August 8, 2007

RT0739
Human-Computer Interaction 8 pages

Research Report

Automatic Accessibility Transcoding for Flash Content

Daisuke Sato, Hisashi Miyashita, Hironobu Takagi, Chieko Asakawa

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Automatic Accessibility Transcoding for Flash Content

Daisuke Sato Hisashi Miyashita Hironobu Takagi Chieko Asakawa

Tokyo Research Laboratory, IBM Research.
1623-14 Shimo-tsuruma
Yamato, Kanagawa, 242-8502, Japan
+81-46-215-4793
{dsato, himi, takagih, chie}@jp.ibm.com

ABSTRACT

It is not surprising that rich Internet content, such as Flash and DHTML, is some of the most pervasive content because of its visual attractiveness to the sighted majority. Such visually rich content has been causing severe accessibility problems, especially for people with visual disabilities. For Flash content, the kinds of accessibility information necessary for screen readers is not usually provided in the existing content. A typical example of such missing data is alternative text for buttons, hypertext links, widget roles, and so on. One of the major reasons is that the current accessibility framework of Flash content imposes a burden on content authors to make their content accessible. As a result, adding support for accessibility tends to be neglected, and screen reader users would be left out of the richer Internet experiences.

Therefore, we decided to develop an automatic accessibility transcoding system for Flash content to allow users to access a wider range of existing content, and to reduce the workload for content authors by using an automatic repair algorithm. It works as a client-side transcoding system based on the internal object model inside the Flash content. It adds and repairs accessibility information for existing Flash content, so screen readers can present more accessible information to users. Our experiment using the pilot system showed that 54% of the missing alternative texts for buttons in the tested websites could be added automatically.

Categories and Subject Descriptors

K.4.2 [**Social Issues**]: Assistive technologies for persons with disabilities; H.3.4 [**Online Information Services**]: Web-based services; H5.2 [**User Interfaces**]: Graphical user interfaces (GUI), Style guides.

General Terms

Algorithms, Design, Human Factors, Standardization

Keywords

Accessibility, Flash content, automatic repair, transcoding, visually impaired

1. INTRODUCTION

These days, Internet content is dramatically changing from static to dynamic, supported by improvements in computer performance and network bandwidth. Flash and Dynamic HTML (DHTML) are representatives of these rich content types, and they appeal to sighted people with their fascinating audio and visual presentations. Although Flash was only a format for simple vector-based animations in 1996, over the following decade, it was enhanced to support high quality audio, scripts with user interactions, built-in GUI components, powerful image processing, and now streaming video support is the latest enhancement. Flash is no longer an animation format, but rather a platform for providing interactive content, and the Flash player is installed in over 98% of the computers in the world as of March 2007 (according to Adobe's survey[1]).

Meanwhile, the content owners using Flash are under pressure to support accessibility. Screen reader users frequently ask for Flash content to be made accessible. In addition, Section 508 of the US Rehabilitation Act [1] came into effect in 2001, calling for improved accessibility for government websites. In response to these requirements, an accessibility framework was designed and built into Flash authoring tools and the Flash player. The architecture is based on the Windows accessibility API, MSAA (Microsoft Active Accessibility). This API is used by a wide range of assistive technologies with screen reading capabilities. If an author of Flash content embeds the appropriate accessibility information into the content by using the Flash authoring tools, then the Flash player can expose that information for screen readers through MSAA. This means that this architecture requires content creators to study methods to make content accessible and to apply them in their daily authoring tasks.

However, it has been very difficult to make the Flash content accessible [4] compared to other formats such as HTML and PDF. The causes can be classified into three kinds of problems. First, the Flash content itself is used to create visually intuitive content using animations and movies. It is not static and the content is dynamically created and frequently changed in response to the

[1] http://www.adobe.com/products/player_census/flashplayer/

users' operations and the content's timeline. Accessibility-enabling methodologies for such rich Internet applications are not mature. Second, there is no tool to evaluate accessibility for existing Flash content [5]. Currently the only method is to use screen readers, which imposes a heavy burden on developers. Third, Flash content developers are unfamiliar with accessibility problems. Based on our preliminary investigation [4], most of them do not even know that Flash has accessibility features.

Due to these problems, not even the basic accessibility enablement requirements have been met in existing Flash content, requirements such as alternative text, controlled reading order, or widgets role information (e.g. for buttons). Unaddressed, Flash accessibility will continue decreasing, and screen readers will be left out of the richer Internet experiences forever. It is very important to develop new technologies to help both developers and screen reader users.

