Making Multimedia Internet Content Accessible and Usable

Hisashi Miyashita, Hironobu Takagi, Daisuke Sato, and Chieko Asakawa

IBM Research, Tokyo Research Laboratory, Japan
{himi,takagih,daisuke,chie}@jp.ibm.com

Abstract. Although multimedia content containing streaming media is now widely used on the World Wide Web, there exist considerable difficulties for blind users to access such content, due to its dynamic changes, keyboard inoperability, and audio interference with the speech from assistive software. In particular, the third problem of audio interference is serious for blind users, since multimedia content often contains streaming media such as video and music which continuously play sounds, and thus they cannot hear the speech, which is masked by the loud media.

In this paper, we propose a new accessible browser that can directly manipulate such multimedia content. In order to control Flash contents, our browser relies on a transcoding HTTP proxy to inject special scripts into the Flash content and then manipulates the embedded streaming media and sound objects via the injected scripts. By using our browser, users can easily turn the volume up or down, play, stop, or pause the streaming media with shortcut keys. Since the users do not need to focus on buttons or sliders for these operations, they can immediately stop or fade out the intrusive media when listening to speech from assistive software.

1 Introduction

Since the Web is extending to a wider range of applications, in new Web sites, multimedia content is becoming increasingly popular for richer user experiences. Many prominent Web sites such as YouTube[1], Yahoo! Video[2], and MTV[3] distribute multimedia content and are accessed by enormous number of users. The multimedia content we discuss here has the following characteristics.

Dynamic changes. Most of the traditional HTML webpages are static. That is, the rendered information is not subject to change once the page is loaded. Unlike static contents, owing to the programmability of JavaScript or Flash with scripting, multimedia contents can change dynamically in response to various inputs.

Interactive user interface. Users want to interact with multimedia content for a variety of reasons, such as selecting movies, playing or stopping the media, and controlling the volume. Therefore, such multimedia content often has its own interactive user interfaces using a mouse or keyboard.

[1] http://www.youtube.com
[2] http://movies.yahoo.com
[3] http://www.mtv.com

C. Stephanidis (Ed.): Universal Access in HCI, Part III, HCII 2007, LNCS 4556, pp. 98–107, 2007.

Streaming media. Much multimedia content provides streaming media such as a movie or music in addition to text, GUI, or document information. Such streaming media is unique among media types in the sense that it delivers transient sounds and images in response to user's operations.

However, with typical accessible user agents such as screen readers and self-talking browsers, this multimedia content is hard for visually impaired people to access due to the following issues.

Issue 1: User agents miss dynamic changes. Most of the dynamic changes in multimedia content are simply ignored by many accessible user agents. For example, many media players show the current time of the movie or music but many accessible user agents cannot handle such dynamically changing information. This is partly because Web browsers or Flash players do not notify the assistive software components of the changes.

Issue 2: Content cannot be controlled using the keyboard. Much multimedia content is heavily dependent on mouse operations rather than keyboard operations. For example, volume sliders in a media player may not be controlled with the keyboard. Whether or not the content accepts keyboard operations often depends on scripts in the content and much multimedia content does not support keyboard operations.

Issue 3: Streaming media interferes with the synthesized speech Since streaming media emits sounds, it interferes with the voices synthesized by accessible user agents. Users cannot easily listen to mixed synthesized voices and media sounds. By controlling the volume of the streaming media, we can considerably reduce this problem. However, none of the user agents can directly control streaming media without using the appropriate button or slider.

These issues are critical for blind people accessing multimedia content on the Internet. According to the User Agent Accessibility Guidelines 1.0 [9], multimedia players shall allow users to start, stop, pause, and navigate multimedia and support full keyboard access. Thus, solving these problems is quite important in order to make multimedia content accessible. To the best of our knowledge, in the existing studies on the Web accessibility, although multimedia content has been discussed, this paper describes the first attempt to make the existing Internet multimedia content accessible.

