Managing usability for people with disabilities in a large Web presence

M. King
J. W. Thatcher
P. M. Bronstad
R. Easton

The case for ensuring that Web sites are usable by people with disabilities is strong in light of the World Wide Web's ubiquity as an essential customer interface for most organizations, the considerable disposable income of people with disabilities, and a growing number of accessibility regulations being applied to public Web interfaces. For small Web sites having a few thousand pages managed by a single centralized IT (information technology) department, ensuring accessible content is a well-understood process supportable with a variety of off-the-shelf solutions. For organizations owning multiple large sites containing millions of pages of content authored by hundreds or thousands of employees and applications, standards compliance management is significantly more complex in general and is particularly challenging in the context of accessibility. This paper describes the design and development of processes and solutions for establishing and maintaining accessibility for a very large Web presence. This includes site templates employing advances in coding techniques that offer dramatic usability improvements for users with disabilities and efficient enterprise-wide compliance-monitoring processes that cover all accessibility standards, including standards requiring human judgment to evaluate.

INTRODUCTION

It is no longer difficult to find organizations that have been working several years to create an accessible Web presence. However, success is not yet common. Most surveys reveal large gaps between accessibility standards and what is implemented. For example, only 22 percent of the Web sites for United States government agencies, which have been mandated to be accessible by Section 508[1] for over three years, meet Section 508 requirements.[2]

Even with management team commitment to Web accessibility standards, when a Web presence is large and has hundreds or even thousands of people and applications who contribute content to it, developing and managing standards compliance is complex. Naturally, it is desirable to start by building compliance at points of entry into the system. This can be aided by standardizing the suite of authoring and content management tools supported in the enterprise and through provision of templates that provide accessibility features. The first section of this paper describes innovative use of template technology that goes beyond providing compliance; the templates enable even complex sites to dramatically increase usability for people with disabilities.

There is no way, however, to control or even be aware of every point of entry into the Web space in order to block inaccessible content from entering. A compliance management system for a large enterprise first requires a content discovery mechanism. Web content discovery systems—known as "crawlers"—vary widely in scalability, performance, and reliability, and managing them is complex. Once discovered, content must then be analyzed for standards compliance. Several commercial products offer automated accessibility analyses of Web documents, and some of these products incorporate Web crawlers. These tools vary greatly in their performance and scalability.

After these automated tools discover and analyze Web content, results must be communicated. For some enterprises there may be reports on exorbitant numbers of Web documents addressed to different content owners, and reliably segmenting the report data into meaningful categories is very challenging.

For example, the automated tools may generate a report on 80,000 Web pages that is sent to the marketing department. This report may state that 30 percent of their content is not compliant, but 20,000 of these pages may actually be owned by the technical support organization. The second section of this paper describes how IBM has addressed these challenges in building an automated compliance monitoring system for its own internal and external Web spaces.

Fully automated compliance monitoring, though useful, cannot indicate whether documents comply with Web accessibility standards. It can only identify problems if the violations of standards are machine-detectable. In fact, more than half of the provisions included in most standards sets, for example, U.S. Section 508[1] or the World Wide Web Consortium (W3C**) Web Content Accessibility Guidelines (WCAG)[3], require human judgment to evaluate. To accurately measure full compliance, human review of content is mandatory. For a large Web space, then, this task may not be feasible, and thus it seems impossible to determine the complete accessibility picture. One could randomly select a few sites for human review, but that would not provide a pan-enterprise perspective. The final section of this paper describes a new approach to this problem that leverages knowledge gained from an automated compliance system to build a statistical model of the Web space, allowing for the creation of an economically feasible sampling system and prediction algorithm that creates a total compliance picture even for a Web space that includes millions of documents.

DEVELOPING USABLE ACCESSIBLE SITES

A large number of processes affect the development and maintenance of accessible Web sites and applications. For example, authors and developers can be educated to create accessible content. Authoring, development, and host services packages affect how people with disabilities interact with sites. These topics are discussed on the IBM Accessibility Center Web site.[4] In this section, we focus on the development of site templates (such as those recently implemented in IBM) that have yielded phenomenal gains, not just in standards compliance, but in usability enhancements for people with disabilities. These gains came at little cost and are very popular with their target audience.

Many people have found the Web more useful as it grows more capable of information consolidation and personalization. However, the Web has become more confusing to people with disabilities. Web portals, which consolidate many information sources, are an example of one such simultaneous advance and step back. A portal page, with its multiple views into many different kinds of information, can be a boon to those who are able to visually scan it like a dashboard and almost instantly detect new information or points of interest. On the other hand, portals offer incomprehensible chaos to the visitor who is blind or visually impaired.

Satisfying the requirements described in accessibility standards is helpful; yet compliance with the standards is not enough to bring order to the chaos. The accessibility features described in this section make a complex page, such as a portal, extremely usable. Moreover, these features can be used to great benefit on any type of Web page.

Navigation challenges

Today's Web portal page is a multicolumn collection of small blocks or windows, each providing a view into a specific set of information. The computer applications that produce these windows are commonly called *portlets*. Each portlet collects and displays its information independent of others that may share space on the portal Web page.

As an example, we consider the needs of employees in a large enterprise with an abundance of information resources. The enterprise intranet portal serves a large employee audience and includes multiple portal pages, each with a collection of portlets. In its nonmodified form, the topmost portal page is the company's main bulletin board, which includes company news, stock market information, an enterprise directory, personal link lists, and search portlets. A second portal page includes portlets for work tools, such as asset and expense reporting, selling information, and personalized industry news. A third portal page concerns employees, with portlets for various human-resources and employee-benefit programs.

To add personalization, employees are given the ability to move portlets around in order to suit individual preferences. Any portlet can be moved to any portal page and positioned on that page as preferred. The result is a large collection of information sources and tools that employees add, delete, and rearrange to meet their own needs and goals. In essence, every employee has his own customized enterprise portal.

The challenge in adding accessibility to this portal is: How can a blind person gain a sense of, and remember, what and where all the pieces comprising the portal are?

Accessibility solutions

An initial accessibility accommodation for navigation in this portal consisted of a pair of "skip links" for each portlet, allowing the user to skip to the next portlet or the previous portlet. These generic aids might have met accessibility requirements, but they failed to go beyond compliance and added little to usability.

Gaining a sense of even a single portal page containing only a few portlets requires strong concentration; understanding it is difficult to achieve. For one blind employee, understanding the home page took many visits and, finally, some explanation from a sighted colleague. In order to improve understanding, the templates used to create portal pages were enhanced with the following methods: structured HTML, landmarks, a page index that accurately described the page, access keys, and provisions for personalized page styling. Each of these techniques (described in the following sections) is embedded directly into the templates, making it easier for portlet developers and content producers to keep new pages accessible. An additional benefit of this approach is that developers no longer need to remember to include obscure or infrequently used features.