Therefore, we decided to develop an automatic accessibility repair algorithm for Flash content and a transcoding tool based on this algorithm. The tool can automatically add useful accessibility information, which is then presented by the Flash player to the screen readers through MSAA. This approach seeks to allow screen reader users to access a wide range of existing content without asking authors to modify the content. Simultaneously, this makes it possible to reduce the workload for content authors by using the automatic repair algorithm. This is the first tool with functions to enhance the accessibility of existing compiled Flash content.

Our approach works as a client-side transcoding system between a Web server and a Flash player. This tool directly changes the object model in the Flash player and controls the player's MSAA output. The current implementation focuses on adding alternative texts and information about buttons. This is some of the most basic accessibility information, but it is often missing or inappropriate in existing Web content.

In this paper, after introducing related work and the issues of Flash content accessibility, an overview of our automatic transcoding method for the Flash will be described. Then experimental results using our pilot system are presented, followed by a discussion of the results to date and future work.

2. RELATED WORK

There have been several research projects on automatic Web content analysis and adaptation [6, 7, 8, 9, 10], but there are few studies applicable to Flash content.

CSurf [6] is a client-side Web content adaptation system that automatically analyzes the visual structure and document content and offers an accessible user interface. Although the concept of this work is also useful for our work, using it to provide an alternative accessible user interface for Flash content involves other problems. Harper and Patel proposed an automatic method to make Web content accessible by extracting "gist" summaries. Asakawa *et al.* [7] proposed a transcoding server for accessible Web content adaptation, which also supports automatic content transformation. However, it only supports static HTML content. The work in [8] discusses an automatic HTML content analysis method based on visual cues. WebInSight [9] automatically produces alternative texts from images by using Optical Character Recognition (OCR). It is able to annotate 43% of the images with

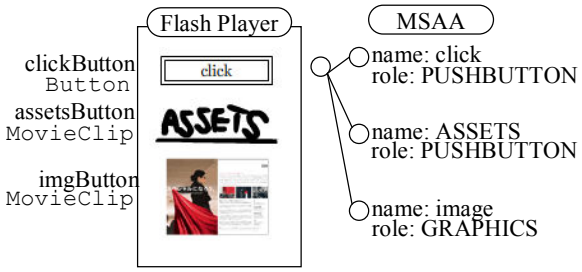

Figure 1: An example of MSAA output

94% accuracy. Accessmonkey [10] is a client-side scripting framework in a Web browser that is useful for providing an alternative accessible user interface. Users as well as Web developers can contribute to make webpages accessible. Some of the scripts on Accessmonkey automatically make Web content accessible. The WebInSight Accessmonkey script is one example and automatically generates alternative texts from images.

Not at run time but at authoring time, some tools can be used to provide accessibility to Flash content. Saito [5] *et al.* proposed a Flash accessibility checking tool applicable for existing Flash content. The AccRepair from HiSoftware [11] checks accessibility problems and offers helpful methods to repair the inaccessible Flash content. This accessibility tool is tightly integrated with the authoring features.

3. ACCESSIBILITY ISSUES FOR FLASH

3.1 Flash Accessibility Architecture

The accessibility architecture of Flash is based on the object model used by Flash content. Figure 1 shows an example of an accessible Flash presentation.

What is presented by the Flash player includes objects in the Flash content such as buttons, texts, and graphics, and each object belongs to one class. Though there are several predefined classes in Flash, we focused on the `Button` and `MovieClip` classes that may play the role of buttons. There are 3 objects of concern here, "clickButton" in the `Button` class and "assetsButton" and "imgButton" in the `MovieClip` class, as shown in Figure 1.

The behaviors of objects are described in their properties by scripts in ActionScript. For example, the following script code moves an object named "assetsButton" 10 pixels to the right when it is clicked and released.

```
assetsButton.onRelease = function() {
      this._x += 10;
}
```

Flash player outputs accessibility information via MSAA the visible objects and based on the properties of the objects such as their name and role. If an object is in the `Button` class then the role of the object in MSAA is PUSHBUTTON. If the object is in the MovieClip class then the role of the object is basically GRAPHICS, and if the GRAPHICS object is also associated with some active behavior then the role of that object is PUSHBUTTON. Therefore, the content in Figure 1 should be read as "click button, ASSETS button, graphics image".