In order to address these issues, we are now trying to make a multimedia accessible browser. In this paper, we mainly focus on Issues 2 and 3. For these issues, our browser can directly manipulate the media player for Flash content by using the technology discussed in [13]. Since our browser directly accesses the internal object model of Flash, we can control media players without any mouse operations. In addition, with our browser, users can immediately change the volume of streaming media at any time without focusing on a volume slider or a mute button. This does not affect the volume of the speech synthesized by the browser. Therefore, users can reduce the volume of a movie or music when they have problems hearing the synthesized speech. For Issue 1, although some activities are now being reported [14,1], since this issue involves many difficult problems involving detecting changes and properly notifying users, we are not addressing it in this paper.

The rest of this paper is organized as follows. In Section 2, we examine the problems that browsing typical multimedia content creates for a blind user by describing a

motivating example. Our solutions to these problems appear in Section 3, where we describe our multimedia accessible browser, its architecture, and the current implementation. In Sections 4 and 5, we discuss some related work and our conclusions.

2 Accessibility Issues of Internet Multimedia Contents

In this section, we consider a motivating example of actual multimedia content in order to clarify the problems.

We are considering a movie distribution site, which is one of the most popular web sites, delivering over 100 million video clips per day to users. In Figure 1, we show an example of the multimedia content this site delivers. In such content, a movie player is embedded in the upper-left corner, and the rest is used for showing DHTML content related to the selected movie. In order to control the movie, these contents provide some Flash widgets at the bottom of the media player. At the left side, we see a play/pause toggle button and skip buttons and at the right side, we see a volume slider and a mute button. In the center, we see the current time and the total time of the movie.

Fig. 1. An example of multimedia contents

For bind people using assistive talking systems such as screen readers and self-talking browsers, such content is difficult to access for the following four reasons.

- The movie starts immediately after the page is loaded (Issue 3). Thus, many blind users using assistive talking systems will get confused since the synthesized speech for guiding the users is suddenly interfered with by the audio of the movie.
- None of the widgets in this media player have any text information (Issue 2). Therefore, typical screen readers only say “Button one, Button two, ...” so that blind users cannot understand what button is for playing or stopping the movie.

– The volume control slider at the right side cannot be controlled with the keyboard (Issue 2) even if the user knew the meaning of the widget, because this player only accepts mouse operations for this slider. Therefore, blind users cannot change the volume of the movie.
– It needs considerable effort for a blind user to control the movie (Issue 3). In order to listen to synthesized speech to navigate in the DHTML part, users need to stop or mute the movie. However, to press the stop or mute button, they have to suspend the navigation in the DHTML part, select the appropriate button, and then push the enter key. These operations are quite irritating for the users.

Based on our investigation of the several sites hosting multimedia content, we found this site is not unusually inaccessible. Almost all of the sites have the similar problems. We believe this situation is partly because making Flash content accessible is not yet a priority [4] and sites that heavily rely on Flash tend to exploit the fancier GUI features, such as the streaming feature with FLV (FLash Video), and the eye-catching vector graphic animation features.

3 Multimedia Accessible Browser

As already shown, multimedia content on the Internet is highly unfriendly for blind users. Therefore, we are now making an accessible multimedia browser, which we call "aDesigner Runtime". Our browser can analyze multimedia content in Flash and directly control the streaming and sound objects. By taking this approach, users are not forced to manipulate the buttons and sliders offered by the multimedia content. Instead they can use predefined shortcut keys to control the media. In this section, we describe the detailed architecture of our browser.

3.1 Overall Architecture

The aDesigner Runtime is designed as a self-talking browser like Home Page Reader [3]. In Figure 2, we show the schematic diagram of our browser.

Our browser first injects special scripts into the incoming multimedia Flash content in a special transcoding HTTP proxy, using the same technology discussed in [13]. Then, the base browser runtime such as Microsoft Internet Explorer with the Adobe Flash Player receives the transcoded content and renders it. Our browser internally has a unified DOM[4]-based interface, which comes from the HTML DOM provided by Internet Explorer and the Flash objects provided by the injected scripts in the Flash contents. Since these DOM objects are integrated into one tree, any program using this interface can access the documents in a uniform way.

By using this DOM-based interface, the user interface part (UI part) provides a speech interface for users. From the UI part, users can navigate in the content and control the media player with the keyboard. Fixed keyboard shortcuts are specially assigned as media player controls, such as play, stop, pause, volume up, volume down, and mute the media. Thus, users can immediately manipulate the actively playing media.

