **subscribe** to a property. Required arguments are the same as for the get request. The requester will be notified of any change applied to the parameter, even internally. Our implementation allows for multiple subscribers.
- **unsubscribe** from a property of a given element.
- **automation** sets automation for a property of a given element. Required arguments are the element instance name, property name, two values and two dates in units of seconds. The property will progressively ramp by linear interpolation from the initial value to the target value at the given stream times.

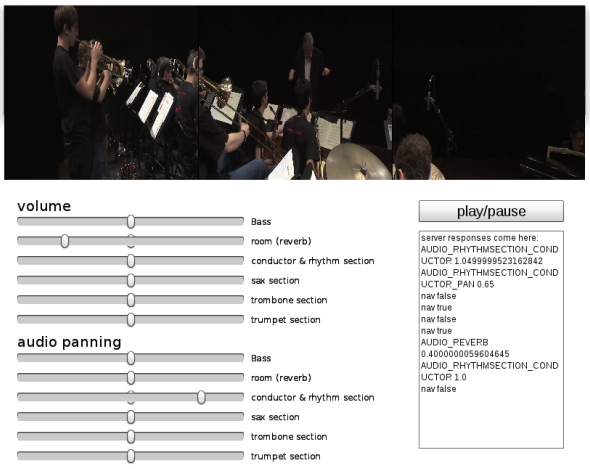

Fig. 2: Our demonstration client embedded in a web page. This provides interactive control of the playhead (play/pause) and independent volume control over six stereo audio channels. The text area is provided here to observe responses from our streaming engine.

In our server implementation, when the session manager receives a parameter request from the client, it automatically subscribes to it, if not yet subscribed, and forwards notification to the client using the RTMP messaging system. Our current system allows for notification of streaming engine internal state (play & pause), playhead position, as well as volume and panning of each channel.

Messaging with our GStreamer element uses Open Sound Control (OSC) [9] for communication. This enables external monitoring of the sessions using available command line OSC tools. Because various features of the streaming engine can be monitored, both at a low level, such as RTP timestamps, and high level, such as file locations, this significantly eases debugging tasks.

Client side

Our sample client is shown in Figure 2, illustrating capabilities required by the Open Orchestra system and supported by our server. The multichannel content used here is a Big Band Jazz piece composed of six independent audio channels and three video channels, stitched together to create a panoramic view, all rendered from the perspective of the bassist. The client implementation used Flex, allowing it to be embedded in a web page or compiled for Adobe Air runtime.

5. CONCLUSIONS

We presented an architecture allowing for the delivery of multichannel content to the web. Our server is composed of a custom streaming engine, capable of per-channel processing, stream synchronization, and remote monitoring and interactive control. This overcomes the lack of multichannel support in current web technologies for delivery of audio/video content. Stream serving to clients is performed using the Adobe RTMP protocol, which allows for delivery to a wide variety of desktop clients, regardless of the operating system and browser. Importantly, the client software, hosted in a website, is downloaded during a user's visit and then launched in a browser-controlled environment, requiring only the Flash plugin is additional software.

Current functionality, including playhead control that maintains stream synchronization, per-channel interactive volume control and panning, are being used in our Open Orchestra project. Our streaming engine allows for processing features such as resampling, time-stretching, and automation of parameter control. It also allows for live encoding of the content with dynamic bitrate control, possibly enabling adaptation of the stream bandwidth to client resources and/or end-to-end network performance. We expect the architecture to scale easily to support additional features as part of our future work.

We have not yet formally evaluated control response time, which is a critical factor in user experience with the system, especially for continuous parameters such as volume adjustment. Fortunately, the latency characteristics of each buffer in the system are generally controllable. However, one must keep in mind the tradeoff between reliability and delay in tuning buffer and audio frame sizes.

6. ACKNOWLEDGEMENTS

The authors would like to thank Teresa Liem for her valuable work on the manuscript, and Canada's Advanced Research and Innovation Network (CANARIE) for funding of this work, conducted under a Network Enabled Platforms (NEP-2) program research contract.

7. REFERENCES

[1] M. Handley, V. Jacobson, and C. Perkins. RFC 4566 - SDP: Session description protocol, 2006.

[2] Adobe Systems Incorporated. Real time messaging protocol chunk stream (version 1.0), 2009.

[3] Information technology – generic coding of moving pictures and associated audio information – part 7: Advanced audio coding (AAC). ISO/IEC 13818-7, 2006.

[4] Information technology – coding of audio-visual objects – part 10: Advanced video coding. ISO/IEC 14496-10, 2009.

[5] Trevor Knight, Nicolas Bouillot, and Jeremy R. Cooperstock. Visualization feedback for musical ensemble practice: A case study on phrase articulation and dynamics. In *Conference on Visualization and Data Analysis (VDA)*, Burlingame, CA, USA, January 2012. IS&T/SPIE.

[6] Stefan Kost. Writing audio applications using GStreamer. In *Linux Audio Conference*, Netherlands, 2010.

[7] Adriana Olmos, Nicolas Bouillot, Trevor Knight, Nordhal Mabire, Josh Redel, and Jeremy R. Cooperstock. Open orchestra: a high-fidelity orchestra simulator. *Computer Music Journal*, *pending publication*.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 3550 - RTP: A transport protocol for real-time applications, 2003.

[9] Matt Wright. Open sound control 1.0 specification. Published by the Center For New Music and Audio Technology (CNMAT), UC Berkeley, 2002.