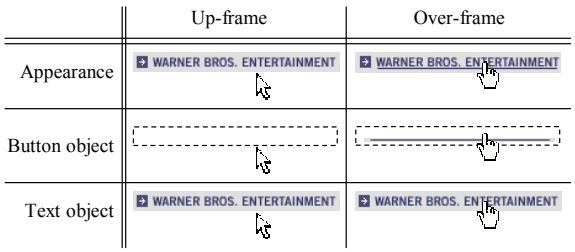

Figure 2: An example of unexposed buttons

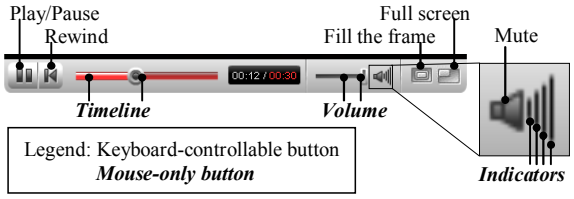

Figure 3: An example of unnecessary buttons

At the same time, the property value for the name can be set with ActionScript or an auto-label function. For example, in ActionScript the script producing that result could include

```
assetsButton._accProps.name = "ASSETS";
```

The auto-label function analyses the content and labels the buttons with text. The result for the objects in Figure 1 is "click button, ASSETS button, graphics image" will be read by screen readers.

3.2 Accessibility Issues

Our recent survey of Flash content accessibility [4] shows that half of the buttons had no alternative text, and none of the buttons had `_accProps` attributes. These results show that Flash content accessibility is not widespread, and this content has even worse issues related to the MSAA exposure of buttons. In addition there are many accessibility issues such as uncontrolled or incorrect reading orders and keyboard operability for GUI widgets.

In this study, we first focused on the issues of buttons, their alternative texts and links, because these fundamental problems to make Flash content accessible and they are frequently encountered in real-world environments. Therefore their repair should be beneficial for blind users.

3.2.1 Unexpectedly Unexposed Buttons and Links

Some buttons and links are not exposed through MSAA. They are exposed as just text objects or nothing is exposed. This issue is caused by the rules for exposing buttons that are followed in the Flash player. We found the following three major problems.

- **Up-frame of a `Button` object is blank**

A button object has four states, up-frame, over-frame, down-frame, and hit-frame, each of which may have distinct graphics. The status is changed by mouse interactions. From the state of up-frame, if the mouse is moved into the area of the button, the state is changed to over-frame. A problem occurs when up-frame has no graphic. In this case, the Flash player ignores the button and does not present the button to MSAA. This situation is not rare, and this issue is mentioned in Adobe's guideline [2]. Such a button appears suddenly when the mouse pointer is moved into the area of the button.

Figure 2 is an example of such buttons in Time Warner page[2]. In this case, the visual feedback for the button is likely to be a link anchor in an HTML page shown in "Appearance" column, but there are two objects, a button object and a text object, and the button object has an underline which appears when the mouse is over the object.

- ***_visible* attribute of a button object is false.**

The Flash player does not expose invisible buttons whose _visible attribute is false, but sometimes this attribute is used as part of a scripting technique. For example, after the mouse pointer is moved into a certain area an event will change the *_visible* attribute of the button to true, so the button is now "visible" for interactions and its MSAA information is also available. Sighted users can access such buttons by observing the appearances around the buttons. However, blind users cannot interact with the mouse or any pointing devices without visual feedback, and therefore blind users cannot access such buttons and remain unaware of their existence.

- **HTML elements described in Flash text object are not accessible.**

Flash player supports a small subset of HTML tags: `<a>`, `<b>`, `<br>`, `<font>`, `<img>`, `<i>`, `<li>`, `<p>`, `<span>`, and `<u>`, and these elements are rendered like HTML. The Flash player, however, only supports exposing plain text without tags. Therefore, the user cannot access links with `<a>` tags or alternative texts for `<img>` tags. Also, an `<img>` tag can embed not just graphics but also a Flash `MovieClip` object, and the embedded object is not visible from MSAA for blind users.

3.2.2 Unnecessary Buttons

The Flash player exposes information about all `MovieClip` objects having one or more active behaviors and each `Button` object, even if the `Button` object has no active behavior. This means unnecessary information is exposed by interactive GUI objects such as slider bars. Since the Flash player uses these simple strategies to expose information, the creators have to consider how the content will actually be exposed.

For example, in a YouTube page[3], the video player shows 5 buttons for play/pause, rewind, mute, fill the frame, and full screen, and 2 sliders for the timeline and volume (Figure 3). However, the MSAA information says that there are 13 buttons. Eight of these buttons are useless to blind users, because 2 sliders (represented by 4 buttons) are have no keyboard access and four others are not functional buttons, but merely represent indicator lines for the volume. Although these buttons have no alternative text, if there were only 3 buttons for play/pause, rewind and mute, then blind users could try clicking the buttons to discover their behaviors. However, so many unnecessary and mostly unused buttons just confuse blind users and discourage them.

[2] http://www.timewarner.com/corp/

[3] http://youtube.com/, actuary you would have to select a video to browse the video player.