Table 1 Example showing replacement of non-standard HTML with semantically correct heading; A. non-semantic technique; B. semantically correct technique

A	<span class = "page-heading">Text for heading </span> ... <span class = "sub-head">Text for sub-heading </span>
B	<h1>Text for page heading</h1> ... <h2>Text for sub-heading</h2>

Structured HTML

HTML coding techniques were changed to use a semantically correct structure: that is, headings, paragraphs, and lists. The previous technique used HTML spans to apply a Cascading Style Sheet (CSS) appearance to a line of text, making the text appear like a heading. Spans were eliminated in favor of true heading markup. These techniques are illustrated in *Table 1*. This change alone had immediate beneficial impact because assistive tools known as screen readers have facilities that take advantage of good structure. For example, a simple screen reader command can quickly invoke the reading of all headings on a Web page, giving a blind visitor an outline of the important information on the page.

Hiding accessibility information

Part of the portal design update consisted of moving from a layout technique that depended on tables and "spacer" images to one that used CSS technology for

Table 2 Examples of code that hides accessibility information: (A) CSS code causes text of class "access" to be hidden from visual display by placing it 3000 pixels to the left of the browser window; (B) the class "access" code wraps around a link, hiding if from view; and (C) the class "access" code wraps around a paragraph, hiding it from view.

A	`.access { position: absolute; left: -3000px; width: 500px; }`
B	`<a class = "access" href = "#page-index"> Jump to portlet page index</a>`
C	`<p class = "access">Additional accessibility information for this site can be found <a href = " http://thissite/access-stmt.html"> on the Accessibility Statement page.</a></p>`

positioning. Because this new approach used no spacer images, the traditional method of attaching a skip link to a spacer image needed reconsideration. A CSS technique was developed that replaced the spacer image technique. The technique "parks" accessibility information in an imaginary space far to the left of what a visual Web browser is capable of displaying. While screen readers can find this information, Web browsers do not. This new method (called "access") was tested with a wide range of screen readers and is now used for more than just skip links.[5] See *Table 2* for examples of making accessibility information invisible to the user.

Landmarks: What part of the page am I listening to now?

Imagine a shopper not knowing whether he or she is standing in a shoe store or a bookstore. This is similar to the experience of trying to understand a portal page with portlets when using a screen reader. Knowing which section of the page currently contains the focus is crucial to understanding the content presented at that time; that is, the context in which the information is interpreted is analogous to the facility in which a shopper finds himself or herself. Therefore, to provide context, major sections of the page are marked with *landmarks* that are read by screen readers but not displayed in Web browsers. The portal page includes three landmarks, shown in bold text in *Table 3*, followed by the HTML code for each. Interior "article" pages with left-hand navigation links and sidebars contain additional landmarks for those elements.

Table 3 Example of landmarks and corresponding HTML code

```
Start of masthead <h2 class = "access">Start of
masthead</h2>
    Start portal tabs <h2 class = "access">Start portal
    tabs</h2>
       A list of all portlets on this page <h2 class =
       "access">A list of all portlets on this page.</h2>
```

Table 4 Example of a portal page index.

```
<div>
  <h2 class = "access">A list of all portlets on this page.
  </h2>
  <ul id = "page-index" class = "access">
    <li><a href = "#sr">Search</a></li>
    <li><a href = "#wn">What's new</a></li>
    <li><a href = "#xx">Mail and calendar</a></li>
    <li><a href = "#on">On Demand</a></li>
    <li><a href = "#ma">Market report</a></li>
    <li><a href = "#el">Essential links</a></li>
    <li><a href = "#ne">News</a></li>
  </ul>
</div>
```

The portal page index: What portlets are on this page?

Completely new to this version of the portal is a portal page index: a table of contents of what appears on the page. Because the portal can be highly personalized with a large variety of possible content, the portal page index is a dynamically constructed index that matches the precise contents of the page. It is formulated as a list of links and is therefore easily handled by at least two of each screen reader's built-in functions: the facility for reading links and the facility for reading lists. A skip link at the top of the page and a skip link at the start of each portlet offer quick ways of finding or returning to the page index. The block of text shown in *Table 4* has the class of "access," which hides it off to the left of the browser window. It also has the identifier "page-index," which is the target of many of the skip links within the page.

Table 5 Example of the use of a semantically correct HTML level-2 heading.

```
<div class = “portlet-head-blue-med”>
  <h2 class = “portlet-head-blue-med”>Essential links
  </h2>
  <span class = “portlet-icons”>...multiple image
  declarations skipped for simplicity...</span>
</div>
```

Portlet headings: Coding as an HTML heading makes them easier to find

Each portlet has a heading (an HTML heading level element) containing the portlet's title, a skip link to the page index, and several properly annotated images providing edit, maximize, minimize, and help functions. Simply coding the portlet's title as a legitimate HTML heading makes each of the portlet headings easier to discover with a screen reader. See *Table 5* for sample HTML showing the use of H2, a semantically correct heading element that can be found and read by screen readers.

Access keys: Fast access to common Web page facilities

Access keys are helpful to those who cannot use a mouse, and are gaining popularity for providing fast access to common Web page facilities. A de facto standard is slowly emerging[6] but is not yet accepted by any standards organization (e.g., W3C). We used the following subset of the emerging standard:

Alt + 0 links to this site's accessibility statement.
Alt + 1 links to the portal home page.
Alt + 2 skips to the index of portlets on this page.
Alt + 9 links to the feedback page.

The Alt + 0 key is well known as a pathway to finding accessibility information. Both blind people and people with mobility impairments often try Alt + 0 to learn of features that a site offers.

Some controversy surrounds the use of Alt keys for accessibility information,[7] but there are no better alternatives currently available. The use of Alt keys has become quasi-standard, and people aware of access keys will expect to use the Alt key combinations. Lastly, experienced screen reader users are accustomed to the keystroke combinations used by their screen reader. For example, Shift + Alt is the access key combination in IBM Home Page Reader, whereas the Alt + Plus key works with the JAWS** (Job Access with Speech) screen reader.

Table 6 Example of a personal style sheet.

```
userContent.css (personal style sheet file)
#w3-ibm-com {font-size:140% !important;}
```

Personal style sheets: Overriding page styles

What if one needed to override page styles? Every page on the site carries an identifier in the body tag that can be used to declare styles specifically for the site. For example, a visitor with low vision might prefer a font-scaling factor for this site but does not want to change the browser setting that affects all sites. A user style sheet, currently supported by all modern browsers, is the place to add a declaration based on the special identifier included on all pages (for example, every page on the IBM intranet site includes a `BODY` element stating `<body id="w3-ibm-com">`). The example shown in *Table 6* increases font size to 140 percent of normal size. It could just as easily be crafted to change font and background colors for higher contrast.

Help for people with other disabilities

Though many of the methods described in this paper seem intended to help only users who are blind, some help people with other disabilities. Personal style sheets, as just described, help those with low vision. Scalable fonts, not described in detail, but easy to implement, also help those with low vision. Access keys help not only users who are blind, but those with mobility impairments who have difficulty using a mouse. For these people, access keys offer rapid access to the search field and feedback links. On pages inside the portal, pages formatted as normal articles, an additional feature helps people with mobility impairments. A “Skip to main content” link is made visible the first time they touch the “Tab” key after loading the page. This link saves many additional “Tab” keystrokes in moving through the usual plethora of navigation links.

