Improving Web accessibility through an enhanced open-source browser

V. L. Hanson
J. P. Brezin
S. Crayne
S. Keates
R. Kjeldsen
J. T. Richards
C. Swart
S. Trewin

The accessibilityWorks project provides software enhancements to the Mozilla™ Web browser and allows users to control their browsing environment. Although Web accessibility standards specify markup that must be incorporated for Web pages to be accessible, these standards do not ensure a good experience for all Web users. This paper discusses user controls that facilitate a number of adaptations that can greatly increase the usability of Web pages for a diverse population of users. In addition to transformations that change page presentation, innovations are discussed that enable mouse and keyboard input correction as well as vision-based control for users unable to use their hands for computer input.

INTRODUCTION

Accessibility technology is of value not only for people who have disabilities, but also for a significant number of workers, many of whom would not consider themselves as disabled or as having a medical condition, but who nevertheless experience difficulty reading information on a computer screen or accurately using a keyboard or mouse. Such technology has the potential to enable people with a broad spectrum of abilities, who may currently be unemployed or underemployed, to fully participate in the workforce.

The World Wide Web has become an indispensable source of information and communication both inside and outside the workplace. For the past few years, work at the IBM Thomas J. Watson Research Center has been directed at creating ways to make the Web more usable for persons with vision and motor limitations. This effort began as a partnership between IBM Corporate Community Relations and organizations serving older adults. Since then, the project has expanded to address the needs of other groups of individuals who have vision and motor limitations that adversely impact their ability to use a computer. Unforeseen when the project began was the software's additional use by low-literacy and limited-English-proficiency students in job-training classes.

This paper begins with background information on the types of problems that may be experienced by Web users, and then describes the Web Adaptation

Technology project, the forerunner of the accessibilityWorks project. The Web Adaptation Technology project provided extensions to the Internet Explorer** browser on the Microsoft Windows** platform. This paper continues with a presentation of accessibilityWorks, a new open-source development effort initially being tested on the Linux** platform, which is an outgrowth of that earlier project. The accessibilityWorks project specifically took the findings from the Web Adaptation Technology work and used them in implementing similar capabilities in the Firefox** and Mozilla** open-source browsers to provide cross-platform functionality. As will be discussed, accessibilityWorks also provides expanded mouse and keyboard adaptations.

■ Our early pilot studies with users revealed that people prefer to use a standard browser with accessibility adaptations added ■

WEB USABILITY

Problems with computer access due to vision and motor limitations can be addressed by various medical, hardware, and software approaches. Large monitors and eyeglasses, for example, can certainly help with visual acuity problems, but these measures may not be sufficient. Consider the case of bifocals worn by many older adults. These glasses can provide some help with acuity difficulties, but wearers of bifocals who spend much time at the computer tend to develop stiff necks from tilting their heads at an awkward angle. This is exacerbated for people with particularly low vision—caused, for example by macular degeneration—because they need to sit very close to computer screens and often develop shoulder and back pain as a result.

Current browsers and desktop environments can adjust some aspects of content presentation to alleviate certain difficulties when reading text. However, although users are often aware that some font enlargement capability is available through the browser, few are aware of browser options for changing colors, font styles, and screen magnification, or for creating style sheets.[1] This lack of awareness is not particularly surprising, given that the options often are configurable only by following a complex series of menu choices and dialog boxes. For example, changing the color of text on a Web page by using Internet Explorer's built-in features requires knowing that `Internet Options` is the required menu selection from the `Tools` menu, then further being able to navigate to the `Colors` dialog box, deselect the `Use Windows colors` checkbox, set the desired colors, and then return to the `Internet Options` menu selection to indicate on the `Accessibility` dialog box that specified Web page colors should be ignored. This is a fairly complex task requiring knowledge of computer software, the ability to see the small boxes to be selected, and the dexterity to click accurately.

Some motor difficulties can be addressed by using the adjustments to mouse and keyboard sensitivity built into current computer operating systems. As with page presentation changes, the use of such adjustments requires that users be aware of the existence of these options, know where they are located in the menu structure, and be able to carry out the steps needed to set the features. Users who have disabilities that impact their hand movement can have great difficulty actually using these tools.[2] Consider the case of key *debounce* time. Persons with tremors, for example, may depress one key multiple times in rapid succession, causing repeated letters to appear when typing. Microsoft Windows allows users to set a debounce parameter to control the length of the time period in which repeat keys will be filtered out. In order to set this parameter, however, users must not only know that this feature exists and how to find it in the `Control Panel` options, but must also be able to navigate the steps required for setting the feature, which in turn requires the clicking of tiny graphical user interface (GUI) buttons and checkboxes.

There are special keyboards and mice available to ease difficulties with double-clicking and scrolling, but, in fact, these special devices provide only partial solutions to the problems experienced by older adults. Accurate mouse usage, for example, requires visual acuity; people with poor vision have mouse difficulties whether they possess good motor control or not.[3] Scrolling is another example of a task that is particularly difficult for reasons related to a combination of visual and motor factors. Visually, the small size of the scroll bar, particularly the target box, can be problematic. In terms of motor

skills, scrolling requires the complex sequence of moving the mouse to the small target box, holding down the mouse button, and then continuing to hold down the button while moving the mouse in the direction needed for scrolling. Solutions that address difficulties in only one of these aspects of scrolling may not succeed.

Considering all the preceding difficulties, we began our project to address the need to allow individuals to tailor Web interactions to their particular combination of requirements. The goals as the project began focused on meeting changing user needs resulting from failing vision and limited dexterity due to aging. The project did not attempt to address either limited hearing, which was not considered by users to negatively impact their Web experience,[4] or blindness, for which other Web accessibility software exists.[5] As the project evolved, however, it became clear that the needs of the original older-adult user population were not unique. In particular, problems with content size, color, and other display characteristics were typical of many users with vision limitations, regardless of age and the specific medical condition that may have led to the limitation. Similarly, motor disabilities, regardless of origin, created keyboard and mouse problems similar to those experienced by older adults. As a result, at present the IBM Corporate Community Relations project has expanded to include approximately equal numbers of members of organizations serving older adults and organizations serving persons with disabilities.[6]

Web adaptation technology: Internet Explorer browser extensions

The project's software architecture was based on the need for client-side transformations combined with a server-side database for storing user preferences. As detailed elsewhere,[7,8] an original attempt to use a server intermediary for providing the transformations was inadequate with respect to security, copyright compliance, system responsiveness, and transformation accuracy. These shortcomings led to this initial approach being abandoned. A database on the server was still needed, however, for storing user preferences because most people used the software in a shared computing environment. Thus, saving preferences on the local client machines was not a viable option.