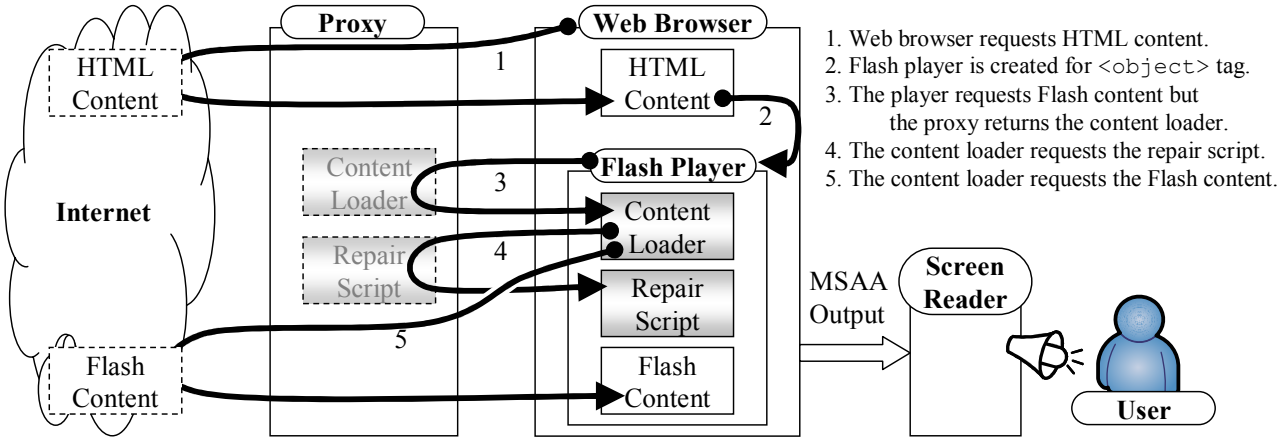

Figure 4: Overview of automatic transcoding system

3.2.3 Buttons without Alternative Texts

Buttons without alternative texts are shown as nameless buttons for MSAA output. JAWS [12], one of the most popular screen readers numbers these buttons such "16 button". Blind users cannot tell what to do with a numbered button unless they memorize the numbers and functions.

As already stated, our investigation shows that half of the buttons have no alternative text, though some are labeled with images of text or symbols that represent actions such as play, stop, or pause. Many buttons are completely unlabeled.

4. AUTOMATIC TRANSCODING SYSTEM

4.1 IMPLEMENTATION

We propose a new type of accessibility enablement for Flash content by a transcoding technique. A typical system setting of our repair method is shown in Figure 4. In our implementation, the proxy is implemented in Java and the content loader and the repair script are implemented in ActionScript. The flow of the system is:

1) When the user specifies the URL of a webpage, the Web browser requests and loads the HTML content referring to the Flash content.
2) The browser creates a Flash player for the `<object>` tag in the HTML content, specifying the location of the target Flash content.
3) When the Flash player tries to load the target Flash content, the proxy returns a "content loader" which is our special Flash content, instead of returning the target content. Our content loader redefines some of the built-in functions in the Flash player so it can inspect the object model in the target content.
4) The content loader loads some scripts from the proxy, which are also Flash content. In our configuration, the loader loads a "repair script" to repair the content. For other functions, the loader could load other scripts. For example, it might load bridge scripts for accessible user agents to access the internal Flash object model [13].
5) Finally, the content loader is ready to load the target content. The target location of the request is actually still the same as the request in 3), but now the proxy will actually pass the request to the Internet.

The repair script currently does not support analyzing dynamic changes of the content, though the repair tool could be launched whenever a repair is needed. Therefore the content could be repaired whenever the MSAA information is requested.

The repair algorithm is not limited to this particular configuration. It could be applied to other repair situations, such as while authoring, or to the Flash player itself. We will discuss these possibilities in the discussion section (see Section 6).

4.2 Automatic Repair Method

4.2.1 Expose Unexposed Buttons and Links

To expose unexposed buttons, the up-frame of these buttons should have some graphics. ActionScript, however, has no functionality to add graphics into the buttons. Therefore we insert alternative buttons to expose the information about such buttons such as a name. If the user clicks an alternative button, it executes the scripts describing the behavior of the original target button. For link anchors, a similar alternative button approach is used. When the user clicks such an alternative button, it changes the location of the Web browser to the location indicated in the target link anchor.