Benefit examples

A person able to see and use a mouse can survey a portal page in a matter of seconds and see what is available. Without the assistance of the special accessible usability features described in this article, a blind person would find it almost impossible to accomplish the same task. The blind visitor would make a first attempt by sequentially reading every word on the page and attempting to discern the structure from semantics. The result for an experienced screen reader user, after 7 to 12 minutes of careful inspection, would be a confused picture of the content.

Instead, with these accessibility features a screen reader user can press the screen reader's hot key to list all headings on a page in outline form. It takes less than 15 seconds to press the hot key and listen to the screen reader read the entire list. The blind user's understanding of what is available is now very similar to that of a sighted user. This is a productivity improvement of as much as 4,800 percent! This may seem like a staggering number, especially given the simplicity of the solution. However, it is typical of the magnitude of benefit that can be obtained with attention to the usability of accessible design.

Summary

Using the techniques described in this paper, people who are blind or visually impaired do not need, as they have in the past, to memorize the position or sequence of every item on complex Web pages. The system now provides lists of headings, lists of links, page indexes, landmarks, and access keys to help improve their ability to move more quickly to the information they want and to add information context.

The portal implementation of the features we have described has resulted in one of the first portal sites to offer a set of accessibility features this rich in capability and choice. Embedding these ease-of-access features directly into templates greatly improves the probability that new pages are both accessible and usable.

AUTOMATED PROGRESS MEASUREMENT

As action plans (such as the widespread adoption of templates) to improve the accessibility of content are executed, we expect to see improvement in the level of compliance to standards. To manage this progress, metrics are essential in maintaining levels of quality of the system. Obviously, metrics constitute overhead, so managing cost through efficiency is also important. In this section, we will survey the approach IBM has taken to developing the automated compliance reporting component of its Web Accessibility Standards Reporting Process (WASRP).

Business requirements

The WASRP has two primary objectives: to inform the chief information officer of progress by division or business unit, and to give application owners and portfolio managers the information they need to plan and prioritize remediation efforts. Thus, it is important that the process be capable of delivering data that can be categorized by business unit and owning manager.

Each business unit executive has a representative on the team responsible for enforcing Web accessibility improvement. These business unit representatives and their executives need new data often enough to manage progress toward their annual targets, but it takes several weeks to receive a report, review it, and lead a specific application through the change process. New data indicating the same trends as previous data is not of much use. To allow sufficient time to ensure the opportunity for significant and useful change in the measurements, a 30-day minimum interval between one report's delivery date and the beginning of the next reporting cycle was set. Because business unit representatives needed data at least every 90 days, preferring it in 60 days if possible, the report generation cycle time was set from 30 to 60 days.

Because IBM's corporate mandate requiring that all the information technology used or produced by IBM be accessible does not designate any exceptions, the scope of the WASRP measurements is any content on IBM's Internet or intranet sites. This is a wider scope than that of any other IBM Web content standard. The WASRP, when it began in 2000, broke new ground in IBM. The only application in IBM that had any measure of how much content existed was the Web search application. The WASRP even included sites that the search application filtered out. This all-inclusive scope would test our concept of scalability.

Technology requirements

The preceding business requirements led to four primary components for the accessibility standards reporting process:

1. *Web crawler*: Crawls each Web space (Internet and intranet) to locate and capture the content for analysis.

2. *Analysis engine*: Analyzes the Web content for standards violations.
3. *Ownership identification*: Identifies the owner of each page of content.
4. *Report generation*: Combines the analysis data with the ownership data to generate reports.

Scalability

In March of 2000, when WASRP development began, the accessibility standards for U.S. Section 508 were still being written. Because we could not locate any existing work in the area of Web accessibility that approached the magnitude of what we were about to attempt, for practical reasons we started WASRP development by working only on public Internet sites. This kept our mission in line with the business priority of focusing first on the customer while giving the development team an opportunity to begin its activity in a relatively small Web space (roughly 3.5 million pages). It took 18 months to move from there to the intranet. That step proved to be surprisingly more difficult than anticipated. We thought we had a scalable solution, but in 2002, when we attempted to go from a 3.5-million page capacity to a 30-million page capacity, we learned what scalability really meant. This jump in scale affected every component of the process, but crawling was the process most negatively affected.

Performance

The primary factor affecting WASRP cycle time is the speed at which data can be moved across networks. Thirty million pages of Web content is a great deal of data, and it takes time to find it and read it. In 2002, we set our absolute minimum throughput requirement at 30 million pages in 60 days, or 500,000 pages a day. Our projected target was one million pages a day. The process has been performing very close to that level since January of 2003, and with a production hardware upgrade, the one-million-page target will easily be surpassed.

Experimentation with the architecture has been done and continues (see *Figure 1*), to assess the effects of component rearrangement, changing the amount of parallelism in the process. One fundamental principle seems to apply: the less we move the data, the better. This tends to result in an architecture which is more serial than parallel. Intuitively, this does not seem optimal, and we believe this is an area where there is still plenty of room for further study.

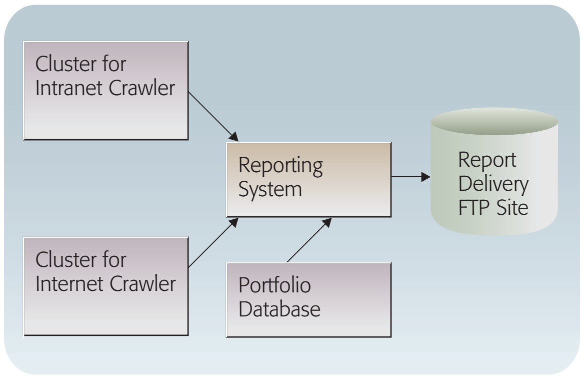


Figure 1
WASRP architecture

The crawling component: Content discovery and retrieval

Because the scope of our implementation included all content available on IBM's Internet and intranet sites, it was necessary to determine what that entailed. When the project began, we had an estimate of the number of Web pages from the database that was used to manage the enterprise application portfolio, which had been greatly improved as part of IBM's Y2K effort. We knew there were some classes of sites that were not included in that database, but we did not know how much content was on those sites.

In any large enterprise, the only way to really know how much Web content is online is to employ a Web crawler. This is a software package that, given a set of starting Web pages, looks for and follows every active element (link, button, etc.) and catalogs its findings. As the crawler is analyzing a Web page to find all the other pages to which it points, a process known as discovery, it can also store the page content in its repository. We evaluated several crawlers based on the criteria of the quality of their discovery abilities, their speed, and their scalability.