The software needed to be used by people for access to the entire Web. Thus, it could not be limited to sites specifically designed or annotated for accessibility. Key to the software design was the fact that many users experience a variety of physical and cognitive limitations that may impact their Web access. These limitations are made more complex by the fact that they can occur in combination and can fluctuate in severity from hour to hour and day to day.[9] Thus, the content transformations needed to work well in combination with each other. Moreover, the sorts of disabilities a particular user will have cannot easily be known in advance. Therefore, a large collection of possible transformations was needed. The transformations implemented were suggested both by interviews with users and by the literature.[10–14] These transformations included options for:

- *Enlarging page content*—for example, magnifying pages and enlarging specific text or images
- *Enhancing text*—for example, changing colors, letter and line spacing, and text style
- *Reducing visual clutter*—for example, stopping animations, hiding backgrounds, and reformatting pages for a single-column layout
- *Enlarging browser controls*—for example, enlarging the cursor and scrollbar
- *Adapting keyboard and mouse settings*

Moreover, the interface for selecting these transformations needed to be accessible by people with a wide range of disabilities.

In addition, our early pilot studies with users revealed that people prefer to use a standard browser with accessibility adaptations added, rather than a specialized browser that has specific buttons for low vision but that offers a more limited set of features overall. Users who had previous Web experience did not like a specialized browser due to lack of the functionality that they were accustomed to using. Instructors who taught Web skills to new users did not like the fact that a specialized browser did not mesh well with existing teaching materials. Based on this feedback, the Web Adaptation Technology software was built as extensions to the Internet Explorer browser.

Finally, because many users who were expected to use the control interface were new to computing, it had to be very simple and straightforward to

operate. The resulting interface was a band appearing at the bottom of the browser that could be hidden after preferences were set. The band comprised a series of panels, one panel for each of the different options that could be set.

Mechanisms for providing added functionality

Functionality was added to the browser through a variety of means, including the automatic setting of browser and operating-system features (including the use of programmatically constructed style sheets), manipulations of the browser's underlying Document Object Model (DOM), and the launching of background tasks to monitor events and provide transformation services. Each of these mechanisms will be discussed in turn.

Browser and operating system features

The Web Adaptation Technology software provides convenient and uniform access to browser and operating-system settings for non-expert computer users or persons who might have difficulty with the physical or cognitive complexity of changing these settings. Examples of Web page transformations using the underlying browser and operating-system features include text style changes, text size increases (up to the limits of what the browser directly supports), and the hiding of images, backgrounds, and animations. Programmatically generated style sheets are used to implement page magnification and increased letter and line spacing. Input adaptations, including allowing use of only one hand as well as keystroke, mouse click, and typing adjustments, all use operating-system features.

Document Object Model

Certain dynamic visual changes to a Web page are accomplished in the Web Adaptation Technology software by means of manipulation of the browser's internal model for the page, the DOM. The DOM is structured as a tree in which all the elements of the page are represented, one element being a child of another within which it is contained. The Internet Explorer implementation uses a *Browser Helper Object* written in Java**, which gives the program access to the DOM after it is constructed (so that any JavaScript** or other dynamic changes to the document have already been made) but before it is displayed by the browser. The software is thereby able to manipulate the representation that will determine exactly what the user will see, before the user sees it. As a result the software does not need to be concerned with how the source HTML (Hypertext Markup Language) is parsed or dynamically generated. It simply deals with the content after the DOM is constructed. Examples of DOM-based visual transformations include changing colors and changing page layout (linearization). An important non-visual transformation involves *chunking* large text blocks into smaller ones in order to get a better idea of where the mouse is pointing. This chunking forms the basis of the *Speak Text* feature for reading text aloud and the *Banner Text* feature for displaying selected text in very large letters at the top of the browser window.

Background Tasks

The remaining functions in the Web Adaptation Technology software are accomplished through background tasks, which are launched as the software is started. Certain keyboard adjustments are handled by a background task, the Dynamic Keyboard,[15] that monitors keystroke events, compares this event stream to a typing model, and then makes adjustments to keyboard features in the operating system. (These features are described in more detail later in this paper.) Image enlargement of GIF (Graphics Interchange Format) and JPEG (Joint Photographic Experts Group) images and sharpening of JPEG images also involve a background task. This task retrieves the images, decompresses them, and enlarges them using bilinear interpolation to eliminate the jagged appearance that results from a simple scaling process. If requested, the images are also sharpened by enhancing the contrast of edges within the image. The images are then compressed and displayed.

Status of the Web Adaptation Technology project

The Web Adaptation Technology software has been in use for over a year by a number of nonprofit organizations sponsored by IBM Corporate Community Relations. The software has been translated into several languages including Spanish, French, German, Italian, Brazilian Portuguese, Chinese, Korean, and Japanese, and it is used not only in the United States but in many other countries worldwide. Recently, it has been developed into a generally available IBM services offering named WebAdapt2Me.[16]

User feedback was gathered during testing with the Web Adaptation Technology to learn more about

what users find beneficial and to determine what further enhancements to the functionality might prove helpful. For example, user comments helped us refine the Speak Text feature, extend the range of transformations within some existing features (e.g., adding an option for increasing text size beyond what the browser could support directly), and enhance feature usage through keyboard navigation.

Importantly, what started as an application for older adults is now being used by many people who come from communities other than the aging population. That this software might serve the needs of users with poor vision, regardless of age, is not surprising. That it can serve the needs of users with keyboarding difficulties, regardless of age, might also have been expected. What is surprising, however, is the extent to which this technology has been adopted by other user groups, including low-literacy and limited-English-proficiency students, and young adults with autism, developmental delays, or attention issues. The Speak Text feature is particularly popular among low-literacy users, and several of the text presentation features have also been adopted for initially unforeseen purposes. For example, the letter and line spacing feature was designed to reduce visual clutter and provide older readers with more white space between text characters. However, reducing visual clutter also proved useful for students with developmental delays or attention problems. Surprisingly, these students often used the page magnification function to reduce the amount of information presented on the screen. More detailed information about these various uses of the Web Adaptation Technology software is reported elsewhere.[6,7]

Firefox and Mozilla browser extensions with accessibilityWorks

Although the Internet Explorer implementation allowed for rapid prototyping because it was possible to use many Windows native functions, the use of these native functions obviously restricted the tool specifically to the Windows operating system. One of the original project goals, however, was to create a cross-platform solution.[17] For the follow-on accessibilityWorks project, therefore, development has focused on the Firefox and Mozilla browsers, which do provide a multiplatform solution.[18,19] In this work, Web page transformations are made within the existing Mozilla programming model in order to minimize platform dependencies. The growing popularity of Linux has also prompted us to further extend some of our Windows-based applications, for example, the Dynamic Keyboard and camera-based user interfaces (discussed later), to run on the Linux platform.

■ What started as an application for older adults is now being used by many people who come from communities other than the aging population ■

For accessibilityWorks, the goal was not simply to recreate features available in the Web Adaptation Technology software, but to expand upon those features. In particular, additional capabilities for mouse correction and vision-based interfaces are being incorporated into the project.

In addition, accessibilityWorks is intended to augment existing accessibility development work for the Linux platform. The growing popularity of Linux has created a developer community that has worked to create accessibility tools for that platform.[20–22] Although some applications for Linux, such as Mozilla, can run on many platforms, others are native to the Linux platform and are often built into the desktop, such as the accessibility applications included in GNOME**.[22] Because the accessibilityWorks input enhancements are closely tied to the operating system, some Linux-specific development was required to implement these features, as will be discussed later in this paper.