[4] Document Object Model. See http://www.w3.org/DOM/ for details

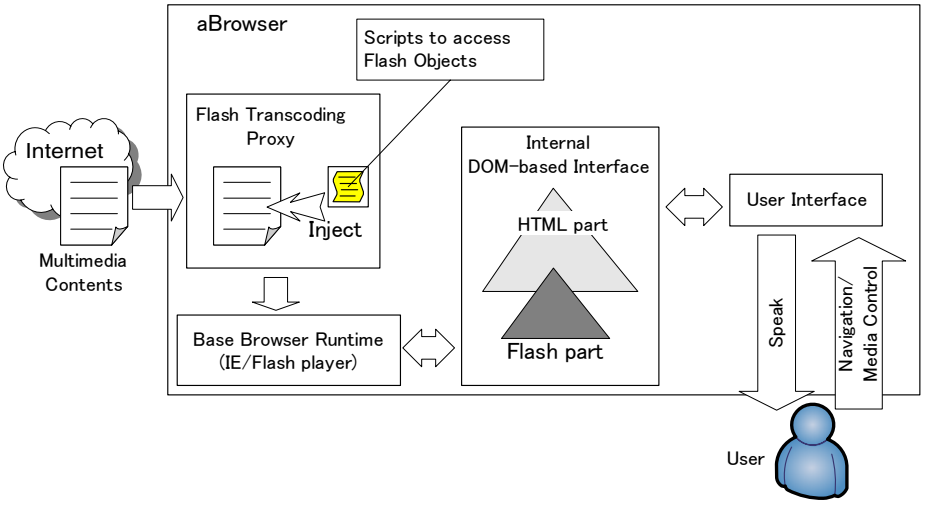

Fig. 2. Overall architecture

3.2 Flash Transcoding Proxy

Our browser uses a special transcoding HTTP Proxy to inject some ActionScript (AS) into the incoming Flash content. The injected AS has two roles: 1) it transforms the Flash objects into a DOM (e.g. a treelike object model), 2) it searches for streaming media objects (e.g. in the `NetStream` class) and sound objects (e.g. in the `Sound` class) and invokes some methods in these objects.

Since Role 1) is discussed in detail in [13], here we only add a few comments about it. The AS embedded in Flash contents traverses the structure of Flash objects from the root stage such as `_level0`. By limiting this traversal to visible objects, we can form a tree in which each node represents a visible Flash object. By using this tree, we can provide a DOM-base interface for Flash content.

For Role 2), we try to traverse all of the objects in the Flash content in order to find as many as possible of the streaming media objects and sound objects. Therefore, we traverse the `_global` property as well as the root stage. We also look into the nonvisual objects. Since the structure built of entire objects in Flash content may contain cyclic references, we track whether or not the next object has already been visited. This may still fail since objects may be stored in local variables where we cannot access from the external AS. However, in our experiments, we were able to all of the critical objects using these steps.

Once we find out the streaming media objects, we can invoke the `pause` method to play or pause the streaming media, and use the `seek` method to set the current position. By accessing the sound objects, we can use the `setVolume` method to set the volume of the media. Sometimes, Flash content contains more than one sound objects, not all of which are linked to streaming media objects. Even in such cases, we can adjust the volume of all of the sound objects with keeping the ratios among their volumes. This treatment is comparable to controlling a master volume control of all of the sound

sources in the Flash content and helps blind users listen to the synthesized speech from the assistive software, including aDesigner Runtime itself and other screen readers.

3.3 DOM-Based Interface

The aDesigner Runtime internally provides a DOM-based interface by accessing the HTML DOM supported by Internet Explorer and the DOM-based interface by the injected AS explained in Section 3.2. The reasons why we choose DOM as the common interface for the accessible browser are as follows.

- DOM is a well accepted standard for Web developers, since Web browsers and Web application frameworks support DOM for document manipulation.
- By leveraging the tree structure of a DOM, we can seamlessly integrate totally different content such as HTML and Flash, in a single DOM tree. Through this integration, the component using this interface (the UI part) does not need special considerations of their differences when we navigate in the DOM tree.
- By using the DOM, we can leave room for other Web-based technologies such as XPath and XSLT. For example, XPath can easily be introduced by using the DOM interface, which is greatly helpful for queries and for evaluating multimedia content.