Some of the accessibility checking tools that were available when we started the project included a crawling component. We quickly found, however, that the tools were not capable of meeting any of our minimum criteria. We learned that enterprise-scale Web crawlers are a very advanced technology requiring enormous resources to develop and maintain. The field continues to evolve rapidly along with the Web. During the last four years, we have investigated approximately six crawlers, and

■ It is desirable to start by building compliance at points of entry into the system ■

we have completely changed the crawling package twice. In our present implementation, crawling is built on an IBM research technology known as WebFountain*.[8]

With our present WebFountain configuration, we store, on average, 1.5 million documents a day in the data repository. Storing the documents in a local data repository provides a tremendous throughput advantage because the checker described in the next section does not have to go back to the Web to read the content; the checker can access the content in a local database, which is faster by nearly three orders of magnitude.

Compliance analysis

Once we have crawled the Web space and stored all the content we wish to analyze in the crawler's data repository, we have to analyze it for accessibility errors. In March of 2000, most of the tools for Web accessibility checking were still in a very rudimentary state, except for Bobby** Version 3.2.[9] Initially, we built Perl** scripts that ran the command line version of the Bobby tool to analyze the data in local files and write reports in XML. This proved to be unreliable and slow as well as clumsy to manage, especially when we were trying to run multiple instances of Bobby at one time to increase throughput.

We needed a better way to control the Bobby tool. Only the checking function was needed, and all the other functions built around it were interfering. We approached the Bobby tool developers, who were at that time at the Center for Applied Special Technologies, to see if we could access Bobby's checking function through an application programming interface (API). They had not intended their Java** class definitions to work as an external API, but the definitions were designed well enough that using the definitions as an API became a superb solution. We developed a middleware package, IBM Web Standards Checker (IWSC), that reads data from the crawler repository, feeds it to a configurable number of instances of Bobby that are running in parallel, and then writes the results to a reporting database. This gave us the control we needed to get throughputs in excess of 80,000 pages per hour on a single CPU system with 1 gigabyte of RAM running at 1.5 MHz. By running five such machines in parallel, we could process 250,000 to 400,000 pages per hour. Because the architecture design includes hardware scalability, by adding more hardware, we could increase that throughput simply by adding more systems to the cluster.

Since then, we have abstracted the IWSC Java interface with Bobby in an IWSC checker interface class. This allowed us to write Java classes to serve as wrappers around the APIs of other checking programs and plug them into IWSC. We have written such an interface class for WebKing**, and we can now plug WebKing's checking capabilities into WASRP. The extensibility of this architecture has proven to be an extremely valuable feature as we work to improve and expand our checking capabilities.

Content ownership identification

In order to generate reports that categorize data by line of management, we need means for identifying a management owner of the pages discovered by the crawler. One option we have considered is to use the meta-data from the page content. Some IBM content has meta-data tags that name the page author. We could extract that information, look up the author in our enterprise directory of employees, and associate the page with the employee's business unit. That meta-data, however, is frequently not present. Another option is to compare the URL (uniform resource locator) string to a list of URLs for which the owner is known. This, for now, has proven to be the better option.

The IBM enterprise portfolio management database contains data about IBM's Web applications. Typically, an application can be thought of as a Web site, or a collection of Web sites, differing only in localization features, that has been developed to serve a specific set of business objectives. The portfolio database record for each application includes the URLs for each deployment of that application. We refer to the URL for the home page as the top-level URL.

The Java program we developed to assign each of the 30 million URLs discovered by the crawler to an application is called Internet Resource Ownership Identifier (IROI). To assign a URL found by the crawler to an application, IROI compares the URL to top-level URLs in the portfolio database to determine if the URL that the crawler found is a child of any of the top-level URLs. For example, if the crawler found http://www.ibm.com/employment/uk/faq.html, IROI would recognize it as belonging to the application listed in the portfolio with the top-level URL http://www.ibm.com/employment/uk/. Unfortunately, the process is not always that simple. Sometimes the form of the URL in the portfolio database is not the same as what the crawler finds. For instance, the portfolio database could accurately list the same application with a home page as http://www.uk.ibm.com/employment. Note the different location of the country code. Nonetheless, it refers to the same home page. If we load that page and click on the link labeled “FAQ” (frequently asked questions), the address that is displayed in the browser address bar is http://www-5.ibm.com/employment/uk/faq.html. Given these last two forms of the URL string, it is now less obvious that the FAQ page is a child of the home page. Nevertheless, it is, and IROI can determine that. There is a wide variety of such “tweaking” of URL strings that must be accounted for in IROI’s algorithms. The tweaking is configurable in several ways, primarily through a set of Java classes that implement an abstract class that represents an allowed tweaking.

There are some pages, such as portals, that can contain content from multiple sources. The content thus may have multiple owners that may even be from different business units. At this time, our measurement system makes the portal owner responsible for overseeing compliance of portlet content.

This has debatable merit, but is presently a technological limitation. To resolve this issue, we first need a means for the content-checking engine to provide a portlet context for any errors that it may find. Then, we would need a means other than URL string analysis to identify the owner of the portlet (this would likely be analysis of metadata).

As IWSC can run a configurable number of checker instances in parallel, IROI can run a configurable number of URL comparison threads in parallel. It presently runs on a server with four 700 MHz processors and has an average throughput of over 300,000 URLs per hour.

Reporting

The accessibility error data from IWSC and the ownership data from IROI are stored in a DB2* database where they can be combined, by URL, to generate reports. We run a Java application that generates standard reports in comma-separated-value format for delivery to the business unit representatives, but, with all the data in a relational database, it is also easy to generate any kind of custom report that is desired.

Summary

The automatic compliance reporting component of WASRP is an assembly of component technologies that all have other uses beyond automated Web accessibility-standards-compliance reporting. We built the ability to use best-of-breed component technologies in the system by designing an architecture that allows the component technologies to be individually upgraded or changed. This has also allowed the system’s functions to be more easily expanded, for example, to include reporting on other types of standards. This type of flexibility has allowed us to economically meet all our business requirements, especially the most difficult challenges of performance and scalability.

INTRODUCING THE HUMAN REVIEW COMPONENT

Automated compliance reporting provides a very limited view of accessibility compliance status. It cannot report the actual level of compliance. For instance, with strictly automated checking, it is not possible to make a claim like “90 percent of our Web pages are fully compliant with accessibility standards.” As explained earlier, it is an extremely useful tool, especially when it comes to monitoring progress in the first phases of compliance efforts because it can identify sources of problems, but it is not capable of identifying pages where there are no problems at all.

More than half of the provisions included in most standards sets, for example, the U.S. Section 508 standard[1] or the W3C WCAG guidelines, require human judgment to evaluate. In addition, among the violations that are machine-detectable, the absence of a violation does not equate to compliance. For example, the lack of an alt text error does not mean that alt text standards are met. This is because the alt text standards require that alt text be appropriate, a matter of human judgment that is beyond the capability of any existing or potential software.

■ Access keys help not only users who are blind but also those with mobility impairments who have difficulty using a mouse ■

The total IBM Web space includes nearly 30 million pages. If one person reviewed one page every 5 minutes, the evaluation would be finished in 1200 years. Alternatively, we could pick the top 100,000 pages and try to get it done in a month; for that we would need 53 people working full time.

