Asynchronous Audio Conferencing on the Web

David A. Turner and Keith W. Ross
Institut Eurécom
BP 193, 06904 Sophia-Antipolis, France
{turner, ross}@eurecom.fr

Published in the proceedings of the International Symposium on Intelligent Media and Distance Education, August 1999

Abstract

In this paper we discuss the Web as a medium for hosting asynchronous conferences that allow participants to leave voice messages in addition to text. After discussing the advantages of adding audio to the asynchronous conferencing setting, we describe a prototype system that we implemented, which relies on a combination of new Web technologies, including ActiveX, Java and dynamic HTML. We made special efforts to keep our implementation free of any special software installation, and thus allow participants to begin participating in the conference immediately upon arrival to the conference site. While the current state of Web infrastructure supports the submission of text data from client to server, by the HTML FORM element or through a Java applet, there does not yet exist an analogous method for capturing audio data at the client and deliver it to the server. Our solution relies on an ActiveX control to provide this missing functionality, which limits the system's accessibility to users of Internet Explorer under Windows. We describe two proposed technologies that would provide developers with a platform independent means of capturing audio data and submitting it to a server.

Introduction

Currently, on the Internet, there are a number of different forms of asynchronous conferencing techniques. Mailing lists are one form of conferencing tool, where participants submit e-mail to a central server, which periodically compiles these comments into a single message and distributes it to the participants. News groups are another from of asynchronous conferencing, where participants place messages within a message tree. Newsgroups now take two different forms. There are the traditional newsgroups, based on the Network News Transfer Protocol (NNTP) [1], and Web-based newsgroups that rely on systems such as HyperNews [2], a CGI-based system in the public domain. The portal sites (Yahoo, Netscape, etc.) are also beginning to provide asynchronous conferencing services to their clients.

These asynchronous systems have so far been based on text messages. In particular, audio has yet to emerge as a medium for asynchronous conferencing on the Internet. However, because computer systems with audio playback and capture capabilities are now becoming common, it is now feasible to add audio to asynchronous conferencing. To better understand the technical issues involved with such development, and begin experimentation with its use, we developed a prototype audio-based asynchronous conferencing system [3].

Adding audio to asynchronous messaging adds a new dimension of possibility for participants to express themselves more completely. Although emotion can be conveyed in text, its expression may be more complete or easier to achieve when using one's voice. For example, an asynchronous classroom where students are reading and discussing poetry would benefit from the use of voice, because the reader's voice can convey emotions that might not otherwise be experienced by a reader. Voice messages also can provide a more personal feeling that is difficult to accomplish with text only. Imagine a private conference set up for the use by a family or group of friends; in such an environment, listening to the familiar voices of family or friends may help to evoke a feeling of closeness, which may be harder to achieve with text-only messages. In addition to the emotional and personal dimensions, voice messaging is highly desirable in the context of an asynchronous classroom for the instruction of a foreign language. In this context, students can replay spoken examples to study the sound and rhythm of the language, and submit their own responses for correction by the instructor.

In addition to improving the expressiveness of messages, audio messaging systems can be more efficient than text-only systems, because message creation can be done more rapidly. For example, a

speaker of English usually speaks at a rate of about 180 words per minute, whereas the majority of users type less than 60 words per minute. However, although message creation is more efficient, message consumption may be less efficient, because people can read text more quickly than the rate at which it is spoken. Also, readers can skim message text in order to omit content they feel is unimportant. But these problems are resolvable, because (1) audio messages can be played back at a faster rate than it was recorded, and (2) special audio rendering techniques, such as those used in SpeechSkimmer[4], provide users with functions that let them "skim" audio in a manner analogous to how they skim text.

Another advantage to audio messaging is that many people consider reading text on the computer screen to be a source of eyestrain, especially when reading through a long document. While the graphical user interface of the monitor may be the appropriate place for the user to process multiple choices presented in a single instant, many users tire when reading sequentially presented information, such as a document. Clicking on hyper links to get to the information one desires is probably the most efficient means of navigating through an information space, but after one has located the information desired, it may be easier for the user to listen to a document rather than read it.