4.2.2 Hide Unnecessary Buttons

It is easy to hide unnecessary buttons by setting the silent property in a button's `_accProps` to true. However, it is difficult to be sure whether or not a target button is needed. In our method, we use these heuristics:

a) A `MovieClip` object is a button when it meets conditions a1, a2, and a3. A `Button` object is a button when it meets conditions a2 and a3.
   a1) The object has either an `onPress` or `onRelease` property.
   a2) The object's area is bigger than 50 square pixels ($width \times height > 50$).

a3) The object does not form a "bar". One exception is that if the object forms a bar with any text information then it is a button. A bar is defined as an object whose width is more than 8 times its height or its height is more than 8 times its width:

$$(\ height \times 8 < width \ \text{ or } \ width \times 8 < height\ )$$

Condition a1) implies what users expect from a button. The `onPress` and `onRelease` properties are essential for button behavior, so the button can respond to the user's click actions. Condition a2) eliminates very small objects as buttons, since even sighted users are unable to click on such small objects. The thin indicators in Figure 3 are examples of small objects that should not be regarded as buttons. Condition a3) implies slender objects tend to be a vertical slider or a horizontal slider widget rather than buttons. The timeline and volume sliders are good examples of such objects.

4.2.3 Provide Alternative Texts for Buttons

This problem is similar to the problem of alternative texts for images in HTML, and various techniques have been developed to automatically insert alternative texts. WebInSight [9] is an integrated system using various techniques. Many of these techniques can be applied to Flash content as well.

In our implementation, we use text information that is not used by auto-label function of the Flash player. There is a lot of text information within the Flash content in the form of dynamic text and static text. We analyze the overlaps between buttons and text strings and link them under in certain conditions.

Also, we can use the names of objects in the scripts, the "instance name" data. They are often meaningful as alternative text, because the content creator named them to help remember the meanings or uses of the objects while writing the script. All `MovieClip` objects and `Button` objects have an instance name, and the creator often names an object in a way that represents its role in the content. For example, a button that starts playing a movie is often named "play".

In our prototype system, buttons without alternative text are repaired according to the following heuristics:

a) A button and a text are linked if $B/I > 0.1$ and $T/I > 0.5$, where B is the button area, T is the text area, and the I is the area of intersection between B and T (Figure 5a). This condition implies the text overlaps the button, so it can be used as a label, but it excludes relatively small labels for the button.

b) A button which is not exposed because of the up-frame problem has no area, so such a button is associated with the nearest text such that $D < d$, where D is the distance to the candidate text and d is the distance from the closest corner of that text to its own center, as shown in Figure 5b.

5. EXPERIMENT

5.1 Experimental Method

Using our transcoding proxy described in Section 4, we conducted an experiment. For our experiment, 28 English webpages and 9 Japanese webpages were selected, all including Flash content. Of the 28 English webpages, 25 were used in our previous experiment to investigate the Flash content accessibility

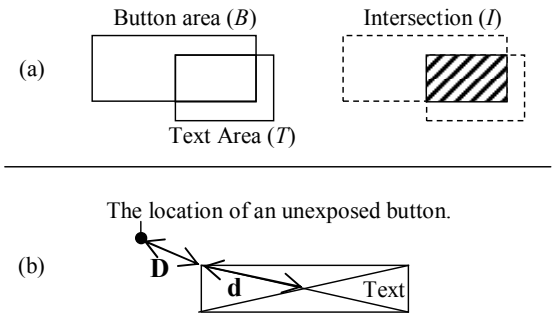

Figure 5: Illustrations for condition used in repair

[4], but some of the content had changed or disappeared since that time. In this sample, there were 29 Flash objects with buttons in 23 of the English webpages, with a total of 428 buttons, and 62% (265) did not have alternative text. There were 9 Flash objects with buttons in the 9 Japanese pages, contain 81 buttons, and 93% (75) had no alternative text.

The method of the experiment was the following:

1. Record the MSAA output of the content before repair, and count how many buttons should be exposed, how many buttons should be hidden, and how many buttons lack alternative text.
2. Repair the content automatically, exposing buttons, hiding buttons, and adding alternative text.
3. Record the MSAA output of the content after repair, and count the correctly exposed buttons, the correctly hidden buttons, and the buttons receiving useful alternative text.

5.2 Results

5.2.1 Detailed Comparison of MSAA Output

First, we will report the details of the repairs for two sample pages, the Time Warner (TW) home page[4] and the Disney Fairies' (DF) page[5]. The TW page has some links to related companies and a relatively simple Flash movie, and the DF page has Flash content that includes buttons and streaming video with controls.