Figure 1 shows the architecture for the accessibilityWorks implementation for Firefox and Mozilla. The elements of this design will be discussed in the remainder of this paper.

Transformations to page presentation

Mozilla provides a self-contained application-development and deployment environment. Applications written for this environment, namely browser extensions, run on all the platforms on which Mozilla runs. This means that most of the

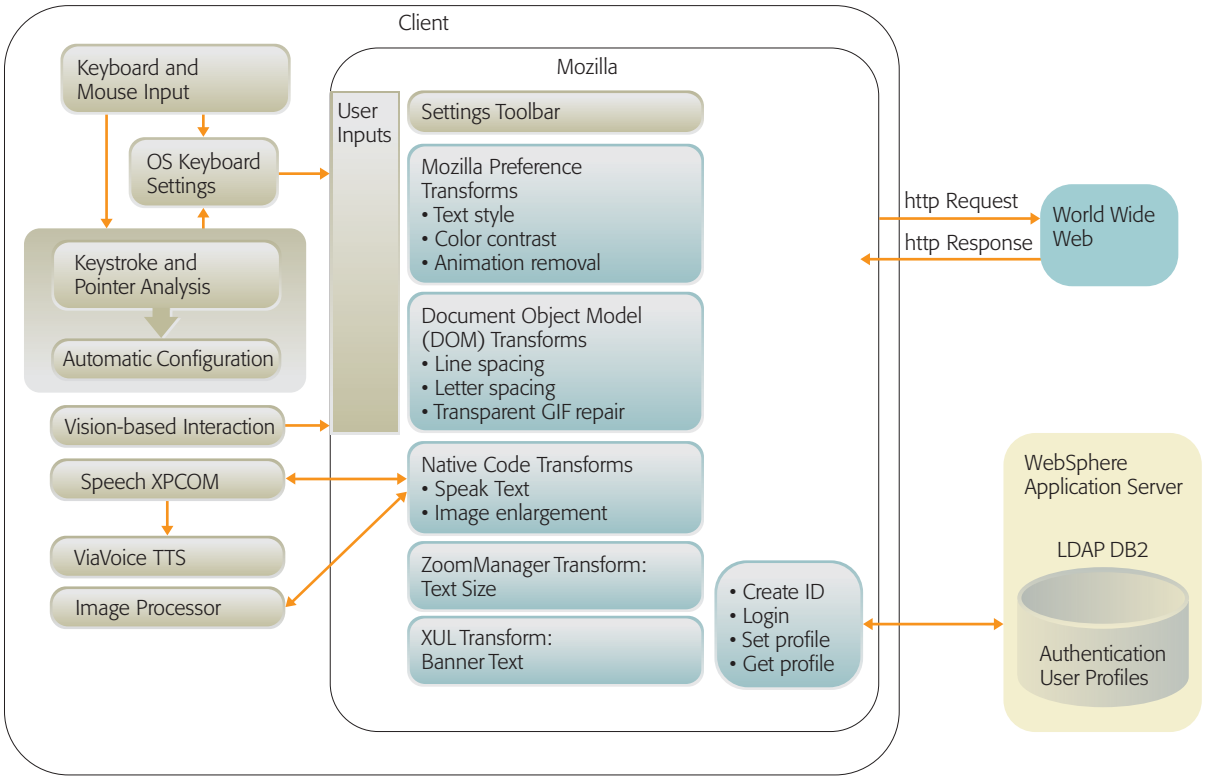

Figure 1
Design elements of the accessibilityWorks Mozilla implementation

code only needs to be written once. In particular, the XML (Extensible Markup Language) User Interface Language (XUL**) feature of this environment permits the runtime assembly of complex user interface elements.[23] From XUL, one can call JavaScript functions to accomplish other tasks, such as initialization, event handling, and DOM modification. The overlay feature of XUL also allows the characteristics of the parts of the browser that frame the browser document window, such as menus and toolbars, to be changed. In Mozilla terminology, these elements are called the *chrome*.

XPCOM is a feature of the Mozilla environment that allows developers to perform lower-level tasks that cannot be done in XUL or JavaScript.[24] (XPCOM stands for Cross-Platform Component Object Model, and is somewhat similar to Microsoft's COM or Component Object Model.) Mozilla comes supplied with a large variety of XPCOM objects that can be instantiated from JavaScript. These objects currently allow preferences to be retrieved and set, and files to be read and written. Developers can also write their own XPCOM objects.

In many ways, Mozilla's application-development environment allowed simplification of the Web Adaptation Technology architecture created for the Internet Explorer implementation. For example, we had to manage a series of dialogs to represent each of the panels in the band. In the current version, Mozilla's XUL language does much of this work for us. XUL provides a user-interface element called a *deck*. Each panel is a member of the deck and has an *index* associated with it. Panels can be brought to the top of the deck (that is, made viewable) simply by changing the current index of the deck.

The panels are designed to be easily used and to allow each user to try out transformations to determine their suitability and effectiveness. ***Figure 2*** shows an example Web page (2A) and then versions of that page with various control-panel features applied. Thus, ***Figure 2B*** shows the Speak Text feature, in which content is read aloud. ***Figures 2C*** and ***2D*** then show the results of the application of specific page transformations to the original page. More specifically, in Figure 2C, Banner Text is shown. Figure 2D shows the page with text, back-

Figure 2
Transforming a Web page for accessibility: (A) the Web page with no transformations applied; (B) application of the Speak Text transformation from accessibilityWorks; (C) the original Web page with the Larger Text and Banner Text transformations applied; and (D) the original Web page with colors changed.

