# Accessmonkey: A Collaborative Scripting Framework for Web Users and Developers

Jeffrey P. Bigham and Richard E. Ladner
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98115 USA
{jbigham|ladner}@cs.washington.edu

## ABSTRACT

Efficient access to web content remains elusive for individuals accessing the web using assistive technology. Previous efforts to improve web accessibility have focused on developer awareness, technological improvement, and legislation, but these approaches have left remaining concerns. First, while many tools can help produce accessible content, these tools are generally difficult to integrate into existing developer workflows and rarely offer specific suggestions that developers can implement. Second, tools that automatically improve web content for users generally solve specific problems and are difficult to combine and use on a diversity of existing assistive technology. Finally, although blind web users have proven adept at overcoming the shortcomings of the web and existing tools, they have been only marginally involved in improving the accessibility of their own web experience.

As a first step toward addressing these concerns, we introduce *Accessmonkey*, a common scripting framework that web users, web developers and web researchers can use to collaboratively improve accessibility. This framework advances the idea that Javascript and dynamic web content can be used to improve inaccessible content instead of being a cause of it. Using Accessmonkey, web users and developers on different platforms with potentially different goals can collaboratively make the web more accessible. In this paper we first present the Accessmonkey framework, describe three implementations of it that we have created and offer several example scripts that demonstrate its utility. We conclude by discussing future extensions of this work that will provide efficient access to scripts as users browse the web and allow non-technical users be involved in creating scripts.

## Categories and Subject Descriptors

K.4.2 [**Social Issues**]: Assistive technologies for persons with disabilities; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces


*W4A2007 - Technical Paper* May 07–08, 2007, Banff, Canada. Co-Located with the 16th International World Wide Web Conference.


## General Terms

Design, Human Factors

## Keywords

Accessmonkey, web accessibility, web scripting, web transcoding, alternative text, Greasemonkey

## 1. INTRODUCTION

Efficient web access for users employing assistive technology remains elusive because many web pages are designed with inadequate regard for usability and accessibility concerns [16]. This is particularly true for blind web users who access the web using screen readers that present a web experience that is devoid of the rich visual structure that often organizes information on the web. While web developers are in the best position to resolve accessibility problems, they are often ill equipped to do so effectively. Existing web standards, such as World Wide Web Consortium (W3C) Web Content Accessibility Guidelines [11] and Section 508 of the U.S. Rehabilitation Act, offer quantifiable rules to which web developers can strive to adhere, but creating truly accessible content requires a more subtle and discerning evaluation [26]. The accessibility of the web would not necessarily be optimized even if web developers would follow established web accessibility guidelines.

Despite efforts to promote accessible web development, web developers often fail to implement even basic elements of accessibility standards. For many web developers, a lack of appropriate experience hinders their ability to adequately judge the accessibility of web content, and even experienced web developers may require more time to produce a visually appealing web page that is also accessible [28]. Available tools can help spot deviation from the easily quantifiable portions of established standards, but they fail to adequately guide developers through the process of creating accessible content. Perhaps it should be unsurprising that, when faced with a deadline to release new web content or when faced with a daunting backlog of web pages to be updated, web developers often delay a full consideration of accessibility. Web developers need tools that can efficiently guide them through the process of creating accessible web pages, and blind users need tools that can help them overcome accessibility problems even when developers fail.

In an effort to make the web independently accessible to blind users, a number of projects have created systems that capable of automatically transcoding documents in order to render them more accessible [14, 20, 19, 17]. These tools

are potentially useful, but their utility is not maximized. They either require web users to employ a specific browser or platform or needed a host machine on which to run. The transformations made by these tools help the web users that use them, but are not easily utilized by web developers who could often benefit from the same underlying technology for automatic improvements. Finally, the opportunity for web users to influence this technology or to independently suggest new accessibility improvements is limited.

As a first step at addressing these concerns, we introduce *Accessmonkey*, a scripting framework that helps web users and web developers collaboratively improve the accessibility of the web. Accessmonkey helps web users automatically and independently transcode web content according to their personal needs on a number of browsers and platforms. Accessmonkey helps web developers leverage the same transcoding technology to more efficiently create accessible content in a way that integrates into current developer workflows. Many existing tools and systems address accessibility problems, but they often require a specific browser or require the user to install a separate tool. Often these tools simply aren't available to users and exist only as proof-of-concept demos. When they are available, they can be difficult for users to independently improve and difficult for developers to integrate into existing tools. Accessmonkey provides a convenient mechanism for sharing techniques developed and insights gained. The framework allows both web users and web developers to collaboratively improve the accessibility of the web by leveraging the work of those that have come before them. Users can effect the accessibility of the web by writing a script and other users can immediately use and adapt the script to fit their own needs. Web developers can use the script to improve the accessibility of their web pages automatically, reducing the job of providing accessibility information to a more efficient editing and approval task. To allow as many users as possible to utilize our framework, we offer several implementations of it that work on multiple platforms and in multiple web browsers.

In this paper, we consider a number of examples that demonstrate how the Accessmonkey framework allows both web users and web developers to benefit from the same underlying script. We first considered how to adapt techniques for automatically supplying alternative text for web images presented in previous work [14] to assist both web users and developers. The system was originally designed to assist web users, but web developers could also utilize the underlying technology to more efficiently provide alternative text for images on the web pages that they create. Implementing WebInSight as an Accessmonkey script allows users to run WebInSight either in a client mode that transparently inserts alternative text or in a developer mode that suggests proposed changes and allows the web page to be saved. We also implement scripts for context-driven web browsing, for personalizing web pages according to the needs of users and for making inaccessible dynamic content accessible. These scripts are described in more detail in Section 4.