This DOM-based interface provides some extensions as well as the standard DOM functions. These extensions can be classified into two groups: (a) navigation functions and (b) multimedia-content control functions. By using (a), we can click a node, obtain a heading node, and do some other tasks to provide an accessible UI. By using (b), we can search for streaming media objects and sound objects, and control the media by accessing them.

3.4 User Interface

The UI part of our browser is designed as a distinct component. This approach has the clear merit that we can present another UI by just replacing this component. For example, we could use a Braille display for blind users to navigate and control multimedia content or use tactile devices to quickly present lots of information. Otherwise, we would have to provide formatted text to other assistive programs such as screen readers by showing it in an appropriate window. This type of flexible user interface will be welcomed by the users who are familiar with their own favorite screen readers. The DOM-based interface is flexible enough for various UI components to provide good user experiences.

3.5 Implementation

We implemented aDesigner Runtime on top of the Eclipse Rich Client Platform (RCP)[5]. Most of the components in our browser are written in Java, though we use C++ to access the HTML DOM and the Adobe Flash Player via the COM (Component Object Model)

[5] http://wiki.eclipse.org/index.php/Rich_Client_Platform

and to access the Microsoft SAPI (Speech API)[6] for TTS (Text-to-Speech) feature. We also use ActionScript to write the scripts injected into the Flash content by the transcoding HTTP Proxy.

In Figure 3, we show a screen shot of our browser. In the current implementation, the user interface is based on tree navigation. That is, we can move up and down and go to the next and previous siblings in the tree made from the content. In the left panel, our browser shows the tree structure and the tree item currently selected. Every time the user moves in the tree, our browser announces the currently selected item. In addition, our browser has useful navigation commands such as "skip to heading" or "skip to next form input item," similar to typical accessible browsers such as Home Page Reader. Note that all of these features can be implemented through the DOM-base interface.