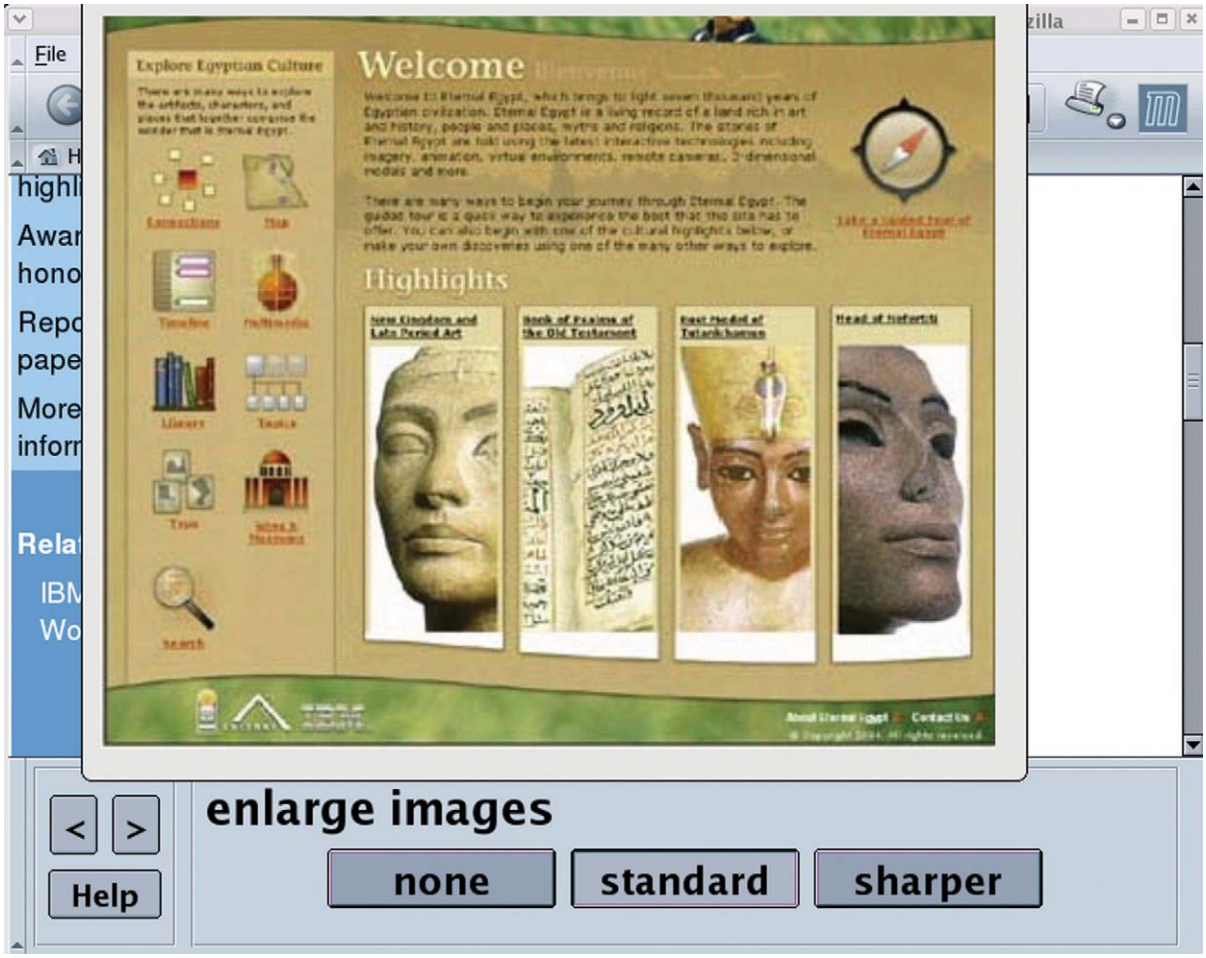

Figure 3
The original Web page from Figure 2 with an image enlarged. An image is enlarged when a user selects this option and then points the mouse at an image.

ground, and link colors all changed. ***Figure 3*** uses the same page to show an example of image enlargement.

Firefox allows users to change Web page color preferences by accessing the `Options` dialog box on the `Tools` menu. There are fewer steps in the Firefox `Fonts and Colors` option than in the comparable Internet Explorer sequence, but modifications do still require that users know that colors can be set, know about terms such as "unvisited link," and have both the vision and dexterity required to set these options by using small checkboxes. As shown in Figure 2D, the `colors` buttons provided by the accessibilityWorks interface greatly simplifies this task.

One useful feature of the Mozilla application environment is that the DOM is extended to the chrome. A tool called the *DOM Inspector* is included with Mozilla. By using this tool one can start at the DOM node whose ID is `main-window`. (This is the main window of the browser's chrome, not the window of the current Web page.) One can then traverse the tree all the way down to the node whose ID is `browser`, which corresponds to the Web page document, and beyond. Along the way, changes can be made to either the chrome or the document. For accessibilityWorks, the chrome was modified to create the settings band that was originally implemented in the Internet Explorer version as a separate dynamic link library (DLL). This technique was also used to increase the font size of the menus and to place a custom icon on the toolbar.

The Speak Text functionality in accessibilityWorks is implemented as an XPCOM object, with calls to functions in the IBM ViaVoice* Text-To-Speech (TTS) library.[25] In Windows, an XPCOM object is a DLL, whereas in Linux, it is a shared library. The TTS XPCOM functions are directly callable from JavaScript, and in turn make calls to the IBM speech API (application programming interface), which is embodied as `ibmeci.dll` on the Windows platform, and as `libibmeci.so` on the Linux platform. The TTS XPCOM object is completely platform independent and thus only has to be recompiled to work on each platform.

The programming of DOM transformations is similar in Mozilla to the equivalent code in the Web Adaptation Technology software. Working in JavaScript though has meant that this implementation is significantly more compact than the Internet Explorer implementation in Java. This advantage is a direct result of Mozilla's XUL-based architecture, which allows the use of scripts that gain control when the page is loaded, much as Internet Explorer's Browser Helper Objects do. The key point is that the scripts do not have to be inserted into the pages themselves before the pages are parsed; the scripts are part of the browser's definition and are run on an event-driven basis. In addition, the user-interface event handling provided by Mozilla allows significantly simpler programming because fewer handlers are needed and they are also less complex.

Initial testing of accessibilityWorks was carried out on Mozilla, and accessibilityWorks currently runs on Mozilla Versions 1.6 and 1.7.3. With the release of Firefox 1.0 the functionality is now being moved to Firefox, which will soon be supported as well.

Keyboard and mouse assistance

The transformations of page presentation discussed so far have aimed at improving overall usability for users who have difficulties reading the content of a Web page. However, reading content is only one aspect of a successful Web experience. Typing words for a Web search, typing e-mail, and completing online forms all require the ability to use a keyboard. In addition, Web navigation is critically dependent on the ability to use a mouse to point and click. These features of a Web interface can create severe difficulties for persons who have a motor impairment.[26] For example, in the case of older adults it is possible, even likely, that users may have arthritis in their hands, making it difficult to produce the fine motor movements required to touch-type or to operate a mouse button. Moreover, conditions such as Parkinson's disease and cerebral palsy are strongly associated with involuntary tremors and uncontrolled movements. Even something as common as a bone fracture or repetitive strain injury can lead to difficulties typing or using a mouse. According to recent government figures, approximately 10 percent of working-age, non-institutionalized adults have significant long-term motor limitations.[27,28] A recent commercial study suggested that more than a quarter of working-age adults experience dexterity limitations and would benefit from using accessible computer technology.[29] To address these issues, the accessibilityWorks project includes options for aiding both text (keyboard) and pointer-based (mouse) input. These common user-input problems are discussed in the next section.