The contributions of our work include the following:

1. We describe how Javascript and dynamic content can be used for accessibility *improvement*.

2. We introduce a framework for designing scripts that enables web users and web developers to utilize the same underlying technology and avoid duplicating work.

3. We demonstrate that Accessmonkey scripts are powerful enough to implement a number of automatic accessibility improvements, including a version of WebInSight for both web users and web developers.

4. We reimplement several previous systems designed to improve accessibility as Accessmonkey scripts. These systems can now be used on more web browsers and platforms by both web users and developers.

# 2. RELATED WORK

Automatically transcoding web content to better support the needs of web users has been a popular topic in web accessibility research for almost a decade, especially in relation to improving access for blind web users. Several systems have been developed that allow web users to arbitrarily transcode web content via scripting and were motivated in part by web accessibility. Accessmonkey seeks to allow both web users and web developers to collaboratively create scripts that direct the automatic transcoding of web content in a way that helps both groups of users efficiently increase web accessibility.

## 2.1 Scripting Frameworks

Greasemonkey was introduced in 2003 by Mark Pilgrim. The project was partially motivated by a desire to provide web users with the ability to automatically transcode web pages into a form that is more accessible. Several examples of such scripts are offered in the book Greasemonkey Hacks [29]. NextPlease! is an example of a Greasemonkey script that has become quite popular among blind web users [35]. This script allows web users to define key combinations that simulate clicks on links that contain user-defined text strings. Currently, to implement similar functionality in their own web pages, web developers cannot directly leverage the code changes made by users of a script like NextPlease! and, instead, must independently decdie on these changes. Accessmonkey extends the original idea behind Greasemonkey by providing a mechanism by which web users who write scripts can also make their scripts useful to web developers. We also provide a web and proxy implementation of Accessmonkey that opens the system to use by additional users.

Scripts designed to automatically improve accessibility are already available as Greasemonkey scripts. Popular existing user scripts include those for automatically detecting and removing distracting ads and those that add new functionality to popular web sites like google.com and hotmail.com. Others automatically add accessibility features to web pages. Often these scripts present solutions that web developers could have included (support for access keys, proper table annotation, etc.), while others address problems that apply to particular subsets of a web pages visitors (high-contrast colors, larger fonts, etc.). Some of the most popular scripts are those that add access keys to popular web sites and those that adapt web pages to be easier to view by people with low vision. A large repository of Greasemonkey scripts is available at userscripts.org, including 49 scripts targeted at accessibility. These scripts alter pages in ways that users have found helpful, such as adding heading tags (<h2>) to the Google results page. Many scripts are available, which suggests that a number of individuals are willing to write such scripts and that many web users find them useful.

Another web scripting framework is Chickenfoot, which allows users to programmatically manipulate web pages using a superset of Javascript that includes methods specific to web browsing [15]. The interface of Chickenfoot is designed to make web page manipulation easier, although it still requires some level of programming knowledge. Platypus, another Firefox extension, seeks to remove this requirement as well by providing an interface that allows users to manipulate the content of web pages by simply clicking on items [33]. Neither of these systems offers a mechanism that allows users to save altered web pages, but both could be a useful as a way to allow users to more easily create scripts.

## 2.2 Accessibility Evaluation

The automatic evaluation of the accessibility of web pages has been a popular topic in both research and industry, and has resulted in the availability of many evaluation tools [4, 9, 1, 8]. Most of these tools have focused on assisting developers in meeting quantifiable accessibility standards, such as the W3C Web Content Accessibility Guidelines [11] or Section 508 of the U.S. Rehabilitation Act. The research community has sought to extend the capabilities of evaluation tools to allow for the automatic detection of more subtle usability and accessibility concerns [21], but tools that can do this well have yet to be developed. Mankoff *et al.* noted that an effective method for identifying accessibility problems in a web page is to have it reviewed by multiple web developers using a screen reader, but that blind web users could effectively detect subtle usability problems [26]. Accessmonkey allows both groups to collaboratively assist in the evaluation and remediation of web content, but neither group must rely on members of the other before accessibility improvements can be implemented.

## 2.3 Automatic Accessibility Improvement

Previous work has explored both automatically improving the accessibility of web pages and creating design tools that will facilitate the production of accessible content by web developers. To take advantage of these systems, content has generally needed to be processed by the web developer using a specialized tool, displayed using a specialized web browser [31], or transcoded for users on-the-fly. A number of systems have explored automatically transcoding documents in order to positively effect accessibility [19, 20, 17, 14].

Harper *et al.* suggested three alternative approaches for implementing the transcoding of documents in the context of making browsing more efficient for blind users by supporting content preview by probing [17]. The authors proposed that transcoding could be implemented in a browser extension or plug-in, as Javascript added to a web page or in a proxy or web server. In this work, the option to transcode documents using a proxy was chosen and a number of systems have chosen this strategy for similar reasons [20, 14, 34]. The authors discounted using a browser extension or plug-in because of the need to create and support several implementations to reach web users employing different web browsers on different platforms.

Using Javascript to transcode content was not chosen because, at the time, many of the web browsers and screen readers used by blind users did not offer adequate support for Javascript. The choice to transcode documents at the proxy level, however, acts as a potential bottleneck, is less private and is potentially less customizable. Using a browser plugin, as other systems have done [18, 35], limits the audience to those using a particular web browser. Accessmonkey uses Javascript to transcode pages and allows scripts written for it to be used on many different platforms and in many different browsers. Users can personalize what selection of available transcoding services they would like to apply to web pages that they view and can also write or modify their own scripts. The most recent versions of popular screen readers offer adequate Javascript support [6, 7].