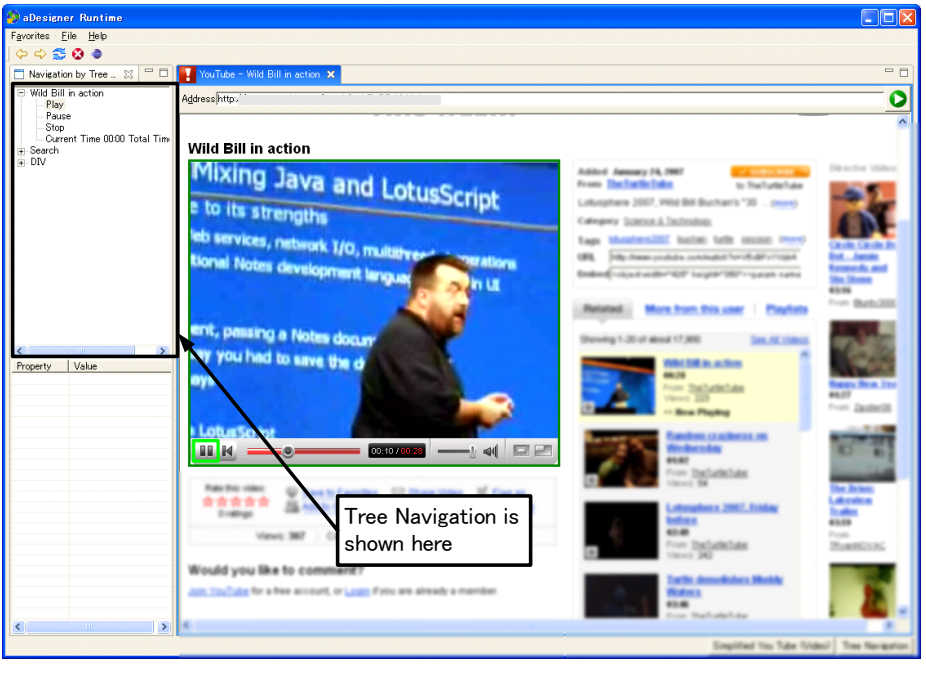

Fig. 3. A screenshot of aDesigner Runtime

Our browser is different from the others in that users can directly control embedded media by simply pressing shortcut keys. For example, by pressing `Ctrl+J`, `Ctrl+K`, or `Ctrl+M`, users can increase, decrease, or mute the volumes of the playing media, respectively. Likewise, by pressing `Ctrl+P`, `Ctrl+S`, or `Pause`, they can play, stop, or pause the media, respectively. Currently, our browser supports this direct media control for Flash and FLV contents, but we will also be able to support Windows Media Player by using the standard API, which will be easier than supporting Flash content.

[6] http://www.microsoft.com/speech/default.mspx

4 Related Work

4.1 Screen Readers and Self-talking Browsers

There exist a number of studies to access Web contents with special browsers for blind people [3,17,10]. However, there is relatively little research on making multimedia contents accessible. Recently, some accessible browsers and screen readers have begun to support multimedia content. For example, Home Page Reader [3] by IBM now supports multimedia applications such as Flash and Windows Media Player. Some screen readers such as JAWS[7] and Window-Eyes[8] also support such multimedia contents. However, all the existing softwares uses MSAA (Microsoft Active Accessibility)[9] to access Flash contents. Therefore, such software can only notify users using functions that are supported for multimedia content in MSAA. Thus, they do not have special functions to control multimedia contents as our browser does. It means that with these programs, we have to focus on the buttons or sliders that are shown by the multimedia content and use them to control the playing media.

Other interesting research coping with simultaneously presenting multimedia content and TTS is Streaming Speech[3] [7]. They proposed an extension to SMIL [2], a standardized multimedia integration language, in order to support 3D spatialization of multiple audio sources. Their approach makes use of the human ability to select multiple audio sources by 3D placement, in contrast to our approach. However, it has not yet been proved whether or not 3D spatialization of multimedia content is accessible for blind users.

4.2 Web Content Adaptation

Adaptation of Web content by transforming or analyzing it is another approach to provide more comprehensible information to a wide range of users [12,15,11,8,6,16]. Also using this approach, there are a few research projects on multimedia content. For example, Rousseau *et al.* proposed an adaptable multimedia presentation framework [12] by extending HTML. Thus, multimedia content has already been described using an extended format. Although HearSay [11] automatically analyzes Web pages and enables audio navigation, it does not address multimedia content. DeMeglio *et al.* showed a customizable multimedia player, called *AMIS*, for DAISY content [6], which has to be natively accessible. Weimann *et al.* proposed a time-dependent presentation system, called *MultiReader*, for multimedia content. MultiReader uses XSL stylesheets to reorganize content into accessible forms and also offers multimedia synchronization methods, which can be specified in JavaScript. However, this system also requires a special transcoding rule for each piece of content.

5 Conclusions and Future Work

We described a new accessible browser that tackles some of the issues when browsing multimedia content. Although lots of multimedia content poses problems for blind users

[7] http://www.freedomscientific.com/fs_products/software_jaws.asp
[8] http://www.gwmicro.com/Window-Eyes/
[9] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msaa/msaastart_9w2t.asp

using assistive speech software, our browser is an important step toward a multimedia-accessible browser. Our browser can directly control multimedia content without considering any special context such as finding special buttons and pressing them. In particular, controlling the volume of multimedia content is indispensable for blind users who need to hear the speech from their assistive software. Our browser can directly control multimedia Flash content by injecting AS code into that Flash content with a transcoding HTTP proxy. With our browser, users can easily do basic operations on the media such as changing the volume with unchanging shortcut keys.

As we have discussed so far, there is a long way to go to make multimedia content accessible. One of the important issues is about more flexible media control of multimedia content. Currently, our browser meets a minimum requirement of multimedia control such as play, stop, pause and volume controls. Since many blind users have prominent ability to recognize high-speed audio [5], in order to leverage their ability for more efficient browsing, we believe that more flexible user interface of multimedia content such as variable speed control and audio skimming control which extracts important sections is valuable especially for blind users to effectively access such content.