Finally, we chose not to consider video at this time for three reasons. First, the bandwidth required for transmitting good quality video data is beyond what many users have available to them through a modem connection, and so the playback delay may be too great. Second, the storage requirement for video data is large, and would therefore consume excessive disk space on the server. Third, although many multimedia-equipped computers have microphones, few have video capture devices. Of course, these factors will undoubtedly disappear over time, making video messaging more attractive; but the aim of our experiment was to implement a system that would be accessible to as many users as possible and function reasonably within the present environment.

Design Objectives

The guiding principle that we followed when designing our prototype system was to make it free of any special software installation, including going to a Web site to download and install a plug-in. If the user has speakers and a microphone connected to her system, then it should be possible for the user within a Web browser to follow a link to a conference, and begin listening to messages and leaving messages of her own.

We were able to reach this goal within reasonable limits. The first time the user goes to a conference, she is presented with a pop-up window asking if she wishes to accept an ActiveX control, with a valid electronic signature. If the user clicks OK, the ActiveX control is loaded and installed. The user is never again confronted with the same question; any conference he may go to, at any Web site, will be able to interact transparently with the installed control.

As a second goal, we wanted the system to be accessible to as many people as possible, that is, to be operational in any browser and on any operating system. Since Java is platform independent, we naturally considered developing the client side of the system with a Java applet. However, Java does not yet provide developers with audio capture functionality. Such functionality should become available in the future, when implementations of the JavaSound API are provided for the various platforms. There is also no mechanism in HTML that provides for audio capture and delivery via an HTTP POST to the server, as there is for text data. There is an Internet draft [5] proposing an extension to the INPUT element of HTML that provides this functionality. Another possible solution is for browsers to support an OBJECT element that displays audio capture controls, and functions within the FORM element for submission of its data to the server. This mechanism is described in the W3C HTML 4.0 specification [7].

It seems likely that such functionality will eventually be provided across all platforms either through the HTML FORM element or through the OBJECT element.

Because a relatively platform independent solution was not available, we had to settle with a platform-specific solution at the client. We chose to use an ActiveX control to provide the audio capture and delivery, thus limiting conference access to users of Internet Explorer running on a Windows operating system.

The Client Web Page

The standard format of a conference is the message tree. Messages are either placed on the top level or under a parent message. Messages are represented by lines containing message title, author

and other information such as date. These lines are then presented as a vertical list in which child messages are indented under their parents. An icon appears to the left of the message, which when clicked, the list of the message's children either appears or disappears beneath the message. This is the standard interface for viewing the contents of a hierarchically ordered collection of objects.

We experimented with a different approach, which was to present a vertical list centered on a "selected" message. Above the selected message are all of its ancestors, and below are all of its children. We found the interface too confusing, because messages change their positions within the display window when selected. However, we feel this approach still warrants some continued evaluation as a possible alternative to the expanding branches of a tree. The reader can see an example of this interface at http:www.eurecom.fr/~ turner/interface2.html.

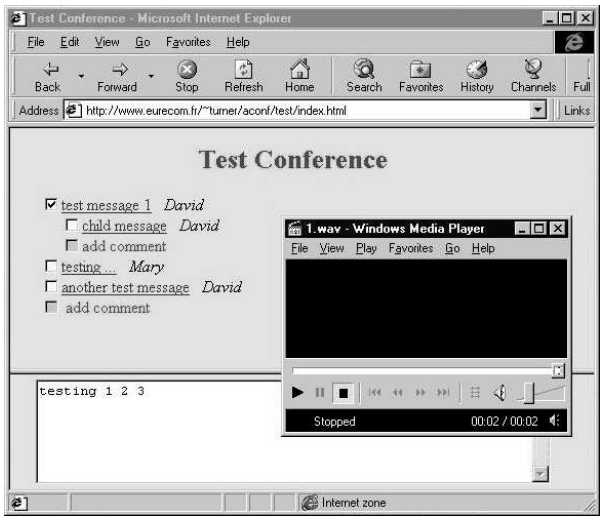

Figure 1 : Message tree interface

We thus remained with the conventional tree with expanding branches. When the user first arrives at a conference, he sees a list of top-level messages. Each message is presented as a checkbox, followed by the message title and the author's name. When the checkbox is selected, the children of the message are displayed, indented, under the message. When the title of a message is clicked, the message text appears in a textbox at the bottom of the screen and the system's media player retrieves and plays the audio component of the message (Figure 1). The user controls the playback of the audio through the playback controls of the media player.