Implementing transcoders as scripts also has the potential to make extending the techniques that they encompass to web development tools easier. While several systems have suggested that techniques used to automatically transcode documents could also be used to help web developers more easily create accessible content [8, 34], this process has often been difficult to directly integrate into existing developer tools. Accessmonkey allows scripts to be written once and included in a variety of tools used by both web users and web developers. To our knowledge this is the first example of a system that unifies the automatic accessibility improvement targeted at web users and web developers in an extensible way.

Despite the similarity in the underlying technology, little work has been devoted to assisting web developers in automatically improving the content of their web pages through specific suggestions. Many tools used for evaluation display general advice about how to rectify problems that are discovered [5, 4, 1], but a web developers must still be skilled in accessibility to correctly act on this advice. The guidance provided is usually general and is often drawn from the standard against which the tool is evaluating the web page. A-Prompt, for example, guides web developers through the process of fixing accessibility problems. Related systems have been designed to assist users in annotating web content for the semantic web [12] and Plessers *et al.* showed how such annotations could be automatically generated as a direct result of the web design process [30]. Accessmonkey scripts can utilize the same technology used to assist web users to help web developers.

## 2.4 WebInSight

WebInSight improves the accessibility of web images by automatically deriving alternative text [14]. This system was shown to be capable of automatically providing labels for 43.2% of web images that originally lacked alternative text with high accuracy by using the title of the linked page for linked images and by applying optical character recognition (OCR) to the images that contained text. WebInSight was originally developed for web users, but it quickly became apparent that the system could easily be adapted to assist web developers in the task of choosing appropriate alternative text. Many available accessibility tools inform web developers when images lack alternative text, but few suggest alternative text or automatically judge the quality of the alternative text already provided.

A system for web developers could make appropriate suggestions for alternative text by using techniques developed for WebInSight. Both ALTifier [34] and A-Prompt [1] used heuristics to perform a similar function for developers, but the WebInSight system is able to do so more accurately. The system also leverages a supervised learning model built from contextual features in order to identify alternative text that is likely to be incorrect [13]. The ability to automati-

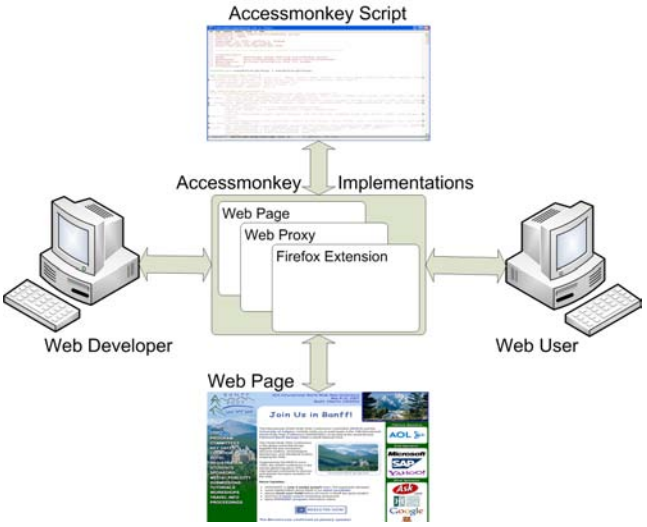


**Figure 1: Accessmonkey allows web users, web developers and systems for automated accessibility improvement to collaboratively improve web accessibility.**

cally judge the quality of alternative text could potentially improve the user experience by eliding alternative text that is likely to be inappropriate. Using the WebInSight Accessmonkey script, web developers are not only told that an image lacking alternative text should be supplied it, but also whether the alternative text provided is likely to be correct.

# 3. ACCESSMONKEY FRAMEWORK

The framework exported by the Accessmonkey system allows users to edit web pages using Javascript. The Greasemonkey Firefox extension [2] is one of the most successful examples of an open scripting framework and exposes the framework from which Accessmonkey is derived. The Greasemonkey extension allows users to inject their own scripts into arbitrary web pages and these scripts can then alter web pages automatically. The main difference between Accessmonkey and Greasemonkey is that Accessmonkey natively supports web developers by providing a mechanism for web developers to edit, approve and save changes that have been made to web pages by user scripts. Figure 1 shows the relation between the components that use the Accessmonkey framework.

Accessmonkey is designed to support multiple implementations which may be placed on a remote server, on the user's computer or directly integrated into web tools. Accessmonkey scripts can be used in different browsers and on different platforms because of the near ubiquity of Javascript. While Greasemonkey is only available on Mozilla web browsers, other major web browsers, such as Internet Explorer, Safari and Opera, already afford similar capabilities and can often run Greasemonkey scripts unaltered. Incapability concerns remain because of differences between the implementations of the ECMAscript standard (commonly known as Javascript) used by different browsers. The primary implementations of ECMAScript are JScript as implemented by Internet Explorer and Javascript as implemented by other popular browsers, including Firefox and Safari. Despite these limitations, web developers are accustomed to writing scripts that are compatible with the different implementations of ECMAScript.