■ Reducing visual clutter also proved useful for students with developmental delays or attention problems ■

Common user-input difficulties

Common keyboard input difficulties include pressing and holding a key too long, difficulty holding down one key while pressing another, pressing two keys at the same time, pressing a key multiple times when only a single press was meant, and pressing the wrong key.[30] Many operating systems, including Windows, UNIX**, and Linux, offer mechanisms for adjusting key repeat rates and debounce thresholds that address certain of these difficulties. These mechanisms are specifically built into the Windows operating system. On UNIX and Linux systems they are provided by the accessX feature of the X Window System**. Each platform offers essentially the same keyboard access features, but implements them in different ways. For example, Windows systems maintain two key-repeat delay values, one of which is associated with the Windows accessibility features and overrides the other value when accessibility features are activated. In contrast, accessX uses a single key-repeat delay value. The accessX features tend to offer greater sensitivity and a greater range of available values than those provided in

Windows. For example, Windows XP offers four key-repeat delay values in the keyboard control panel and five different values in the accessibility control panel. Together these enable delays of 250, 300, 500, 700, 750, 1000, 1500 and 2000 msec. Repeats can also be turned off completely. AccessX provides values in the range 100–1510 msec, in increments of 10 msec, plus the ability to turn off repeats.

■ In many ways, Mozilla's application-development environment allowed simplification of the Web Adaptation Technology architecture created for the Internet Explorer implementation ■

However, comparatively few users know about the existence of these options, and even fewer know how to adjust them, much less what the optimal settings are for their own needs. To further compound the situation, a user's optimal settings may vary over the course of the day, from early-morning alertness to post-lunch fatigue. Often users have to adjust the settings by trial and error, and this can lead to a strategy known as *satisficing*, that is, finding a solution that is just about satisfactory, but far from ideal.

Mechanisms for correcting common user-input difficulties

In the Web Adaptation Technology software, several keyboard and pointing options were provided. Some of these were explicitly provided as settings under direct user control. An example is the *One Hand* feature that eliminates the need to hold keys down while pressing other keys. Setting the One Hand option turns on the `StickyKeys` functionality in the operating system. In other cases, such a simple setting of keyboard adjustments was not possible. Specifically, for those options where a range of settings are available and where users typically do not know their own ideal settings, automatic adjustment was implemented through the Dynamic Keyboard application.[15] Consider, for example, the case of debounce time that was mentioned earlier. Certain motor disabilities cause users to depress one key multiple times in rapid succession, creating repeated letters when typing. The debounce setting available in many operating systems can filter out such repeated keystrokes by controlling the length of time after a given keystroke before another typed letter will be accepted. In order to use the debounce setting, though, users must know about this feature, be able to find and set it through menus and dialog boxes, and be able to specify the required debounce time in milliseconds. The Dynamic Keyboard simplifies this process by automatically detecting the need for a debounce adjustment and then setting debounce parameters appropriate for the user. Most important, this software monitors typing behavior throughout a user's session. Thus, as typing needs change, adjustments to typing features are continuously updated.

At present the Dynamic Keyboard functionality adjusts key repeat delays, repeat times, and bounce thresholds. It has no user interface. The goal is to perform adaptations that improve the user's input accuracy when problems are observed, but otherwise remain unobtrusive. This is achieved by monitoring specific aspects of a user's keystrokes and adjusting the appropriate keyboard accessibility parameters accordingly. User studies have shown these interventions to be remarkably effective among users with noticeable keyboard difficulties, and they have also proved quite popular with these users.[15]

Separate Dynamic Keyboard implementations have been developed for Windows (used in the Web Adaptation Technology software) and Linux systems with accessX (used in the accessibilityWorks software). These have essentially identical functionality, but each uses the native operating-system features to implement the adaptations. This means that the sensitivity of the response depends on the sensitivity of the host system and necessarily reflects any constraints imposed by the native operating system. An example of such a constraint and its effect on the Dynamic Keyboard's effectiveness is given in the following paragraph.

Specific Dynamic Keyboard experiments have been performed with adjustments to the key repeat delay, key repeat rate, and bounce thresholds.[15] These user studies have suggested that the key-repeat delay adjustments made by the Dynamic Keyboard are

appropriate, effective, and acceptable to users. This adaptation thus represents an initial solution to the single greatest keyboard accessibility barrier. The current algorithm for automatic adjustment of key repeat rates requires some improvement, however, to account for the many compensatory strategies people use. For example, bounce thresholds as implemented in Windows have proved inappropriate for automatic adjustment, and in general for users with low error rates. This is partly because the shortest available debounce setting in Windows is 500 msec, which is too long for most users, forcing them to deliberately slow down their typing of double letters. The Windows version of the Dynamic Keyboard is therefore unable to respond appropriately when a (more typical) debounce time of 50 msec is indicated. Activating the debounce feature in Windows also prevents keys from repeating, and many users find this side effect unacceptable. On Linux systems, a more appropriate range of debounce settings is available (0 to 910 msec in increments of 10 msec), and there is no key-repeat side effect. The debounce feature may therefore prove more acceptable to Linux users.

New keyboard assistance, such as the automatic filtering of multiple key presses, will also be incorporated in accessibilityWorks. Users with poor hand-eye coordination can find it difficult to position their fingers precisely over the keyboard. As a result it is quite common for them to press both the intended key and an adjacent key at the same time. Research has shown that careful analysis of the number of times each key was pressed and released can indicate which key was intended with 80 percent accuracy.[31] This functionality has been referred to as `OverlapKeys`.

The success of the Dynamic Keyboard as an approach to solving input difficulties suggests the value of applying similar principles to other input devices, such as the mouse. For example, should the user accidentally click both the left and right buttons at the same time, filtering like that used for `OverlapKeys` can be applied. Research is currently being performed to establish other areas of possible assistance, and the resulting utilities will be incorporated into later versions of accessibilityWorks.

Camera-based user interfaces

In some user cases, even the types of adjustments for keyboard and mouse correction discussed earlier are not sufficient. Consider, for example, the case of individuals with spinal cord injuries who are not able use their hands well enough to control either a keyboard or a mouse. Custom hardware can be built for users with such disabilities, but this process is expensive and time-consuming. Moreover, the hardware can also become unsuitable if the needs of the user change. Camera-based user interaction has the flexibility to provide interfaces that are far more universally accessible than such hardware-based interactions.

Camera-based interfaces rely on visual recognition of the user's actions. Specific body movements can take the place of buttons and pointing devices. Unlike hardware devices, elements of a camera-based interface can be rearranged easily to satisfy different requirements or reconfigured to adapt to different movement patterns. The result is an interface that can be customized to match a user's current abilities.

Over the last several years IBM Research has developed several systems targeted for users with physical disabilities that take advantage of camera-based interactions. Two of these have proven useful and reliable and are being incorporated into accessibilityWorks.

The TouchFree** Switch, released by Edmark in 1999, represents the simplest variant of a vision-based interface. As shown in ***Figure 4***, the TouchFree Switch provides an alternative to the physical switches often used by people with severe physical disabilities to control so-called scanning interfaces. It allows users to locate interactive regions in video images of themselves and to train the system to recognize specific changes in appearance in those regions. Thereafter, when a region detects such a change, it generates an event that can be used by the scanning software to allow control of a full range of computing activities. In this way users can interact with the computer by a tip of the head, a shrug of the shoulder, finger movement, or nearly any other body movement that produces a change visible to a video camera. These "buttons" can be repositioned or trained to recognize a different movement in a few seconds, allowing an interface that can adapt easily to meet a user's immediate needs.