The human review component (HRC) of WASRP addresses this challenge with nonrandom sampling and statistical inference. The statistical approach included in the solution to this problem is the core of what makes it both feasible and revolutionary.

Fundamentals of the method

If one were to conduct human accessibility reviews of a few hundred randomly selected pages out of several million pages on IBM's Internet presence, the results would not give much useful information about how accessible the remaining millions are. There are simply too many factors, such as different authors or departments, amount of training, or static versus dynamic pages, affecting the accessibility of the pages to expect the random sampling to predict how accessible all the pages are with any useful degree of certainty.

In light of this, how can a few hundred human reviews be made more representative of the remainder of the pages? Is it possible that a method for choosing pages to review exists which would enable the human review results for those pages to predict the accessibility of all the pages? This can be done by leveraging the available knowledge of the entire population of millions of pages that we have from two primary sources: our map of Web pages to the IBM application portfolio and automated checking and mining of information from every Web page in the total population.

First, our mapping of the millions of pages to IBM Web applications (as described in the section "Content ownership identification") derived from our portfolio management system provides a basis from which we can make further meaningful subdivisions of the pages, to generate representative samples. By analogy, demographers divide the U.S. population into smaller groups based on an attribute such as income and then independently sample those groups based on already well-established knowledge of the groups' characteristic behaviors. In the IBM Web space, we must work from available information about each application and sample the application economically in order to leverage our predictions of the overall state of the application's accessibility. We have three categories of available and useful knowledge for each application: the *tree structure* of the applications, the distribution of *machine-detectable accessibility errors* within each application, and a partial understanding of the *HTML structure* of machine-scanned pages.

The division of the millions of pages into individual Web applications is immensely valuable because it provides a starting point for analyzing the tree structure of the Web space. This first division provides a list of top-level pages for each Web site that represents starting points for subdivision into samples of pages that have similar characteristics. Knowledge of the tree structure (top-level pages and subsequent server directory structure) of a set of Web pages is valuable because accessibility errors tend to cluster on pages within the tree structure.[10]

Our second primary source of information about the entire population of millions of Web pages comes from machine-automated checking and mining of structural information from the Web pages. From this we know what machine-detectable accessibility errors are present, and we can also collect some measures of the HTML structure of those pages, for example, counts of graphics, headings, or form elements. Because Web developers who do not build accessible Web pages tend to make both mistakes that are machine-detectable and mistakes that are not machine-detectable, it is reasonable to assume that pages which are similar in their distributions of machine-detectable errors and structure will also be similar in their distribution of errors that cannot be detected automatically. This assumption is the key to reaching a worthwhile level of certainty with a practical level of investment in manpower in conducting human reviews.

By traversing the application tree structure and measuring the similarities of pages with respect to machine-detectable traits, we can mathematically determine sets of related pages for which we expect the distribution of errors that are not machine-detectable to be similar. We can then sample a very small number of pages in each set and have a high degree of certainty that the distribution of errors in the sample is statistically equivalent to the distribution of errors in the set.

Scope of outcomes

The scope of measurements produced by the HRC is defined in terms of the Web space and the granularity of supportable subdivisions of the Web space. There are two Web spaces: Internet and intranet. There are three possible levels of granularity in each Web space: enterprise, business unit, and application. Reporting costs increase with the level of granularity; for example, making claims about each application would require that every application be sampled.

Until we have a working prototype of the HRC process, it is not possible to know what level of granularity can be supported within the financial and cycle-time constraints under which the production version of the process will operate. We believe it will be practical to create reports that support statements such as “X percent of Internet content and Y percent of intranet content owned by business unit U is compliant.” In addition, the HRC measurements will also provide data describing the degree of noncompliance, for example, “Typical pages owned by business unit U have N severe problems.”

We can estimate the cost for the most detailed and most expensive reports as follows. The process is designed to yield an average review time of five minutes per page. The 3.5 million pages of Internet Web space are divided into approximately 200 applications. With experimental modeling, we

■ When we attempted to go from a 3.5-million page capacity to a 30-million page capacity, we learned what scalability really meant ■

estimate the average number of pages (p) that will be required from each sample, and the average number of samples (s) required from each application, in order to support an acceptable degree of certainty. This results in a period of time equal to ($p \cdot s \cdot 200$ reviews)$\cdot$(5/60) hours to review the Internet Web space. The intranet Web space is divided into a similar number of applications. The values for p and s, however, may be different for the latter Web space.

Process description

The HRC process has three basic parts. First, URLs for sample page sets are extracted from the automated-compliance-reporting data repository. Second, the sample of URLs undergoes human review for accessibility errors. Finally, the resulting data is merged with the data from the automated-compliance-reporting process to yield a complete report. The sample set, extraction, and merge algorithms depend on the statistical model of the Web space, which we now describe.

The *error profile* of a Web page is a numeric representation of all the accessibility errors on that page. Two common methods for defining an error profile are a pair (number of priority-one errors, number of priority-two errors) and a set {number of *type-one* errors, number of *type-two* errors, . . . , number of *type-n* errors}, where priority is a categorization by severity of error and *type one, type two, . . . type n* are tests for a specific type of error. (The concept of an error profile can be extended to include structural similarity by adding components that are counts of structural elements.) A group of pages has an average error profile. The Web-space model makes it possible to predict the average profile of non-machine-detectable errors for a population based on errors found during human review of a sample of the population.

The application that extracts the sample URLs from the data repository for the automated-compliance-reporting process is called the Web-space modeler (WSM). The first task of the WSM is to divide the total population of URLs in the Web space into subpopulations to be sampled (see *Figure 2*). It does this by the use of two principles. First, similar problems are likely to be found in similar locations within an application. Second, differences between error profiles suggest the boundaries of page clusters (subpopulations) within the application. The WSM uses the errors found by automated compliance checking to create error profiles for every page in the total population.

Assuming we can afford the finest level of granularity, each application will contain at least one subpopulation. In this case, the WSM will examine all the pages in each application to determine if the pages in that application should be divided into multiple subpopulations. The WSM starts with the top-level page of an application that was identified during the ownership identification process (see the section "Content ownership identification") of the WASRP. It first compares the error profiles of pages in the same directory as the top-level page, grouping pages with sufficiently similar error profiles together in a subpopulation. As the attention of the WSM passes from the "roots" to the "leaves" of the directory tree, pages are assigned to existing or new subpopulations sequentially, starting with pages in directories that are closest to the directory of the top-level page. If the unassigned URLs have error profiles similar to those of their nearest neighbor, they are assigned to the subpopulation to which the neighbor belongs. Otherwise, a new subpopulation is created to contain the dissimilar pages.

After the WSM assigns the pages of an application to subpopulations, it selects a sample set of pages within each subpopulation for human review. When human reviewers find errors on pages, those errors are assumed to be characteristic of pages within the subpopulation they represent. The number of pages selected for human review depends on the number of subpopulations. Because labor is expensive compared to automatic review, there is a motivation to keep the number of subpopulations low. The number of subpopulations depends largely on how similar we wish error profiles for clustered URLs to be. There is also the possibility, if the number of subpopulations is low compared to the amount of time available for human review, that we will be able to oversample from subpopulations that happen to contain URLs with relatively diverse error profiles.