Many web browsers and screen readers do not work well with Javascript code, and some ignore it altogether. These browsers are currently left out. For example, web browsers that cannot be updated (such as those used by some PDAs) may not be capable of handling Javascript scripting, such browsers are likely to contain have this capability in the future. We believe that the amount of web content currently available that utilizes Javascript will encourage this to change, but we are currently investigating an implementation of Accessmonkey capable of altering web pages on a remote server before delivering the content to the web user. The scripts presented in this paper have been tested with Window-Eyes 6.0 [6].

Accessmonkey gives users the option of running solely on the client side, as opposed to systems that are designed to run on a remote server. A disadvantage of our approach is that Javascript limits the space of possible transcoding operations that can be performed, but, as shown in Section 5, many of the transcoding operations that have been previous suggested can be achieved using Javascript only. Furthermore, as we discuss in Section 6, future versions of Accessmonkey may allow Java code to be bundled with Accessmonkey scripts in order to enhance their functionality.

## 3.1 Writing Scripts

An Accessmonkey script shares its structure with a Greasemonkey script but relies on additional functionality provided by an Accessmonkey implementation. Greasemonkey scripts can be run unaltered in Accessmonkey implementations. Figure 2 shows that Accessmonkey scripts are a specific subset of possible Greasemonkey scripts, which are, in turn, a subset of all Javascript scripts. Accessmonkey scripts are expected to provide a mechanism for users to view, edit and approve changes that are automatically made by the script when appropriate and rely on functionality exposed by the Accessmonkey implementation to facilitate this. Accessmonkey differentiates two modes of operation: a user mode in which changes are automatically made to a page and a developer mode in which users are provided an interface that allows them to edit, approve and save automatic changes. A script can query the implementation in which it is running to determine the mode that is currently activated and to obtain a pre-defined area in which the script can place its developer interface. The implementation provides functionality that coordinates which script's developer interface should be displayed and allows changes that have been made to the web page to be saved.

To write a script, a user must be able to write Javascript code, but any user can use an existing script. Future versions of this tools will include a mechanism to help users locate applicable scripts. We also plan to explore ways of enabling users who are not technically savvy to create scripts (see Section 6).

## 3.2 Requirements

An Accessmonkey implementation requires only a few elements. First, the implementation must have the capabilities of Greasemonkey. Specifically, it must be able to load a web page, add custom Javascript to it and execute this Javascript. The implementation must provide the stan-

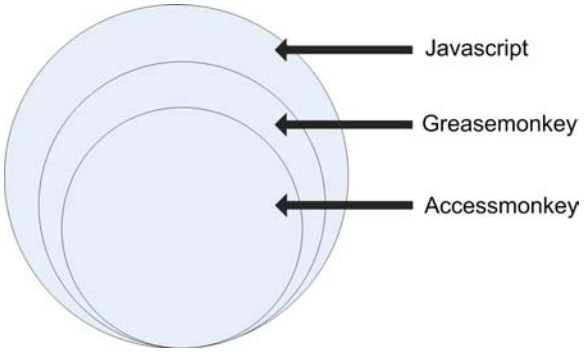


**Figure 2: Accessmonkey scripts are a subset of Greasemonkey scripts which are a subset of Javascript scripts.**

dard Greasemonkey API [29] and two additional methods required for Accessmonkey features. The first method returns a Boolean value indicating whether the system is in developer mode or user mode, which allows user scripts to appropriately alter their presentation and editing options. The second method returns a reference to a **div** element that represents the script's development area. The script may append elements to this element to form its developer interface. This interface supports the user in viewing, editing and approving changes automatically suggested by the script. Each implementation must also provide a mechanism for saving changes that were made to the web page by the user. Figure 3 shows an implementation of Accessmonkey running a script. The select boxes and buttons at the top of developer interface displayed in this screenshot allow users to switch the tool that is currently being applied, switch usage modes, and save changes that have been made to the web page by the script.

## 3.3 Developer Workflow

An important consideration for the usability of our system is how it will fits into the workflow of web developers. We hope to address one of the main shortcomings of accessibility tools cited by developers, which is their inability to integrate well into current developer workflows [21]. Designing a tool that easily integrates into the wide diversity of products used by developers is impractical, but the implementations provided allow our system to be immediately available to web developers. Accessmonkey integrates into the developer workflow by first allowing developers to make and edit potential changes to the document and then providing a mechanism for the developer to save changes.

Current implementations will allow many web developers will be able to use the system immediately, but web pages that are generated in a dynamic way using underlying data sources and web page templates will be unable to leverage the system. Developers of such systems will still benefit because Accessmonkey stresses the importance of providing suggestions in addition to identifying problems. Ideally, the system would be implemented directly into the tools already used by web developers. The simple and open scripting framework exposed by Accessmonkey allows users to develop such implementations that more closely integrate into these tools. Our current implementations still requires an external program for accessibility improvement, but previous work has shown that an improved workflow that still involves the use of several applications can nevertheless dramatically increase efficiency [23].

# 4. IMPLEMENTATIONS

Users employ a variety of web browsers on a number of platforms, and, similarly, web developers use a number of development tools, many of which are proprietary. Accessmonkey should be easy to integrate into these tools. The decision to implement Accessmonkey as a scripting framework using Javascript allows for a number of implementations to be developed because many platforms already contain support for Javascript.

Creating implementations of Accessmonkey that integrate directly into all possible tools used by users and developers is impractical. Instead, Accessmonkey provides a simple framework which can be extended to other tools and platforms by users. We have created three implementations of Accessmonkey that cover a wide variety of use cases: a Firefox extension, a stand-alone web page, and a web proxy. Web users and developers can access the full range of Accessmonkey functionality by using any of these implementations. In future work, we plan to create implementations of Accessmonkey that are directly integrated into additional web browsers and web development tools. To support browsers incapable of running Javascript code, we also plan to explore a proxy-based implementation that can apply user scripts to a web page before forwarding the transcoded page to users.