Table 1 shows the details of the MSAA output, both before and after repair, and Table 2 shows statistics about the MSAA output. Before repair, 4 buttons were exposed in the TW page, and 19 buttons were exposed in DF page. In the TW page, 8 buttons were not exposed because they had no graphics in the up-frame. Of the 8 visible buttons, one is a link to another page and the rest are buttons to change the graphics and the targets of the links. In the DF page, the button named "45 Button", the handle of the slider to change the video position, is an unnecessary button because the button can only be controlled with the mouse. Therefore, in the TW page, 12 buttons (4 exposed and 8 unexposed) had no alternative text, and in the DF page, 15 buttons (excluding the "45 Button") should have alternative text.

After repair, the 8 buttons are exposed correctly with proper alternative text in the TW page, and all of the texts were found in

[4] http://www.timewarner.com/corp/

[5] http://disney.go.com/fairies/movies/videos.html

Table 1: Details of MSAA output before and after repair.

TimeWarner: Home (TW) http://www.timewarner.com/corp/			Disney Fairies - Videos - (DF) http://disney.go.com/fairies/movies/videos.html		
MSAA Output		Status	MSAA Output		Status
Before	After		Before	After	
-	View the Company Fact Sheet Button	T1	4 Button	Meet the Fairies Button	T1
8 Button	thumb2 Button	X1	7 Button	Books Button	T2
9 Button	thumb3 Button	X2	9 Button	Movies Button	T3
10 Button	thumb4 Button	X3	Movies Button	Movies Button	-
11 Button	thumb5 Button	X4	24 Button	Games & Activities Button	T4
-	AOL Button	T2	28 Button	Create a Fairy Button	T5
-	HOME BOX OFFICE Button	T3	Home Button	Home Button	-
-	TIME WARNER CABLE Button	T4	34 Button	Parents Button	T6
-	NEW LINE CINEMA Button	T5	41 Button	Videos Button	T7
-	TURNER BROADCASTING SYSTEM Button	T6	42 Button	Photo Gallery Button	T8
-	TIME INC. Button	T7	43 Button	History Button	T9
-	WARNER BROS. ENTERTAINMENT Button	T8	44 Button	Pause Button	N1
			45 Button (Unnecessary)	handle Button	X1
			47 Button	Rewind Button	N2
			50 Button	SoundController Button	N3
			Send to a friend Button	Send to a friend Button	-
			53 Button	item_1 Button	X2
			56 Button	item_2 Button	X3
			58 Button	item_3 Button	X4

Status	
Tn :	Repaired with Text
Nn :	Repaired with Instance Name
Xn :	Not Repaired

Table 2: Number of buttons before and after repair.

Page Title	TimeWarner: Home (TW)		Disney Fairies - Videos - (DF)	
	Before	After	Before	After
Total Buttons	4	12	19	19
Unxposed Buttons	8	0	0	0
Unnecessary Buttons	0	0	1	1
Buttons with Invarid Alternative Text	12	4	15	3

the text objects in the content, as indicated by **T** in the table. However, 4 of the previously exposed buttons were associated with names that are meaningless for blind users (**X**), because these buttons are merely graphical objects. In the DF page, 12 buttons received useful alternative text and 3 buttons were not repaired (**X**). Nine text objects (**T**) and 3 instance names (**N**), "Pause", "Rewind" and "SoundController", were used in the repair. Unfortunately, the "45 button" was not properly hidden by our current heuristics, because it is bigger than the size we used in the heuristics.

For the TW and DF pages, the MSAA output was effectively repaired and the accessibility was improved over the original content. In contrast to our expectations, the auto label function of the Flash player does not use effectively, so there is much text information can be used for repair.

5.2.2 Overall Results

In this section, we will describe the overall results for the 29 Flash objects in the 23 English pages.

Table 3 shows the status of buttons exposed or hidden, before and after repair. There were 20 unexposed buttons and 26 unnecessary buttons before repair. After the repairs, all 20 of those buttons were exposed properly, and 16 of the unneeded buttons are properly hidden, but 10 buttons were not hidden and 4 buttons were hidden as false positives. As a result, 428 buttons were exposed to MSAA.

Table 4 shows the fractions of accessible and inaccessible buttons and the numbers for three different repair statuses, repaired with text, repaired with instance name, or incorrectly broken. Buttons with invalid text (265 buttons) and unexposed buttons (20 buttons), making a total of 285 buttons were the targets to be repaired. Of these, 54% (155 buttons) were labeled with the correct text. Also, 4 buttons were labeled incorrectly, although they had valid text before the repairs. The alternative texts of the 155 repaired buttons came from 113 pieces of text information in the content and from 42 instance names in the scripts. There were 422 buttons in total, and 137 (32%) were accessible before repair, but 288 (68%) were accessible after repair.