In other applications, tools were developed to control the mouse pointer. In one version, the

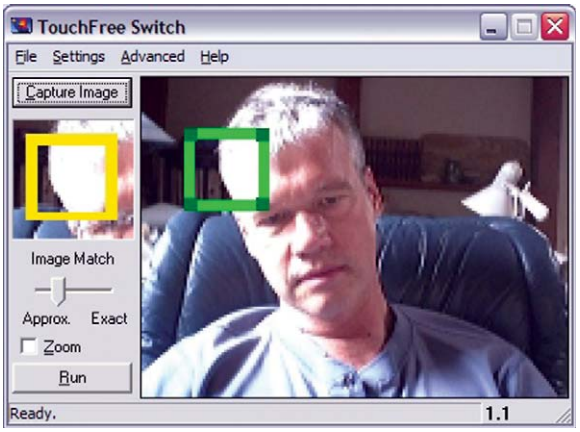

Figure 4
The TouchFree Switch allows users to control input by moving any part of one's body. In this picture, the user is controlling input by tilting his head

camera was placed near the computer screen looking back at the user. By monitoring the movement of users' faces, the system was able to position the pointer on the screen with character-level accuracy, such that users felt as though they were actually pointing with their noses.[32]

The accessibilityWorks project will incorporate these capabilities as alternative input methods for pointing and clicking. Using a camera connected by USB (Universal Serial Bus), users will be able to position the pointer by aiming their faces, and then "click" either by allowing the pointer to dwell at a location on the screen or by moving some other body part.

Vision-based input requires a small amount of setup on the part of users or their facilitators. Because face-finding algorithms are not sufficiently reliable under all the imaging conditions where accessibilityWorks may be used, users must tip their heads left-and-right three times at the beginning of each session (or whenever face-tracking performance degrades). This distinctive motion is easy to recognize and provides the system with sufficient information to begin tracking the user's face. If a separate movement is to be used to generate the click, then an interactive region, similar to those used in the TouchFree Switch, can be positioned in the video image such that users can "touch" that region of the image with some part of the body. If the system is being used in an environment where other movements may appear in the interactive region, users can demonstrate a specific movement to the system so that only that touch elicits a click. For example, if users want to use a shoulder shrug to click, they may select a region just above the shoulder such that the shoulder moves into it when they shrug. If they are concerned about people triggering the button by walking behind them, they can train the system to be more specific by demonstrating the shrug.

These tools provide basic mouse function for users with certain physical disabilities. Our goal is to take better advantage of the flexibility of camera-based interaction to provide a richer set of interactions for users with a wider range of limitations. Our current vision system can recognize many different input signals in user body movements, but the way this capability is provided to the user is limited. We can provide more flexibility by classifying these control signals according to the dimensionality of the information they contain (i.e., zero-, one-, two-, or three-dimensional information). If the inputs required by an application are also so classified, then user signals can be mapped to application inputs of the same type, allowing users to control an application with whatever body movement they would like. For example, users could control pointer movement using facial aiming for a time. Should they tire, they could switch to moving the hand within a region on a surface.

Ideally, to take full advantage of vision-based interaction the design of the browsing interface should be modified. A good example involves link selection. Typically, links are selected by two-dimensional positioning of the pointer followed by a click, but for some users accurate two-dimensional positioning can be difficult. To create an interface for such a user, the links on a page can be extracted automatically and placed in an auxiliary window in linear order. One of several one-dimensional positioning actions, such as an up/down tip of the head or a lateral motion of a hand, can then be used to select links.

Camera-based user interfaces require significant software infrastructure. Image interpretation must be fast and reliable, image buffer management and display routines must be efficient and flexible, and methods must be in place to assemble these building blocks into useful, user-friendly components. A comprehensive library of low-level image-process-

ing routines and common high-level operations for finding and tracking body parts is important. The code must be efficient enough to handle at least 15 frames a second in order to support good quality user response.

Over the last several years just such an infrastructure has been developed under Windows.[33] This VI (Visual Interface) Architecture is now being ported to Linux in such a way as to have a common code base for both platforms. The bulk of the code is already system-independent C++. The GUI, which serves as the control panel for applications, is being moved to a cross-platform development toolkit. Generic image display routines are often not fast enough to support video rates, and camera interfaces vary significantly among systems. These elements are optimized for each platform, then encapsulated in VI objects, so that any camera-based interface developed with the VI Architecture can run on either platform without code changes.

CONCLUSION

Over the past few years, work at the IBM Thomas J. Watson Research Center has been directed at developing tools to support computer use by persons who, for various reasons, have difficulty using such technology. The effort to improve Web access discussed here is an example of one such project. The hallmark of this work has been ongoing user involvement and iterative testing throughout the design and development process. Partnerships between IBM Corporate Community Relations and nonprofit organizations provided the perfect environment for this work. Through such partnerships with organizations serving both older adults and persons with disabilities, valuable feedback was received throughout the development process to enable the design of a system that would meet user needs.

The Web Adaptation Technology application, a platform-dependent set of extensions for Windows Internet Explorer, was developed first. Its successor, accessibilityWorks, provides a cross-platform implementation using the Firefox/Mozilla browser. This new software builds on the lessons learned from the initial work and expands its capabilities. It is not a port of the earlier software, but rather a new implementation with additional functionality. In particular, it expands on the earlier work in the area of input technologies through the addition of keyboard and mouse assistance as well as vision-based user interfaces. The accessibilityWorks software is

■ According to recent government figures, approximately 10 percent of working-age, noninstitutionalized adults have significant long-term motor limitations ■

in early test stages on the Linux platform by persons who have low vision, learning disabilities, spinal cord injuries, and cerebral palsy.

The accessibilityWorks browser approach, which allows users to control presentation and input, is complementary to that involving the issue of Web page compliance with accessibility standards and regulations,[34–36] for which excellent tools have been developed to help content providers.[37] In many cases these guidelines do not directly impact page presentation, but rather address issues related to making Web pages capable of being rendered by assistive technology devices such as screen readers.

The work described here takes a broader view of Web accessibility. Rather than addressing Web page compliance, this software provides for changes that directly impact the user experience. The types of page transformations and input adaptations discussed do not modify page sources. They simply modify presentation and input on a computer as requested by the user.

It is important for Web page authors to still adhere to accessibility guidelines. Even with accessible Web design, however, any one page cannot meet the needs of each and every individual. The features enabled by accessibilityWorks allow users to control their Web experience to meet individual needs not addressed by the guidelines (see, for example, References 38–41). Thus, even well-designed pages can be changed to meet a broader spectrum of user needs than that addressed by the standards and regulations,[42] thus increasing the number of people who are able to use the entire World Wide Web with ease. Taking a similar approach to that of the W3C** (World Wide Web Consortium) guidelines

regarding User Agent Accessibility for browers,[43,44] accessibilityWorks allows individuals to have control over existing content on the open Web with respect to both the way pages are presented and how user input is provided.