Under the last child message in each exposed list is an "add comment" link. The user clicks on this link to insert a new message at that point in the tree. After clicking on one of these add comment links, the message tree disappears and is replaced by a message creation interface (Figure 2). This new screen includes textboxes for the user's name, message title and message text. In addition to the text input boxes are audio capture controls in the form of buttons. These buttons are labeled RECORD, STOP, PLAY and ERASE. The user clicks RECORD to start audio capture, STOP to stop capture, PLAY to playback the contents of the capture buffer, and ERASE to erase the contents of the capture buffer. He can also click STOP to stop playback and RECORD to append onto data in the capture buffer. Once the user is satisfied with his message, he clicks the SEND button to submit the message to the conference server. If he decides not to submit his message, he clicks on the CANCEL button.

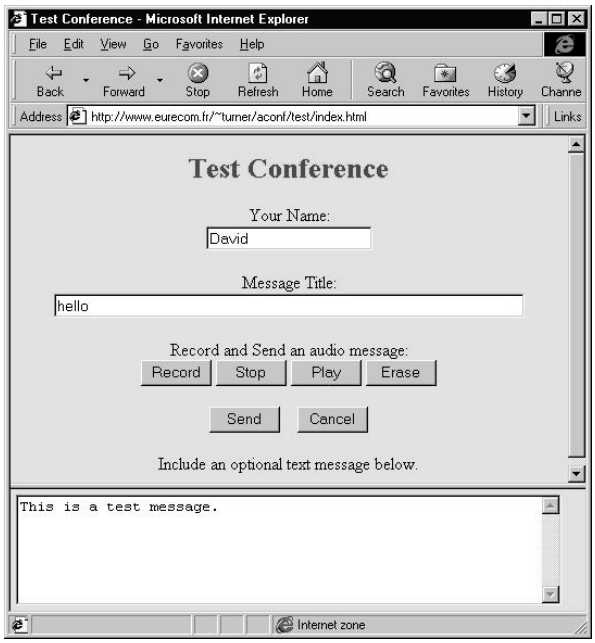

Figure 2 : Message creation interface

The transition between the message tree of Figure 1 and the message creation interface of Figure 2 is done with dynamic HTML, which means that no interaction with the server is required. Although this technique provides for an extremely fast user interface, there is a problem regarding the browser's back button, which when clicked within the message creation window will not bring the user back to the message tree, but to the page from which the conference was originally entered. This behavior may not be

understood by the user, because the two states of the unified Web page have the appearance of being two different Web pages. This problem will exist for all such Web applications that utilize the document object model to provide a highly interactive Web page. The Web page designer may need to provide better clues than we do in our prototype so that the user perceives the Web page as a single document, which will be exited when clicking the back button.

The Conference Server

System-level data is organized under a single directory: in our implementation we called the directory aconf. In this top-level directory we place all of the files which are common to all conferences, and subdirectories that contains files for specific conferences. The files in the top-level system directory include: an ActiveX control in a signed cab file (Aconf.cab), a Java applet to retrieve conference text data (AconfApplet.class), a cascading style sheet [6] used to format the appearance of all conferences (Aconf.css), a file of JavaScript functions that controls the Web document, ActiveX control and Java applet (Aconf.js), and an HTML document that defines the scrollable text area to appear in the lower frame of the We page (TextBox.html). Each conference subdirectory contains the following files: an HTML document that controls the frame layout of the Web page (index.html), an HTML document that defines the elements that appear in the upper frame of the Web page (msgTree.html), a file containing conference-specific global JavaScript variables (params.scr), a text file that contains the text data for all messages in the conference (data.txt), and the audio files named after their associated message numbers (1.wav, 2.wav, 3.wav, etc.).

The system requires a Web server, which may at any time be asked to deliver any one of the files identified in the previous paragraph. In addition to the Web server, a conference server (written in Java) listens for TCP connection requests on some predetermined port number. This port number is configured in the JavaScript file Aconf.js. The conference server's job is to accept new messages sent from the client and add them to the public message tree. It does this in two steps. First, it assigns a message number to the new message and saves the audio data as a file with name equal to the message number. Second, it appends the textual data of the new message to the conference text data file: data.txt.

Web Page Initialization

The user enters a conference by loading the index.html file in the conference directory, which results in an initialization sequence that involves the loading of various supporting files, the installation of an ActiveX control, the running of a Java applet, and the execution of JavaScript that builds the message tree.