Figure 6 shows the fractions of repaired buttons for various webpages, showing some of the content patterns and effects of our repair method. There are three patterns in the content, totally repairable, partially repairable, and not repairable.

Our results for the 9 Japanese pages were quite different from the results for the English pages. Of a total of 81 buttons, 10 unexposed buttons were repaired and 9 alternative texts were repaired with 2 texts and 7 instance names. Only 7% (6 buttons) were accessible before repair and 16% (15 buttons) were accessible after repair.

6. DISCUSSION

6.1 Practicality of the Transcoding Tool

The evaluation results (Section 5) showed the possibilities of the tool in improving the accessibility of Flash content in the real world. The automatic repair algorithm utilized only the information existing inside the Flash content, but it was able to drastically improve the accessibility of the buttons for screen reader users. One of the important observations from the results is the amount of text information inside the Flash content. In the overall results, 155 buttons were repaired and 73% (113 out of the 155) buttons were repaired with text information that already existed near the button object (Table 3). This ratio was higher than our expectations, since Flash player already has an automatic labeling function. This means there is room for improvement in the accessibility of Flash content by simply implementing our algorithms in the Flash player. Potential drawbacks should be considered in such an implementation, such as the performance impact and the ratio of false inferences. However, there are clear possibilities for improvement.

Another observation involves the ratio of repaired buttons based on instance names, 27% (42 out of 155) in the overall results (Table 3), and 24% (42 out of 172) of the buttons that were not repaired by the text extraction method were repaired using instance names.

This method is comparable to a text extraction method implemented in voice browsers for HTML documents. Each voice browser has its own text extraction algorithm for URL strings for image links that lack alternative text. This particular method was invented for the earliest voice browser development in the 1990s, and most screen readers still utilize the algorithm to compensate for missing alternative texts. We could not find empirical results on the ratios for HTML, but 54% seems to be much higher than for HTML. We believe this is because programmers follow naming conventions for objects in their programs, and tend to name the objects in relation to their functions to make it easy for the programmers themselves to understand and remember their meanings, with the beneficial side effect of understandable names for blind users. Therefore, integration of text extraction methods from instance names should be generally useful to compensate for buttons lacking alternative texts.

Our results also show that our methods of exposing unexposed buttons worked well, since 100% (20 out of 20) were repaired by our heuristic rules (Table 2). Since complete analysis of the unexposed buttons requires highly accurate static analysis of the scripts, which is not yet available for this type of application, it is good that the heuristic rules are sufficiently accurate as an alternative.

6.2 Effect of Ideographic Characters

The results for Japanese pages were significantly worse than for English pages. The reasons seem to be related to the complexity of ideographic characters. There are only around 50 characters (including capital letters) for alphabetical languages, but there are about 5,000 characters used in China and around 2,000 characters used in Japan. Therefore, it is difficult to install a variety of fonts in a personal computer compared to the range of alphabetic fonts. One result is that content creators tend to use images to present various fonts for ideographic characters, and that affects the ratio of text information inside Flash content. Also, the instance names are declared in English, not Japanese, since the scripts are basically written in Latin-like characters. The problems with ideographic characters are beyond the scope of our assumptions, and future research will have to consider them.

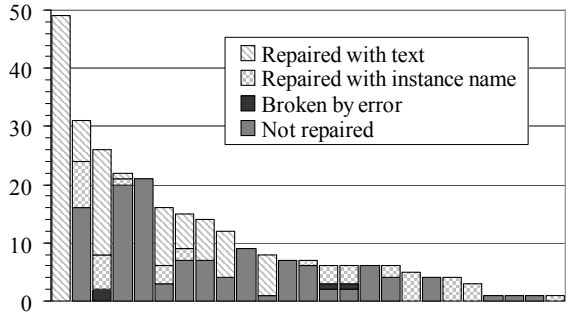

Figure 6: Fraction of repaired and not repaired buttons in the each page.

Table 3: Status of buttons before and after repair

	Before	After
Total exposed buttons	428	428
Unexposed buttons	20	0
Unnecessary buttons	26	10
Hidden by error	-	4

Table 4: Fraction of accessible and inaccessible buttons with repaired status.

	Accessible	Inaccessible
Befor Repair	137	285
Repaired with text	113	△ 113
Repaired with instance name	42	△ 42
Broken by error	△ 4	4
After Repair	288	134

6.3 Applicability of the Repair Algorithm

Our approach to improve the accessibility of Flash content is also applicable in other configurations. The following are some possible applications.