## 4.1 Firefox Extension

The Firefox Extension implementation is a straightforward adaptation of the existing Greasemonkey extension, which was the motivation for Accessmonkey and already provides much of the required functionality. To allow the extension to fully support Accessmonkey scripts, we enhanced the extension by adding the Accessmonkey-specific methods described earlier in Section 3.1 and added a toggle that allows users to switch between user and developer mode. Finally, we added the ability to save changes that were made to the web page. A screenshot of the resulting system is shown in Figure 3.

## 4.2 Web Proxy

The web proxy implementation of Accessmonkey is implemented as an Apache module. In proxy mode, this module simply inserts a script containing the Accessmonkey code into each web page that a user visits. Disadvantages of proxy-based approaches were discussed previously in Section 2.3, but for some users it is the most viable option because it does not require Firefox. Currently, the administrator of the proxy is responsible for adding new user scripts, although future versions may allow users to upload scripts and have them immediately included in their suite of Accessmonkey scripts. Eventually, we would also like to offer a proxy-based solution that processes web pages on the fly according to user scripts as the user browses the web.

For security reasons, some methods in the Greasemonkey API do not have direct analogies in Javascript. The Greasemonkey method used to retrieve the content of an arbitrary URLs is useful for allowing scripts to include information derived from web services or integrated from other web sites. For security reasons, the analogous Javascript functionality is restricted to retrieving content from the same do-

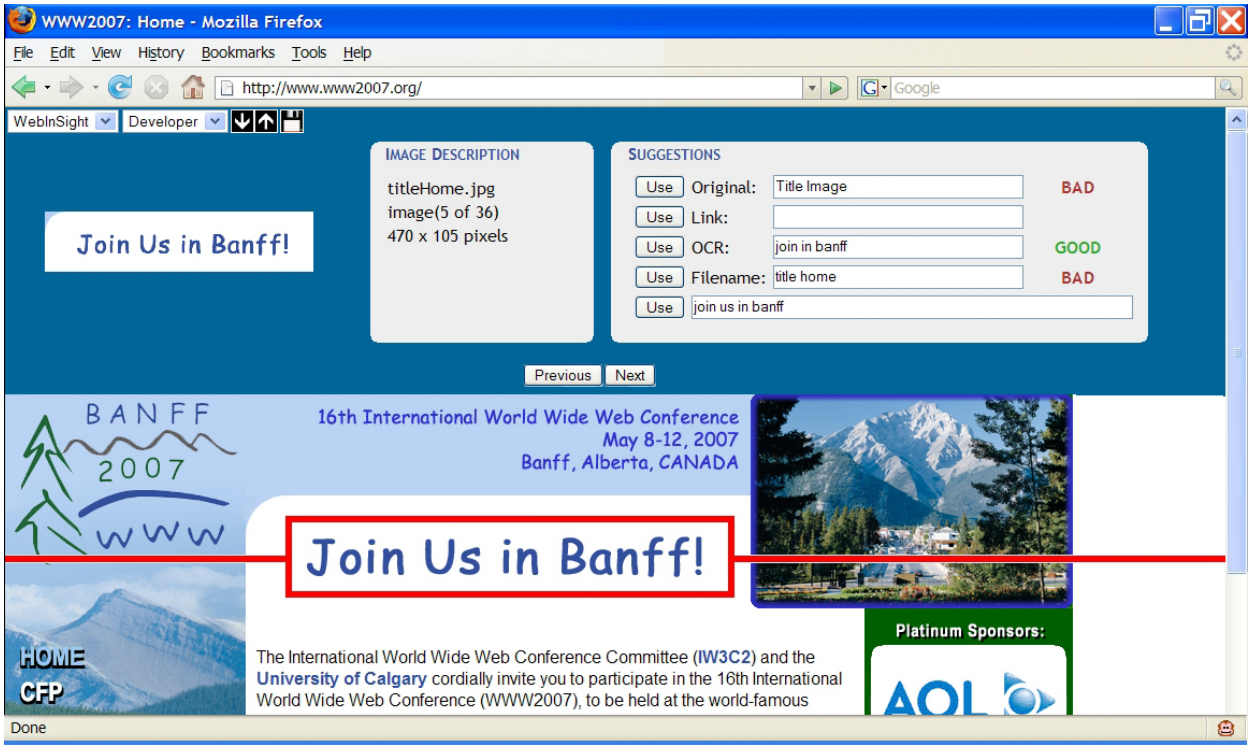


**Figure 3: A screenshot of the WebInSight Accessmonkey script in developer mode applied to the homepage of the International World Wide Web Conference. This script helps web developers discover images that are assigned inappropriate alternative text (such as the highlighted image) and suggests appropriate alternatives. The developer can modify these suggestions, as was done here, to produce the final alternative text.**

main as where the script is located. To allow Accessmonkey scripts running in this implementation to incorporate data not available on the original domain of the web page, this implementation allows scripts to request the content of any URL from the proxy, which effectively anonymizes these requests. To avoid abuse, the proxy implementation limits use of the system to registered users.