ACKNOWLEDGMENTS

We acknowledge IBM Corporate Community Relations (http://www.ibm.com/ibm/ibmgives/about/index.shtml), particularly Stan Litow and Paula Baker, for continuing support of this research effort. We are also grateful to our early collaborators, Peter Fairweather, Rich Schwedtfeger, and Sam Detweiler, for their insights on this work.

CITED REFERENCES

1. V. L. Hanson, "Facing the Future: Including Elderly Users When Considering Universal Access," in *Universal Access in HCI: Inclusive Design in the Information Society, Volume 4*, C. Stephanidis, Editor, Lawrence Erlbaum Associates, Inc., Mahwah, NJ (2003), pp. 394–398.
2. S. Trewin, "Configuration Agents, Control, and Privacy," *Proceedings of the ACM 2000 Conference on Universal Usability (CUU 2000)*, Arlington, VA, November 16–17, 2000, ACM, New York (2000), pp. 9–16.
3. J. A. Jacko, A. B. Barreto, G. J. Marrnet, J. Y. M. Chu, H. S. Bautsch, I. U. Scott, and R. H. Rosa, "Low Vision: The Role of Visual Acuity in the Efficiency of Cursor Movement," *Proceedings of the Fourth International ACM Conference on Assistive Technologies (ASSETS 2000)*, Arlington, VA, November 13–15, 2000, ACM, New York (2000), pp. 1–8.
4. V. L. Hanson, "Web Access for Elderly Citizens," *Proceedings of the Workshop on Universal Accessibility of Ubiquitous Computing (WUAUC'01)*, Alcácer do Sal, Portugal, May 22–25, 2001, ACM, New York (2001), pp. 14–18.
5. IBM Home Page Reader 3.04, IBM Corporation, http://www-3.ibm.com/able/solution_offerings/hpr.html.
6. V. L. Hanson and S. Crayne, "Personalization of Web Browsing: Adaptations to Meet the Needs of Older Adults," *Universal Access in the Information Society*, in press.
7. V. L. Hanson and J. T. Richards, "Achieving a Usable World Wide Web," *Behaviour and Information Technology*, in press.
8. J. T. Richards and V. L. Hanson, "Web Accessibility: A Broader View," *Proceedings of the Thirteenth International ACM World Wide Web Conference (WWW2004)*, New York, May 19–21, 2004, ACM, New York (2004), pp. 72–79.
9. P. Gregor, A. F. Newell, and M. Zajicek, "Designing for Dynamic Diversity—Interfaces for Older People," *Proceedings of the Fifth International ACM SIGCAPH Conference on Assistive Technologies (ASSETS 2002), Edinburgh*, Scotland, UK, July 8–10, 2002, ACM, New York (2002), pp. 151–156.
10. *Older Adults and Information Technology: A Compendium of Scientific Research and Web Site Accessibility Guidelines*, National Institute on Aging, Bethesda, MD (2002).
11. A. Arditi, "Web Accessibility and Low Vision," *Aging & Vision* **14**, No. 2, 2–3 (Fall, 2002).
12. K. V. Echt, "Designing Web-Based Health Information for Older Adults: Visual Considerations and Design Directives," in *Older Adults, Health Information, and the World Wide Web*, R. W. Morrell, Editor, Lawrence Erlbaum Associates, Inc., Mahwah, NJ (2002), pp. 61–88.
13. S. J. Czaja and C. C. Lee, "Designing Computer Systems for Older Adults," in *The Human-Computer Interaction Handbook*, J. Jacko and A. Sears, Editors, Lawrence Erlbaum Associates, Inc., Mahwah, NJ (2003), pp. 413–427.
14. S. Dailey, "Using Cognitive Aging and Vision Research to Develop Senior-Friendly Online Resources," AARP (2004), http://assets.aarp.org/www.aarp.org_/articles/research/oww/university/DaileySlides.ppt.
15. S. Trewin, "Automating Accessibility," *Proceedings of the Sixth International ACM SIGACCESS Conference on Assistive Technologies (ASSETS 2004)*, Atlanta, GA, October 18–20, 2004, ACM, New York (2004), pp. 71–78.
16. WebAdapt2Me, IBM Corporation, http://www-306.ibm.com/able/solution_offerings/WebAdapt2Me.html.
17. P. G. Fairweather, V. L. Hanson, S. R. Detweiler, and R. S. Schwerdtfeger, "From Assistive Technology to a Web Accessibility Service," *Proceedings of the Fifth International ACM SIGCAPH Conference on Assistive Technologies (ASSETS 2002)*, Edinburgh, Scotland, UK, July 8–10, 2002, ACM, New York (2002), pp. 4–8.
18. Mozilla 1.x Releases, The Mozilla Organization, http://www.mozilla.org/releases/.
19. Download Firefox, The Mozilla Organization, http://www.mozilla.org/products/firefox/all.html.
20. M. De La Rue and S. Snider, *Linux Accessibility HOWTO*, The Linux Documentation Project, http://www.tldp.org/HOWTO/Accessibility-HOWTO/.
21. D. Bolter, Linux Accessibility Resource Site (LARS), Adaptive Technology Resource Centre (ATRC), http://lars.atrc.utoronto.ca/.
22. Disability Access to GNOME, The GNOME Project, http://developer.gnome.org/projects/gap/.
23. Welcome to XULPlanet, XULPlanet.com, http://xulplanet.com/.
24. N. Deakin, *XPCOM Interfaces*, XULPlanet.com (November 13, 2004), http://xulplanet.com/tutorials/xultu/xpcom.html.
25. ViaVoice, IBM Corporation, http://www-306.ibm.com/software/voice/viavoice/.
26. S. Keates, P. Langdon, P. J. Clarkson, and P. Robinson, "User Models and User Physical Capability," *User Modeling and User-Adapted Interaction (UMUAI)* **12**, No. 2–3, 139–169 (2002).

27. E. Grundy, D. Ahlburg, M. Ali, E. Breeze, and A. Sloggett, *Disability in Great Britain: Results of the 1996/7 Disability Follow-Up to the Family Resources Survey*, Department for Work and Pensions, UK.

28. Disabilities/Limitations, National Center for Health Statistics, http://www.cdc.gov/nchs/fastats/disable.htm.

29. *The Market for Accessible Technology—The Wide Range of Computer Abilities and Its Impact on Computer Technology*, Forrester Research, Inc. (2003), http://www.microsoft.com/enable/research/phase1.aspx.

30. S. Trewin and H. Pain, "Keyboard and Mouse Errors Due to Motor Disabilities," *International Journal of Human-Computer Studies* **50**, No. 2, 109–144 (February 1999).

31. S. Trewin, "Extending Keyboard Adaptability: An Investigation," *Universal Access in the Information Society*, in press.

32. R. Kjeldsen, "Head Gestures for Computer Control," *Proceedings of the IEEE ICCV Workshop on Recognition, Analysis and Tracking of Faces and Gesture—Real Time Systems (RATFG-RTS '01)*, Vancouver, BC, Canada, July 13–August 13, 2001, IEEE Press, New York (2001), pp. 61–67.