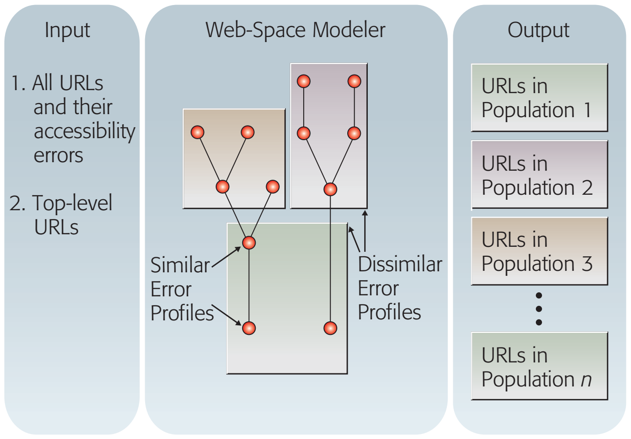


Figure 2
The Web-space modeler

Human page review procedure

This section describes the steps necessary for the human review procedure.

Choosing an error profiling method

The type of error profile we choose to generate with the human review procedure determines the accuracy and efficiency of the procedure, which is very important for containing expense.

An example of an inexpensive profile is an "excellent-good-fair-poor" scale in which "excellent" means the page has no errors of any kind, "good" indicates one error, "fair" indicates two to four errors, and "poor" indicates five or more errors. Using this profile, it is likely that many pages would be rated as "poor" after evaluation of only one or two criteria, and all evaluation of remaining criteria could be skipped. If knowledge of which types of errors are most common is used to order the criteria, this could result in a very efficient procedure.

A downside to such a simple profile is that it does not include any means for weighting the importance of a criterion and that it does not allow a great deal of distinction among applications; for example, two applications could be rated as fair, but one might be much worse than the other. This reduces the value of the data to managers who may be prioritizing remediation expenses.

Some accessibility errors are more serious than others. For example, missing or misleading alt text errors on an image link or an image button are serious errors and roughly of equal importance. Those errors on active images are more serious than missing alt text on an informational image. Finally, missing alt text on formatting images is of the least importance.

One way to score the results of the human review is a list of error counts: the number of severe errors, moderate errors, and minor errors. This is related to, but not the same as, the priority distribution of errors employed in the automatic review. In particular, missing alt text on a formatting image is a priority-one error in the automatic review but it is viewed here as a minor error. We call this a "severity error profile."

The relationship between human-review error reporting and automatic-review error reporting is important. We are projecting the findings about errors in a small subset of the whole population. The more homogeneous the population is, the better that prediction will be. In a population where half the pages have 10 priority-one errors and half have 10 priority-two errors, if error reporting takes account only of error count, the whole population is seen as homogeneous. On the other hand, if the error profile is taken to be a pair of numbers, priority-one errors and priority-two errors, then the populations are almost as different as possible, as measured by the error similarity calculations presented in the section "Error similarity and average error profile calculations."

There is an interesting Web accessibility competition called the Accessibility Internet Rally,[11] which is staged in cities around the United States. For the rally, Web design and development teams are trained in accessibility issues and then paired with nonprofit organizations for whom they build Web sites in a one-day rally. Volunteer judges then judge the sites, and the winning teams are announced at a gala event. It is important there too to limit the amount of time required for judging. That process[12] is similar to the severity error profile described earlier except that at most three errors are counted in each category. With no errors in a given category (e.g., severe errors), the team gets zero points; one error yields five points; two errors yield nine points, and three errors results in the maximum score which is 10 points. Of course, high scores are bad. This "capped" severity error profile has the benefit that the judge or human reviewer can just stop counting or analyzing once three errors in a given category are found.

Our human review procedure collects data that can be formulated in several ways; obviously the error count could always be used, but we can also obtain a count of errors by error type or by error severity.

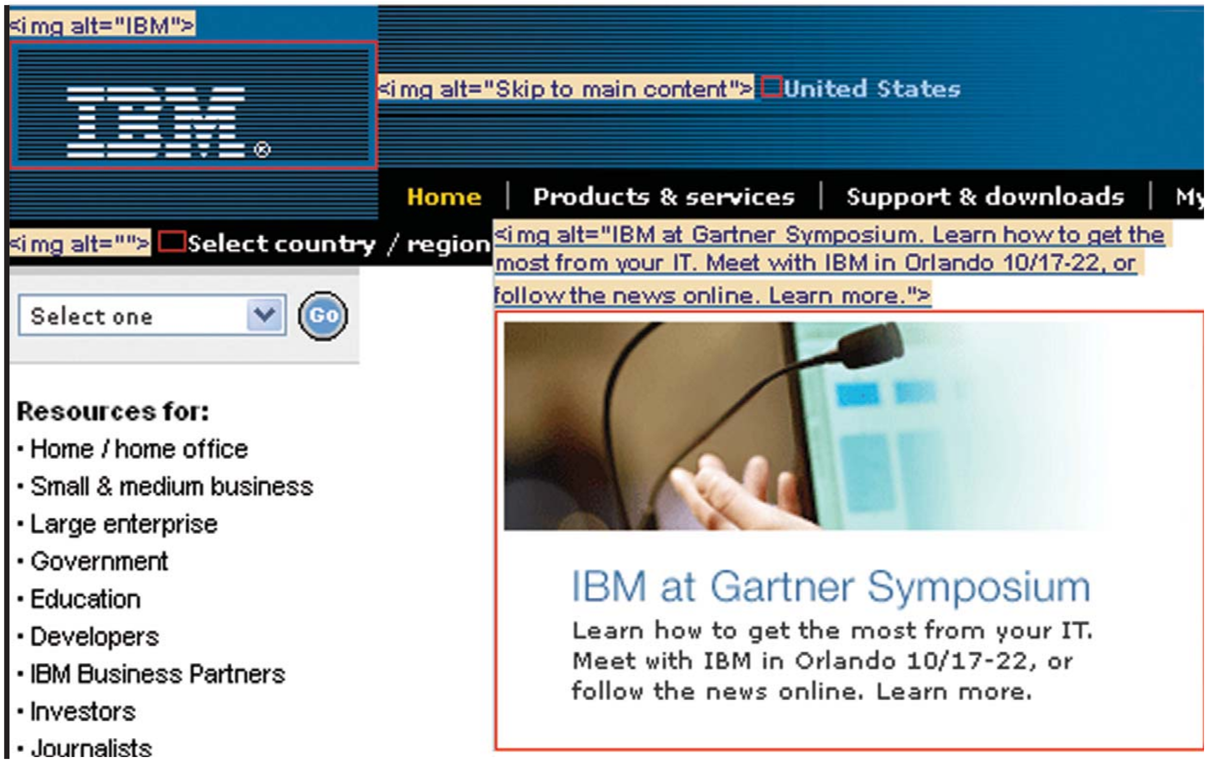
Procedure outline

In designing the human review process, we were confronted with two major requirements that seemed to be at odds with each other. On the one hand, we wanted to minimize the time required to complete the human review on each page; on the other hand, we wanted to maximize the likelihood of finding errors if errors were present.