## 4.3 Web Page

Several popular evaluation tools are web-based [4, 9, 10]. Visitors to these web sites can enter the URL of a web page that they would like to evaluate and the tools will analyze it. Such tools are convenient because they don't require users to install additional software and can be accessed from anywhere. Because the evaluation is done remotely, these tools require the web page to be publicly available and, therefore, may be inappropriate for accessibility evaluation of private or pre-release web pages. To allow users of our system additional flexibility, we have created a web-based version of Accessmonkey.

Our web implementation of Accessmonkey requires a browser that supports Javascript, but requires the user to neither use a specific browser nor install an extension, which opens Accessmonkey scripts to potential users that prefer Internet Explorer, Opera or another web browser. This implementation allows a large portion of web users and developers to use Accessmonkey scripts.

Our web page version of Accessmonkey is implemented using a variation on the module for the Apache Web Server that we developed for our proxy implementation. When users visit the Accessmonkey web page they are first asked for a URL. The system then fetches that URL and alters the page returned in a way that allows it to be displayed at a local address. All of the URLs in each web page are automatically translated into fully qualified URLs that are directed through the proxy. This Accessmonkey implementation uses the same techniques for producing the full Accessmonkey API that were required in the proxy implementation discussed previously.

## 4.4 Future Implementations

Future implementations will allow more web users and developers to use Accessmonkey on more platforms. Turnabout is a plug-in for Internet Explorer that is similar to Greasemonkey and allows user-defined scripts [3]. It could be modified to provide the added functionality required of an Accessmonkey implementation. We would also like to add the capability of running Accessmonkey scripts directly in web development tools. SeaMonkey Composer and Adobe Dreamweaver are attractive options because they already support Javascript, although we would like to eventually create Accessmonkey implementations for other popular tools, such as Microsoft FrontPage.

# 5. IMPLEMENTED SCRIPTS

We have implemented several scripts for our system that demonstrate the usefulness of the Accessmonkey architecture. Our current implementations are both strengthened and limited by their restriction of only using Javascript. Re-

stricting our scripts to Javascript allows them to be easily extended to many other platforms, but comes at the cost of accepting the limitations Javascript. For instance, our scripts cannot gain pixel-level access to images. One method of circumventing this limitation is to utilize web services, as we did for our WebInSight script so that it could access OCR functionality. In this section, we further demonstrate the diversity of powerful transformations that can be accomplished using Accessmonkey and how they can be leveraged by both web users and developers.

## 5.1 WebInSight Script

The inspiration for Accessmonkey came from our desire to allow web developers to leverage the technology developed for WebInSight (described in Section 2.4) to make the creation of accessible web pages easier. Our belief is that web developers would be more likely to create accessible content if they are given specific suggestions on how to do so instead of being forced to go it alone. Our WebInSight Accessmonkey script is an example of such an approach.

A screenshot of the Accessmonkey system running the WebInSight script is shown in Figure 3. The developer interface provides web developers with functionality to quickly approve and edit the alternative text assigned to each image in the page. To assist in this process, the system provides several automatically-computed suggestions for appropriate alternative text that web developers can select with a single click after optionally editing the suggestion.

The script automatically computes all suggestions, except for the OCR suggestion, which is retrieved from a web service. Each suggestion is automatically evaluated by the system and the best suggestion is always displayed in the lowest text box. The interface allows developers to skip images that are unlikely to be informative and assign these images a zero-length alternative text. The system does not provide developers with a button that automatically applies alternative text to all images because the system's suggestions are not always correct. Following the spirit of Accessmonkey, blind web users can also utilize this script. In user mode the script simply inserts the best alternative text for each image directly into the page, although users are provided the option to preface each inserted alternative text label with a string indicating that it was automatically generated.

## 5.2 Context-Centered Web Browsing

Mahmud *et al.* introduced a context-driven method called CSurf for browsing the web that they showed to be much more efficient than browsing with a traditional screen reader [25]. The increased efficiency of this method is derived from its ability to automatically direct users to relevant content, instead of requiring them to read a web page starting at the beginning, as is common in most screen readers. When using the system, users are directed to content related to links that they have followed. The text of a link is likely similar to the content in which they are interested. The system calculates where in the web page to begin reading by choosing the section of the web page that contains content most similar to the text of the link that was followed. This enhanced functionality is expected to be included in the Hearsay browser [31].

We have implemented a variation of this accessibility improvement as an Accessmonkey script. On every web page, the system first adds an onclick event to each anchor tag on the page. When a user clicks on a link, the text of the link is recorded. When a new page is loaded, the script checks to see if it occurred as a result of the user clicking a link. If so, it then finds the DOM element of the page that is most similar to the text of the clicked link using a simple word-vector comparison. The focus of the web page is changed to the identified DOM element, which allows modern screen readers to begin reading at that location. The system also assists web developers in setting skip links, which are links at the beginning of a web page that are visually hidden but provide a mechanism to screen reader users to skip to the main content area of a web page. This Accessmonkey script detects content on the web page that is likely to be relevant, highlights the identified area and adds the skip link if it is approved by the user. While this script cannot perform the full machine learning and semantic analysis that is done in CSurf, it allows this powerful technique to be used by users immediately with the tools they already own.

## 5.3 Personalized Edge Services

Iaccarino *et al.* outlined a number of edge services that transcoded web content into a form that better suited the personal needs of web users [20]. Accessmonkey provides an ideal framework in which to implement these edge services and we have replicated many of them as Accessmonkey scripts. The original intent of the edge services was to provide web users with disabilities options for personalization. By implementing them as Accessmonkey scripts, web developers can leverage them as well. Although many of these services are not appropriate for all users, web developers may employ them to produce alternative views for specific audiences.