33. R. Kjeldsen, A. Levas, and C. Pinhanez, "Dynamically Reconfigurable Vision-Based User Interfaces," *Machine Vision and Applications* **16**, No. 1, pp. 323–332 (2004).

34. J. Brewer, Web Accessibility Initiative (WAI), World Wide Web Consortium, http://www.w3.org/WAI/.

35. W. Chisholm, G. Vanderheiden, and I. Jacobs, *Web Content Accessibility Guidelines 1.0*, World Wide Web Consortium (May 5, 1999), http://www.w3.org/TR/WCAG10/.

36. *Section 508 Standards*, Center for IT Accommodation (CITA), Office of Governmentwide Policy, U.S. General Services Administration, http://www.section508.gov/index.cfm?FuseAction = Content&ID = 12.

37. J. Maeda, K. Fukuda, H. Takagi, and C. Asakawa, "Web Accessibility Technology at the IBM Tokyo Research Laboratory," *IBM Journal of Research and Development* **48**, No. 5/6, 735–748 (2004).

38. B. Leporini and F. Paternò, "Increasing Usability when Interacting through Screen Readers," *Universal Access in the Information Society* **3**, No. 1, 57–70 (March 2004).

39. J. Mankoff, A. Dey, U. Batra, and M. Moore, "Web Accessibility for Low Bandwidth Input," *Proceedings of the Fifth International ACM SIGCAPH Conference on Assistive Technologies (ASSETS 2002)*, Edinburgh, Scotland, UK, July 8–10, 2002, ACM, New York (2002), pp. 17–24.

40. *Beyond Accessibility: Treating Users with Disabilities as People*, Neilson Norman Group (November 11, 2001), summary available at http://www.useit.com/alertbox/20011111.html.

41. J. J. Powlik and A. I. Karshmer, "When Accessibility Meets Usability," *Universal Access in the Information Society* **1**, No. 3, 217–222 (June 2002).

42. V. L. Hanson, "Taking Control of Web Browsing," *New Review of Hypermedia and Multimedia* **10**, No. 2, 127–140 (December 2004).

43. J. Gunderson, "W3C User Agent Accessibility Guidelines 1.0 for Graphical Web Browsers," *Universal Access in the Information Society* **3**, No. 1, 38–47 (March 2004).

44. I. Jacobs, J. Gunderson, and E. Hansen, *User Agent Accessibility Guidelines*, World Wide Web Consortium (October 16, 2002), http://www.w3.org/TR/2002/PR-UAAG10-20021016/.

Accepted for publication January 24, 2005.
Published online July 13, 2005.

Vicki L. Hanson
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (vlh@us.ibm.com). Dr. Hanson is a research staff member and manager of the Accessibility Research department at the Thomas J. Watson Research Center. She received a B.A. degree in psychology and speech pathology and audiology from the University of Colorado in 1974, and M.A. and Ph.D. degrees in cognitive psychology from the University of Oregon in 1976 and 1978, respectively. She worked as a postdoctoral fellow in the Laboratory of Language and Cognition at the Salk Institute and then as a research associate at Haskins Laboratories in New Haven before joining the IBM Research Division in 1986. She is active in the ACM (Association of Computing Machinery) organization and currently serves as Chair of the ACM Special Interest Group on Accessible Computing (SIGACCESS). She is a recent ACM Fellow.

Jonathan P. Brezin
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (brezin@us.ibm.com). Dr. Brezin is a research staff member in the Enhanced Web Experience department at the Thomas J. Watson Research Center. He received a B.S. degree in mathematics from Cornell University in 1963 and a Ph.D. degree in mathematics from the City University of New York in 1967. He served on the faculties of the University of Minnesota and the University of North Carolina before joining IBM in 1983 to work on the 801 minicomputer project.

Susan Crayne
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (crayne@us.ibm.com). Ms. Crayne is a software engineer in the Accessibility Research department at the Thomas J. Watson Research Center. She received a B.S. degree in mathematics from CUNY City College, where she was a member of Phi Beta Kappa. She then became Manager of Application Development tools at Auragen Systems, a startup computer manufacturer. Subsequently, she ran a software consulting business specializing in Windows development and image-processing applications. After joining IBM in 1997, she co-designed and developed a tool for teaching mathematics and science to middle school students. Since 2000 she has focused her efforts on improving the accessibility of the Web.

Simeon Keates
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (lsk@us.ibm.com). Dr. Keates is a research staff member in the Accessibility Research department at the Thomas J. Watson Research Center. He received B.A. (with honors) and M.A. degrees in engineering and a Ph.D. degree in rehabilitation engineering from the University of Cambridge, UK. After completing his Ph.D. degree in 1997, he continued his research at Cambridge until 2003, when he joined IBM. His work both at Cambridge and IBM has focused on computer access for motion-impaired users and also on the broader issues of inclusive design. Dr. Keates is a member of the Institute of Electrical and Electronics Engineers and the Association of Computing Machinery.

Rick Kjeldsen
IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (fcmk@us.ibm.com). Dr. Kjeldsen received his B.S.E.E degree (with great distinction), from Clarkson University in 1981, his M.S. degree from the University of Massachusetts in 1987, and his Ph.D. degree from Columbia University in 1997. He was a member of the Exploratory Computer Vision department at the Thomas J. Watson Research Center from 1987 through 2001, at which time he joined the Accessibility Research department. He has published and filed patents on a wide range of subjects in artificial intelligence, computer vision, and perceptual user interfaces. His current research is aimed at the application of human-centric computer vision to create advanced multimodal user interfaces for both general and special needs users.

John T. Richards
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (ajtr@us.ibm.com). Dr. Richards is a research staff member and manager at the Thomas J. Watson Research Center. He received a B.A. degree in psychology from Alma College in 1974, and M.S. and Ph.D. degrees in cognitive psychology from the University of Oregon in 1976 and 1978, respectively. He joined the IBM Research Division in 1978. He is a Fellow of the Association of Computing Machinery.

Cal Swart
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (cals@us.ibm.com). Mr. Swart is a senior software engineer in the Enhanced Web Experience department at the Thomas J. Watson Research Center. He received a B.S. degree from Calvin College in 1969 and joined IBM in 1982. He has served in research and programming roles in numerous graphics, networking, K-12 internet access, and PDA (Personal Digital Assistant) programming projects. Most recently, he has been concentrating on Web accessibility and portal applications.

Shari Trewin
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (trewin@us.ibm.com). Dr. Trewin is a research staff member in the Accessibility Research department at the Thomas J. Watson Research Center. She received B.Sc. (with honors) and Ph.D. degrees in computing science and artificial intelligence from the University of Edinburgh, Scotland, in 1991 and 1998, respectively. After completing a Ph.D. degree, she continued her research at the University of Edinburgh until 2000, at which time she joined IBM. She serves as co-editor of the Universal Remote Console suite of standards for remote control of devices and services, developed within the INCITS (International Committee for Information Technology) V2 standards committee. ■