- **Repair Tool for Authoring Time**

Adobe's authoring tool for Flash can extend the functions using a JavaScript API. The AccRepair for Flash from HiSoftware [14] is an extension for this authoring tool. This tool provides functions to check errors and to show a list of errors using a dialogue interface with messages like "The object has no alternative text". Although the tool only informs us about the errors, a newer tool with our repair approach could offer repair candidates or automatic repairs. Suggesting repair candidates could reduce the content creator's workload.

- **Flash Player's Algorithm for Exposing Buttons**

Our approach could be applied to Flash player's button-exposing algorithms. Our application was not implemented with the

player's collaboration, though it could become a more effective approach than our early system using a proxy and repair scripts.

- **Automatic Annotation Generator**

We could provide an alternative interface for the Flash content by annotation-based transcoding or with another specialized browser for non-visual users. One prototype is our aiBrowser [13]. Also, our repair method could be used to generate annotations for aiBrowser. Annotation-based transcoding systems and alternative interfaces can benefit from annotations which are often created by people. Therefore, the automatic repair techniques could help the authors of annotations.

7. CONCLUSION

In this paper, after discussing the existing accessibility problems with Flash content, we described an automatic transcoding method for Flash that works between a Web server and the Flash player. It analyzes the object model of the Flash, directly changes it, and thus controls the player's MSAA output. As first steps, we focused on supplementing the alternative text and repairing the button information, both of which are often missing or inappropriately extracted in existing Flash content. These two items are still among the most important for Web accessibility, and most Flash content tends to have these kinds of problems.

Our experiment using the pilot system showed that 54% of the missing alternative texts could be added automatically for buttons in the tested websites. We conclude that our approach could drastically improve the accessibility of Flash content.

Our future work is first to expand the coverage of our approach. For example, Flash content is often used in a "windowless" mode. This makes the Flash content completely inaccessible to screen readers, because in windowless mode there is no MSAA information available. Another area needing work is to modify the reading order, since this is complicated for screen readers. Next, we plan to apply machine learning techniques, instead of just applying heuristic algorithms.

Through these efforts, we hope to improve the accuracy of automatic transcoding as much as possible. At the same time, we will pursue the goal of completely accessible and usable interfaces, not just for Flash, but also for any other types of dynamic content. We plan to integrate metadata-based transcoding in the future.

8. ACKNOWLEDGEMENTS

We would like to thank Bob Regan and Andrew Kirkpatrick from Adobe Systems, Inc., for their technical support and for sharing their valuable experiences with Flash accessibility.

9. REFERENCES

[1] Section 508 of the Rehabilitation Act; see http://www.section508.gov/.

[2] Regan, B. *Best practices for accessible Flash Design, 2005.* http://www.adobe.com/resources/accessibility/best_practices/best_practices_acc_flash.pdf

[3] W3C, *Understanding WCAG 2.0*. 2007. http://www.w3.org/TR/UNDERSTANDING-WCAG20/

[4] Asakawa, C., Itoh, T., Takagi, H., and Miyashita, H. Accessibility Evaluation for Multimedia Content. In *Proceedings of Universal Access in Human-Computer Interaction (UAHCI 2007)*. To appear.

[5] Saito, S., Takagi, H., and Asakawa, C. Transforming Flash to XML for Accessibility Evaluations. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2006)*. pp. 157-164.

[6] Mahmud, J., Borodin, Y. and Ramakrishnan, I. CSurf: A Context-Driven Non-Visual Web Browser, In *Proceedings of the Sixteenth International World Wide Web Conference (WWW2007).*

[7] Asakawa, C., and Takagi, H. Annotation-Based Transcoding for Non-visual Web Access. In *Proceedings of the 4th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2000)*. pp. 172-179.

[8] Yang, Y. and Zhang, H. HTML Page Analysis Based on Visual Cues. In *Proceedings of the Sixth international Conference on Document Analysis and Recognition*, 2001, pp. 859-864.

[9] Bigham, J. P., Kaminsky, R. S., Ladner, R. E., Danielsson, O. M., and Hempton, G. L. 2006. WebInSight:: making web images accessible. In *Proceedings of the 8th international ACM SIGACCESS Conference on Computers and Accessibility,* pp. 181-188.

[10] Bigham, J., and Ladner, R. *Accessmonkey:* A Collaborative Scripting Framework for Web Users and Developers. In *Proceedings of International Cross-Disciplinary Workshop on Web Accessibility (W4A 2007)*.

[11] HiSofware, AccRepair® for Flash. http://www.hisoftware.com/accrepair_flash/

[12] Freedom Scientific, JAWS. http://www.freedomscientific.com/

[13] Miyashita, H., Sato, D., Takagi, H., Asakawa, C. aiBrowser for Multimedia – Introducing Multimedia Content Accessibility for Visually Impaired Users. *Submitted to ASSETS 2007.*