The tools we use for human review are crucial in order to meet time requirements. We use "favelets" that modify the visible page by highlighting constructs (borders around images) and adding text to the page (the alt text on the images). *Figure 3* shows a sample of part of the IBM Home Page with a favelet having highlighted active images and displaying their alt text.

Favelets (also called "bookmarklets") have been around for a long time,[13] but only recently has the idea been applied to accessibility in the work of the Accessible Information Services of the National Information Library Service of Australia.[14] Favelets consist of JavaScript** code that is associated with a favorite link in a browser. When the favelet is activated, the current page can be modified.

As seen in the figure, each image has alt text corresponding to the text on the image, with two exceptions. The "skip to main content" link is an invisible image, and there is another image just below the IBM logo that is suspicious. Active images should never have empty alt text (i.e., `alt=""`), which this one does, unless there is text in the containing anchor that specifies the target of the link. In this case, the tester can move the mouse from the rectangle that has `alt=""` to the link text



<img alt="IBM">
<img alt="Skip to main content"> United States
Home | Products & services | Support & downloads | My
<img alt=""> Select country / region
Select one
Go

<img alt="IBM at Gartner Symposium. Learn how to get the most from your IT. Meet with IBM in Orlando 10/17-22, or follow the news online. Learn more.">

Resources for:
- Home / home office
- Small & medium business
- Large enterprise
- Government
- Education
- Developers
- IBM Business Partners
- Investors
- Journalists

IBM at Gartner Symposium

Learn how to get the most from your IT. Meet with IBM in Orlando 10/17-22, or follow the news online. Learn more.

Figure 3
Example of a favelet showing alt text for images

"select country/region" and determine that the image is in fact inside the anchor and the `alt=" "` text is acceptable.

With favelets designed specifically for our human review process—leaving to automatic review what can be done automatically—the human review process can be streamlined significantly. The following is a brief overview of the steps of the IBM human review process.

1. Begin with a favelet that alerts the reviewer to multimedia and audio-file types that require captions or text transcripts and that must be examined.
2. Use a favelet to highlight active images (image links, image map areas, inputs of type image) and their alt text to be checked for adequacy. Here and in the other steps of the human review process, the actual errors, like having no alt text at all, are not reviewed because they will have been picked up in the automatic review process. The favelet alerts the reviewer to, for example, "1 alt text error and 5 active images to review."
3. Use a favelet to highlight formatting images for which the alt text needs to be checked.
4. Run a favelet on the page to highlight larger images and their alt text to be reviewed. The reviewer needs to check those large images for charts, graphs, or screen shots that may require a long description; then a favelet is applied to the page to examine the long descriptions and determine adequacy.
5. Apply a favelet to determine the number of form controls for which the labeling needs to be reviewed and highlight labels and titles.
6. Examine the page to see if data tables are present, and if so, use a favelet to highlight the table markup; check that the markup meets accessibility requirements.
7. Test the skip link and record an error if it is not present or if it does not work with the keyboard.

There are additional tests that can be applied and used only if certain data items are present on the sampled page. For example, the automatic test could look for plug-ins or applets, and if found, trigger a human review of those objects for accessibility. We are not including these tests in the first version of the human review component.

Error similarity and average error profile calculations

In the preceding section, we discussed how the WSM uses the difference between two error profiles to determine whether the Web pages they represent belong to the same subpopulation. The measurement of the differences between two error profiles is called the *error similarity*. There are a variety of ways to calculate error similarity. The options we are employing in our prototype models are described in the following subsections.

N-space error similarities

Similarity between the accessibility errors of a pair of pages is represented by a single number. As errors are often qualitatively different (e.g., alt text errors and labeling errors), in n-space error similarity, the different types are considered as n independent dimensions that describe a multidimensional space. The numbers of errors of a particular type are represented as values along the dimension representing that error type. Thus, vectors within the error space represent each URL's error report, and similarity between pairs of error reports is described as differences between vector pairs. Two metrics of error similarity are used, and their usefulness to represent variance in error similarity is discussed in the next subsection.

Euclidean error similarity

The similarity of the error reports of two pages is set as the distance between the endpoints of their error vectors. The more similar the error profiles of two pages are, the smaller this distance is. If two pages have the same kinds and numbers of each error type, their corresponding vectors are the same, and the similarity is zero.

Cosine error similarity

Cosine similarity deemphasizes the importance of the number of errors on similarity and focuses more on error type. The cosine similarity of vectors is the cosine of the angle between the two vectors. If the error vector of one page is a multiple of the error report of another page, then the similarity is maximal.

Average error profile

This profile is calculated as the average number of errors of each type for each URL within a subpopulation or directory.

■ We store, on average, 1.5 million documents a day in the data repository ■

Sampling methods

In the interest of finding an efficient and accurate sampling method, we compare three different methods of selecting URLs for human review.

In the first method, *random sampling*, the population set is the unit of interest, that is, the application or business unit. Random sampling is easily accomplished but may be inaccurate. In the second method, subpopulations are defined as the top-level URL and the URLs behind it. Only the top-level URLs undergo human review. This method has the clear advantage of being easily accomplished, and its drawback is that it is not clear that the top-level URLs will be representative of other URLs within an application. The third sampling method defines populations based on folder structure and error similarity, as described in the section, "Process description." The accuracy of these three sampling schemes can be evaluated through simulation.

Calculating sample size for a population set

The number of individual URLs selected for human review is determined according to the mean error similarity within each subpopulation. We assume that populations with a greater mean error similarity (i.e., URLs that are very divergent in error profiles), as indicated through WASRP, are also very divergent in types of errors as assessed by HRC (see the section "Web-space model assumptions"). Thus, HRC sampling is more extensive for subpopulations with high mean error similarity. We assume that proportioning sampling in this way allows more accurate prediction of the fraction of pages that fail to meet Web accessibility standards.

In the overview of the process of forming subpopulations in the section "Process description," we described how error similarity determines the way to cluster nearby subpopulations of URLs. We use a threshold for determining the degree of similarity sufficient to split or combine folders. This threshold, named *theta*, needs to be "tuned" in order to satisfy prediction accuracy and the amount of time devoted to conducting reviews.

Making assertions about application accessibility

We wish to make assertions about the population based upon both human and automated reviews. WASRP gives us reports of errors for every URL, but for errors that can be found only through human review, we must generalize from our HRC sample to the entire population. By combining our automated data with the inferences we generate from human review, we are able to say, with a good degree of confidence, that *x* percent of the pages in a population are 100 percent compliant with accessibility requirements.