We have replicated many of these edge services as Accessmonkey scripts, including services that replace all images in a page with links to the image, delete the target attribute from all anchor tags and add access keys to all links. Such improvements can make the web more accessible to certain individuals. These scripts can also be used by web developers, although, because of the nature of the transformations applied, they may be best used to help create alternative versions of a web page rather than used to create a universally accessible version.

## 5.4 Site-Specific Scripts

Many useful accessibility improvements cannot yet be implemented in a general way that will apply to multiple web sites. Such scripts can reorganize a site's layout, add accessibility features, or improve the accessibility dynamic content. We have implemented several scripts that demonstrate the dramatic improvements that can be made by Accessmonkey scripts targeted at specific sites. For example, the web page of a popular online retailer contains menubar at the top listing the major categories of products that they sell, organized in a tree. This menubar (and the elements contained within it) are inaccessible because they require the use of a mouse. We wrote an Accessmonkey script that allows the same content to be accessed via keyboard commands (see Figure 4). In this example, the menu content is available to screen reader users, but is not efficiently conveyed to them. Figure 5 demonstrates another example of a site-specific script that, in this case, removes distracting ads and places the main content of the page closest to the top in reading order.


**Figure 4: The menubar of this online retailer is inaccessible due to its reliance on the mouse. To fix this problem we wrote an Accessmonkey script that makes this menu accessible from the keyboard.**

The content that the scripts in this section modify already exists on the page, and, therefore, blind web users could potentially conduct these transformations independently. This is in contrast to the content in images or Flash content which is more difficult to access. While figuring out how to create a script that will improve accessibility may take time, the user will benefit from these improvements on subsequent visits to the page. These improvements could be leveraged by other web users visiting the page and, perhaps, even the web developers responsible for creating it. Javascript is a powerful mechanism for transcoding content and we are exploring how users can more easily discover and apply these scripts.

## 6. DISCUSSION & FUTURE WORK

Accessmonkey provides a common framework for web users, web developers, and web researchers to share automatic accessibility improvements. To facilitate this collaboration, we plan to create an online repository where such scripts can be posted and shared. We also plan to explore methods for enabling users to easily locate and, perhaps automatically, install scripts from this repository. For example, users could arrive at a news site to which they have not been before and be immediately greeted with the possibility of jumping directly to the content, navigation or search areas of the page.

Creating an Accessmonkey script currently requires a knowledge of Javascript programming. The Platypus Firefox extension allows users to create scripts for Greasemonkey by clicking and dragging elements with the mouse [33]. The extension contains keyboard support, but it still relies on visual feedback to alter content. Chickenfoot is a keyword-based system that allows users to naturally create simple scripts that might also be able to be extended to improve accessibility [24]. Other work has explored programming-by-demonstration methods for automating web tasks, such as Web Macros [32], PLOW [22], and Turquoise [27]. We are exploring methods targeted at providing similar functionality to web users for creating Accessmonkey scripts in a way that does not require using Javascript or a specific input device.

The transformations that current Accessmonkey scripts can achieve are currently limited by the Javascript programming language. While Javascript is more than adequate for achieving many transformations, more complex transformations often require the availability specialized libraries for natural language processing or image manipulation that are currently not available in Javascript and that would be difficult to implement in this arena. Accessmonkey scripts currently rely on web services for this advanced functionality, but a better solution may be for scripts to use supplementary libraries. We will explore both adding commonly-used functionality to Accessmonkey implementations and allowing user scripts to bundle Java code libraries. The implementations that we have provided already support calling Java code from Javascript and, so, a main challenge is to provide a standardized method for users to include such code along with their scripts and supporting such bundles in a variety of Accessmonkey implementations.

## 7. CONCLUSION

We have introduced Accessmonkey, a common scripting framework inspired by Greasemonkey that allows both web users and web developers to write and utilize Javascript scripts to automatically improve web accessibility. We have shown how implementations of our system can be created that run in a variety of platforms including as a Firefox extension and on a web page. We have converted our WebInSight system for automatically generating and inserting alternative text into web pages into an Accessmonkey script that allows both web users and web developers to use the technology built into WebInSight in order to improve the accessibility of web images. We have reimplemented several existing systems for automatically improving web pages, which renders these systems available on more platforms and allows them to more easily be utilized by web developers. We have also demonstrated that dynamic content can be made accessible on a per-site basis. Finally, we have shown that dramatic changes are possible by reconfiguring pages with Javascript and have discussed how users might easily create and use this functionality in the future.

## 8. ACKNOWLEDGMENTS

This research was funded by National Science Foundation grant IIS-0415273 and a software grant by GW Micro. We thank Oscar Danielsson, Gordon Hempton and Ryan


**Figure 5: This script moves the header and navigation menus of this site to the bottom of the page, providing users with a view of the web page that presents the main content window first in the page.**

Kaminsky for their contributions to the original WebInSight system and thank Sangyun Hahn and T.V. Raman for their insights and guidance. Finally, we would like to thank our anonymous reviewers for their insightful comments and suggestions.

## 9. REFERENCES

[1] A-Prompt. Adaptive Technology Resource Centre (ATRC) and the TRACE Center at the University of Wisconsin. http://www.aprompt.ca/.

[2] Greasemonkey Firefox extension. http://greasemonkey.mozdev.org/.

[3] Turnabout. Reify Software. http://www.reifysoft.com/turnabout.php.

[4] Watchfire Bobby. http://www.watchfire.com/products/webxm/bobby.aspx.

[5] Firefox accessibility extension, 2006. Illinois Center for Information Technology.