The other significant issue we should cope with is dynamic changes of multimedia content. Lots of multimedia content heavily uses dynamic visual changes. Since most of those changes are only eye candy, notifying visually impaired people of all of the changes is simply annoying and unhelpful. Some activities such as WAI-ARIA [14] are now tackling this problem by defining metadata for DHTML (Dynamic HTML). With WAI-ARIA, Web developers can add some metadata to DHTML content to articulate changes. However, since WAI-ARIA is defined as a set of internal metadata specific to DHTML, it has two limitations. First, we cannot apply it to other media types than DHTML, such as Flash. Second, we have to change the content itself to provide the metadata.

Therefore, we are now considering external metadata which can be applied to various types of media including Flash and DHTML. We do not have to change the multimedia content itself to attach such external metadata. With such metadata, accessible browsers can appropriately notify the users about dynamic changes of the contents.

References

1. Accessibility/IAccessible2 http://www.linux-foundation.org/en/Accessibility/IAccessible2
2. Synchronized multimedia integration language (SMIL 2.0) - [2nd edn]. W3C Recommendation (January 07, 2005) http://www.w3.org/TR/2005/REC-SMIL2-20050107/
3. Asakawa, C., Itoh, T.: User interface of a home page reader. In: Assets '98: Proceedings of the third international ACM conference on Assistive technologies, pp. 149–156. ACM Press, New York (1998)
4. Asakawa, C., Itoh, T., Takagi, H., Miyashita, H. Accessibility evaluation for multimedia content. In: UAHCI (2007)
5. Asakawa, C., Takagi, H., Ino, S., Ifukube, T.: Maximum listening speeds for the blind. In: International Community for Auditory Display 2003, pp. 276–279 (2003)
6. DeMeglio, M., Häkkinen, M. T., Kawamura, H.: Accessible interface design: Adaptive multimedia information system (amis). In: ICCHP 2002, pp. 406–412 (2002)

7. Goose, S., Kodlahalli, S., Pechter, W., Hjelsvold, R.: Streaming speech[3]: a framework for generating and streaming 3D text-to-speech and audio presentations to wireless PDAs as specified using extensions to SMIL. In: WWW pp. 37–44 (2002)
8. Huang, A.W., Sundaresan, N.: Aurora: a conceptual model for web-content adaptation to support the universal usability of web-based services. In: CUU '00: Proceedings on the 2000 conference on Universal Usability, pp. 124–131. ACM Press, New York, USA (2000)
9. Jacobs, I., Gunderson, J., Hansen, E.: User agent accessibility guidelines 1.0. W3C Recommendation (December 17, 2002) http://www.w3.org/TR/UAAG10/
10. King, A., Evans, G., Webbie B.P: a web browser for visually impaired people. In: Proceedings of the 2nd Cambridge Workshop on Universal Access and Assistive Technology, pp. 35–44 (2004)
11. Ramakrishnan, I.V., Stent, A., Yang, G.: HearSay: enabling audio browsing on hypertext content. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, pp. 80–89. ACM Press, New York, USA (2004)
12. Rousseau, F., García-Macías, J.A., de Lima, J.V., Duda, A.: User adaptable multimedia presentations for the world wide web. Computer Networks 31, 11–16, 1273–1290 (1999)
13. Saito, S., Takagi, H., Asakawa, C.: Transforming flash to xml for accessibility evaluations. In: ASSETS, pp. 157–164 (2006)
14. Schwerdtfeger, R.: Roadmap for accessible rich internet applications (WAI-ARIA roadmap). W3C Working Draft (December 20, 2006) http://www.w3.org/TR/aria-roadmap/
15. Takagi, H., Asakawa, C., Fukuda, K., Maeda, J.: Site-wide annotation: reconstructing existing pages to be accessible. In: ASSETS, pp. 81–88 (2002)
16. Weimann, K., Langer, I., Weber, G.: Adaptation of multimedia browsing techniques. In: ICCHP, pp. 135–142 (2004)
17. Zajicek, M., Powell, C., Reeves, C.: A web navigation tool for the blind. In: Assets '98: Proceedings of the third international ACM conference on Assistive technologies, pp. 204–206. ACM Press, New York, USA (1998)