Web-space model assumptions

IWSC and HRC error similarity must co-vary so that subpopulation formation, based on IWSC, is relevant for HRC. Data from the IWSC automated review is used to split applications into discrete populations for HRC sampling; thus the errors that are found by automated and human accessibility review must be congruent. The congruence only need be in the similarity of accessibility errors that are found in neighboring pages. It is assumed that if errors found among pages in a population by automated review are diverse, then the errors found by manual review will also be diverse. This assumption can be tested by conducting human reviews on 100 percent of the pages for a representative set of subpopulations and comparing those results to results made following the proposed sampling methods.

Measurement of error similarity should be valid, accurately representing similarity of errors in a way that could reveal real-world relations among pages, such as use of similar templates or creation by the same author. Because the forming of populations also accounts for tree structure, this assumption is not as vital as the first; that is, if two pages that belong to different business units and that were created by different authors have a high error similarity by chance, they will not be assumed to be in the same population.

CONCLUDING REMARKS

The process of ensuring accessibility of a large Web presence is bounded by many constraints. Variability of implementation stems from differing production techniques over time and among the many business units within the firm. Automated analysis is effective for only a subset of the required features, leaving the remaining features to undergo human inspection. Time and cost realities lead us to a multifaceted approach.

Templates are very beneficial for providing a high level of compliance for new documents, especially in the area of navigation features. A robust education program helps developers and content producers understand how to ensure standards compliance for the content they add to the templates.

Automated techniques crawl and analyze machine-testable features of the entire inventory, reporting on areas needing remediation. The most daunting part of compliance assurance, human review, can be efficiently guided by statistical sampling processes that optimally segment the Web space for compliance-level assertions.

The combination of these techniques, repeated at a regular frequency, identifies problems for remediation. No single approach can achieve the effectiveness and cost efficiency of the combination of these techniques. The use of these techniques results in a Web presence that is more accessible and more easily used by a significant population of people with various disabilities.

ACKNOWLEDGMENTS

Human review, typically including analysis of source code, has long been a staple of Web accessibility evaluation. We are grateful to the people of Accessible Information Services of the National Information and Library Services of Australia. Their ideas concerning using favelets for human review of Web pages are key to the success of this project. We are grateful to John Slatin and Kay Lewis for their input.

*Trademark, service mark, or registered trademark of International Business Machines Corporation.

**Trademark, service mark, or registered trademark of Massachusetts Institute of Technology, Watchfire Corporation, The Perl Foundation, Parasoft, or Sun Microsystems, Inc.

CITED REFERENCES AND NOTES

1. *Section 508 of the Rehabilitation Act: Electronic and Information Technology Accessibility Standard*, U.S. Access Board (2000), http://www.access-board.gov/508.htm.
2. D. M. West, *State and Federal E-Government in the United States* (2003), http://www.insidepolitics.org/egovt03us.html.
3. *Web Content Accessibility Guidelines, Version 1.0,* World Wide Web Consortium, W3C Recommendation (May 5, 1999), http://www.w3.org/TR/WCAG10/.
4. IBM Accessibility Center Web site, http://www.ibm.com/able.
5. Screen Reader Visibility Web site, http://css-discuss.incutio.com/?page=screenreadervisibility.
6. *Building In Universal Accessibility + Checklist,* CabinetOffice (May 2002), http://www.cabinetoffice.gov.uk/e-government/resources/handbook/html/2-4.asp.
7. D. Featherstone, "More Reasons Why We Don't Use Accesskeys," Web posting (December 6, 2003), http://www.wats.ca/articles/accesskeyconflicts/37.
8. D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien, "How to Build a WebFountain: An Architecture for Very Large-Scale Text Analytics," *IBM Systems Journal* **43**, No. 1, 64–77 (2004).
9. Bobby was originally developed by the Center for Applied Special Technologies (http://www.cast.org) and later purchased by Watchfire Corporation.
10. P. Bronstad and J. Slatin, "Using Web Site Interconnectivity to Find Clusters of Accessibility Problems," *Proceedings of the Conference on Technology and Persons with Disabilities* (2004), http://www.csun.edu/cod/conf/2004/proceedings/147.htm.
11. AIR-Austin Web site, http://www.knowbility.org/air-austin/.
12. J. M. Slatin and S. Rush, *Maximum Accessibility, Making Your Web Sites More Usable for Everyone,* Addison-Wesley, Boston, MA (2003).
13. G. R. Notess, "Bookmarklets, Favelets, and Keymarks: Shortcuts Galore," *Online* **27**, No. 4 (July 2003), http://www.infotoday.com/online/jul03/OnTheNet.shtml.
14. *Web Accessibility Tool Bar,* National Information and Library Services (NILS), Accessible Information Solutions (2004), http://www.nils.org.au/ais/web/resources/toolbar/index.html.

Accepted for publication February 15, 2005.
Published online August 11, 2005.

Matthew King
IBM Corporate Headquarters, 90 South Cascade Ave., Suite 800, Colorado Springs, CO 80903, (mattking@us.ibm.com). Mr. King is a staff engineer in IBM's corporate IT organization. Since 1998, he has been directing efforts to ensure the accessibility of all IT infrastructure and tools used in IBM worldwide. His interest in accessibility extends beyond his 15-year IBM career as he is blind and has been using and working on assistive technologies since the 1980s while attending the University of Notre Dame. In 1989 he received a B.S. degree with majors in electrical engineering and music and started at IBM as a manufacturing engineer.

James W. Thatcher
Accessibility Consulting, 800 Double Bend Back Road, Austin, TX 78746, (jim@jimthatcher.com). Dr. Thatcher retired in 2000 after 37 years in IBM Research. He received his Ph.D. degree in computer science from the University of Michigan in 1963. In the early 1980s, after 20 years of research in theoretical computer science, he developed one of the first screen access systems which, in 1986, became IBM Screen Reader for DOS. He then led the development of the first screen reader for a graphical user interface, IBM Screen Reader/2. In 1996 he joined the IBM Accessibility Center and led the development of the IBM Accessibility Guidelines. He was vice-chair of the Advisory Committee impanelled by the Access Board to propose accessibility standards for Section 508.

Philip Matthew Bronstad
Brandeis University, Department of Psychology, MS 062 PO Box 549110, Waltham, MA 02454, (bronstad@brandeis.edu). Dr. Bronstad is a postdoctoral fellow at Brandeis University. He received his Ph.D. degree in quantitative psychology at the University of Texas at Austin in 2004, where he worked in the psychology department and at the Institute for Technology and Learning. Dr. Bronstad studies face perception and human-computer interaction.

Robert Easton
IBM Thomas J. Watson Research Center, PO Box 218, Yorktown Heights, New York 10598 (bob_easton@us.ibm.com). Mr. Easton is a senior Internet analyst in the Information Technology department at the Watson Research Center. He manages the Collaborative Computing group that focuses on the integration of collaborative technology into IBM's computing infrastructure. His current research interests include improving accessibility techniques for Internet-based applications. Mr. Easton received an M.S. degree in management of technology from Polytechnic University in Brooklyn, NY. ■