[6] GW Micro Window-Eyes, 2006. http://www.gwmicro.com/Window-Eyes/.

[7] JAWS 8.0 for windows. Freedom Scientific, 2006. http://www.freedomscientific.com.

[8] Lift. UsableNet, 2006. http://www.usablenet.com/.

[9] W3C markup validation service v0.7.4, 2006. http://validator.w3.org/.

[10] Web accessibility checker. University of Toronto Adaptive Technology Resource Centre (ATRC), 2006. http://checker.atrc.utoronto.ca/.

[11] Web content accessibility guidelines 2.0 (wcag 2.0). World Wide Web Consortium, 2006. http://www.w3.org/TR/WCAG20/.

[12] S. Bechhofer, C. Goble, L. Carr, S. Kampa, W. Hall, and D. De Roure. Cohse: Conceptual open hypermedia service. *Frontiers in Artifical Intelligence and Applications*, 96, 2003.

[13] J. P. Bigham. Increasing web accessibility by automatically judging alternative text quality. In *Proceedings of the 12th international conference on Intelligent user interfaces (IUI '07)*, New York, NY, USA, 2007. ACM Press.

[14] J. P. Bigham, R. S. Kaminsky, R. E. Ladner, O. M. Danielsson, and G. L. Hempton. Webinsight: Making web images accessible. In *Proceedings of 8th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '06)*, October 2006.

[15] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th User Interface Software and Technology*, 2005.

[16] D. R. Commission. The web: Access and inclusion for disabled people. The Stationary Office, 2004.

[17] S. Harper, C. Goble, R. Stevens, and Y. Yesilada. Middleware to expand context and preview in hypertext. In *Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility (ASSETS '04)*, pages 63–70, 2004.

[18] S. Harper and N. Patel. Gist summaries for visually impaired surfers. In *Proceedings of the 7th international ACM SIGACCESS conference on Computers and Accessibility (ASSETS '05)*, pages 90–97, New York, NY, USA, 2005. ACM Press.

[19] A. W. Huang and N. Sundaresan. A semantic transcoding system to adapt web services for users with disabilities. In *Proceedings of the fourth international ACM conference on Assistive technologies (Assets '00)*, pages 156–163, New York, NY, USA, 2000. ACM Press.

[20] G. Iaccarino, D. Malandrino, and V. Scarano. Personalizable edge services for web accessibility. In *Proceedings of the 2006 international cross-disciplinary workshop on Web accessibility (W4A '06)*, pages 23–32, New York, NY, USA, 2006. ACM Press.

[21] M. Y. Ivory. *Automated Web Site Evaluation Reseachers' and Practitioners' Perspectives*. Kluwer Academic Publishers, 2003.

[22] H. Jung, J. Allen, N. Chambers, L. Galescu, M. Swift, and W. Taysom. One-shot procedure learning from instruction and observation. In *Proceedings of the International FLAIRS Conference: Special Track on Natural Language and Knowledge Representation.*

[23] R. E. Ladner, M. Y. Ivory, R. Rao, S. Burgstahler, D. Comden, S. Hahn, M. Renzelmann, S. Krisnandi, M. Ramasamy, B. Slabosky, A. Martin, A. Lacenski, S. Olsen, and D. Groce. Automating tactile graphics translation. In *Proceedings of the Seventh International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '05)*, pages 50–57, New York, NY, 2005. ACM Press.

[24] G. Little and R. C. Miller. Translating keyword commands into executable code. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 135–144, New York, NY, USA, 2006. ACM Press.

[25] J. Mahmud, Y. Borordin, and I. Ramakrishnan. Csurf: A context-driven non-visual web-browser. In *Proceedings of the International Conference on the World Wide Web (WWW '07).*

[26] J. Mankoff, H. Fait, and T. Tran. Is your web page accessible?: a comparative study of methods for assessing web page accessibility for the blind. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '05)*, pages 41–50, New York, NY, USA, 2005. ACM Press.

[27] R. C. Miller and B. Myers. Creating dynamic world wide web pages by demonstration, 1997.

[28] H. Petrie, F. Hamilton, and N. King. Tension, what tension?: Website accessibility and visual design. In *Proceedings of the international cross-disciplinary workshop on Web accessibility (W4A '04)*, pages 13–18, New York, NY, USA, 2004. ACM Press.

[29] M. Pilgrim, editor. *Greasemonkey Hacks: Tips & Tools for Remixing the Web with Firefox*. O'Reilly Media, 2005.

[30] P. Plessers, S. Casteleyn, Y. Yesilada, O. D. Troyer, R. Stevens, S. Harper, and C. Goble. Accessibility: a web engineering approach. In *Proceedings of the 14th international conference on World Wide Web (WWW '05)*, pages 353–362, New York, NY, USA, 2005. ACM Press.

[31] I. Ramakrishnan, A. Stent, and G. Yang. Hearsay: Enabling audio browsing on hypertext content. In *Proceedings of the 13th International Conference on the World Wide Web (WWW '04)*, 2004.

[32] A. Safonov. Web macros by example: users managing the www of applications. In *CHI '99 extended abstracts on Human factors in computing systems (CHI '99)*, pages 71–72, New York, NY, USA, 1999. ACM Press.

[33] S. R. Turner. Playtpus firefox extension, 2006. http://platypus.mozdev.org/.

[34] M. Vorburger. Altifier: Web accessibility enhancement tool, 1999.

[35] H. Wang. Nextplease!, 2006. http://nextplease.mozdev.org/.