The ActiveX control is referenced with the HTML <OBJECT> tag as follows:

```
<OBJECT id="Aconf" classid="CLSID:996F6602-
7895-11d2-8F18-006008644547"
codebase="http://www.eurecom.fr/~turner/aconf/
aconf.cab#Version=1,0,0,4"></OBJECT>
```

The classid attribute is a 128-bit globally unique identifier (GUID) that Windows uses to identify components. When the browser encounters this element, it searches the system registry to check if a current version of this object is installed. If it does not find that a recent version is present, it will retrieve the cab file and install the control. Then the browser creates an instance of the component, which it allows JavaScript to access through the value of the id attribute.

The JavaScript initialization code is started by setting the onload attribute of the <BODY> tag, as in <BODY onload="buildMessageTree()" ...>. This function adds the HTML elements that comprise the message tree. Before it starts adding elements, however, it first calls the loadData() method of the Java applet to retrieve the text data for the conference, which it does by issuing a GET request to the Web server. When this call returns, the script enters a loop in which it adds two <DIV> sections for each message in the message tree. The first pair of <DIV> sections for the first message in the conference will look as follows:

```
<DIV id="1" class="0"
     style="padding-left: 20; display:none">
<INPUT type="checkbox" onclick="expand('1')">
<A href="1.wav" onclick="play('1')">
   test message 1
</A>
<SPAN style='font-style:italic'>
   David
</SPAN>
</DIV>

<DIV id="r1" class="1"
     style="padding-left: 40; display: none">
```

```
<INPUT type="checkbox" disabled>
<SPAN onclick="addComment('1')"
      style="cursor:hand; color:red">
   add comment
</SPAN>
</DIV>
```

The second <DIV> section is used to provide a link that allows the user to add a comment at the bottom of the message's list of children. Each time a message needs to be added to the tree, two new <DIV> sections for the message are inserted just above the parent's ending <DIV> section. For more details, see the source code available at [3].

User Interface

When the initialization script completes in the client browser, the client will have all of the text data, including message titles, author names and message text. However, the audio data will remain at the server. When the user wishes to listen to a message (and view its text content), he clicks the message title, which appears as a link. The message title link for message 1 of the test conference has the following form:

```
<A href="1.wav" onclick="showtext('1')">
   test message 1
</A>
```

Clicking this link causes the browser to do two things. First, it executes the JavaScript function showtext(), which places the message text in the textbox in the lower frame. Second, it tells the system's media player to retrieve and render the wave file referenced in the href attribute. Figure 1 shows one possible result of clicking on this <A> element.

At present, when retrieving audio files from a Web server, RealNetwork's G2 player will not render the audio file in streaming mode, that is, it will wait until it has retrieved the entire file before beginning playback. The Microsoft Media Player introduces a fixed delay (settable by the user) before beginning playback of the audio file. If the audio playback rate is greater than the average bandwidth available over the TCP connection, and the length of the audio message is long enough, then the player will halt before the message has finished, while it rebuilds the playback buffer.

It is possible for these players to calculate an appropriate playback buffer size based on the average available bandwidth and file size. The file size is available from Web servers that include the Content-Length header in their response messages, which is the recommended behavior for HTTP 1.1 conformant Web servers (see §14.14 of [6]). If the encoding rate is greater than the available bandwidth , then a relatively large amount of data needs to be buffered prior to commencing playback in order to avoid buffer starvation. On the other hand, if the playback rate is less than the available bandwidth, then the initial size of the playback buffer only needs to be large enough to overcome the delay that results from the TCP slow-start mechanism.

When the user wishes to expand a branch of the message tree (that is, look at the child messages of a particular message), she clicks the checkbox next to the message. Consider the first message of the test conference example of Figure 1; the checkbox comes from the following HTML tag:

```
<INPUT type="checkbox" onclick="expand('1')>
```

When selected, the browser adds a check mark to the interior of the checkbox, and calls the JavaScript function expand() with the message number as its argument. The expand() function checks to see if the children of the message currently have the display attribute of their enclosing <DIV> tags set to “none,” which makes them invisible, or “block,” which makes them visible. If it finds that the children are invisible, it changes their display attributes to “block.” If it finds that the children are visible, it changes their display attributes to “none” and also does the same for all descendents of the children. After performing this replacement operation, the browser re-renders the HTML document to reflect the modifications the script has made to its contents.

If the user wishes to add a message, she clicks on the "add comment" phrase that occupies the position in the tree she wishes to insert a new message. The add comment phrase for adding child messages to message number 1 is rendered from the following HTML:

```
<SPAN onclick="addComment('1')"
      style="cursor:hand; color:red">
   add comment
</SPAN>
```

When the user clicks on this text, the addComment() function is called, which is another one of the JavaScript functions contained in the script file msgtree.scr. This function sets the display attribute of the <DIV> containing the message tree to “none” and the display attribute of the <DIV> containing the message creation controls to “block.” This transition from message tree display to message creation display (Figure 1 to Figure 2) is practically instantaneous,

because the browser doesn't need to retrieve any data over the network.

At this point, the user sees text boxes for a name, message title and message text. Also, there are buttons for controlling the audio capture and message submission. Clicking any of these buttons results in the execution of a JavaScript function to perform the task. After the user fills in the name and message title fields, and optionally the text field, she records a message. When she is satisfied with the recording, she clicks on send, which passes execution to the JavaScript function send(). This function transfers the text data from the HTML text boxes to the ActiveX control, and then calls the send() method of the control. The control establishes a TCP connection with the conference server and submits the new message, which is sent as text data followed by the audio data.

Conclusion

We have shown in our prototype that it is possible to construct a Web-based audio conferencing system that allows immediate user interaction without installation complexities. The user is presented only once with a window asking for permission to install an ActiveX control. Our approach, however, limits use to clients with Internet Explorer running under a Windows operating system.

Presently, it is possible to achieve platform and browser independence only by providing separate programming solutions for each targeted environment. However, when JavaSound becomes available, it will be possible to redesign the system so that it runs in all supporting browsers.

Another alternative is to extend the HTML <INPUT> element to include an audio value for its type attribute. When confronted with such an element, the browser would render an audio capture control to provide the user with a means of inputting audio. This is analogous to the way the browser renders a textbox control to allow the user to input text. A proposal for such an extension is currently in the form of an Internet draft [Salsman], and thus may eventually be adopted as a standard and implemented. Such an extension would eliminate the need for the use of mobile code such as the ActiveX control used in our prototype, and would thus increase the security of user systems.

We have also demonstrated that dynamic HTML can be used to provide a complex Web page interface, which eliminates the delays associated with a CGI-style system that requires frequent retrieval of new HTML documents from the Web server.

Our prototype system relies on a standard Web server to deliver all of the objects to the client, and only implements a server component that performs a single function: to add new messages to a conference. This conference server component is platform independent, because it is written in Java.

References

[1] B. Kantor, P. Lapsley, "Network News Transfer Protocol," Network Working Group, RFC 977, February 1986; available at http://info.internet.isi.edu/in-notes/rfc/files/rfc977.txt.

[2] D. LaLiberte, "HyperNews," freeware software, available at http://www.hypernews.org/HyperNews/get/hypernews.html.

[3] D. Turner, W. Ross, Asynchronous Audio Conferencing, prototype Web application, Oct. 1998, available at http://www.eurecom.fr/~turner/aconf/.

[4] B. Arons., "SpeechSkimmer: A system for interactively skimming recorded speech," ACM Transactions on Computer-Human Interaction, March 1997.

[5] J. Salsman, "Form-based Device Input and Upload in HTML," Internet-draft submitted to the W3C HTML activity for forms, 20 March 1999 (work in progress); available at http://www.ietf.org/internet-drafts/draft-salsman-www-device-upload-05.txt.

[6] R. Fielding, et al., "Hypertext Transfer Protocol -- HTTP/1.1," Network Working Group, RFC 2068, January 1997; available at http:// info.internet.isi.edu/in-notes/rfc/files/rfc2068.txt.

[7] D. Raggett,, A. Le Hors, I. Jacobs, eds., "HTML 4.0 Specification," W3C Recommendation, 24 April 1998; available at http://www.w3.org/ TR/REC-html40/.

[8] L. Wood, et al., "Document Object Model (DOM) Level 1 Specification," DOM Working Group, W3C Recommendation, 1 October 1998; available at http://www.w3.org/TR/REC-DOM-Level-1/.