

Arboretum and Arbility: Improving Web Accessibility Through a Shared Browsing Architecture

Steve Oney^{1,2}, Alan Lundgard², Rebecca Krosnick², Michael Nebeling^{1,2}, Walter S. Lasecki^{2,1}

¹School of Information, University of Michigan

²Computer Science & Engineering, University of Michigan
{sony, arlu, rkros, nebeling, wlasecki}@umich.edu

ABSTRACT

Many web pages developed today require navigation by visual interaction—seeing, hovering, pointing, clicking, and dragging with the mouse over dynamic page content. These forms of interaction are increasingly popular as developer trends have moved from static, linearly structured pages to dynamic, interactive pages. However, they are also often inaccessible to blind web users who tend to rely on keyboard-based screen readers to navigate the web. Despite existing web accessibility standards, engineering web pages to be equally accessible via both keyboard and visuomotor mouse-based interactions is often not a priority for developers. Improving access to this kind of visual, interactive web content has been a long-standing goal of HCI researchers, but the obstacles have exceeded the many proposed solutions: promoting developer best practices, automatically generating accessible versions of existing web pages, and sighted-guides, such as screen and cursor-sharing, which tend to diminish the end user’s agency and privacy. In this paper, we present a collaborative approach to helping blind web users overcome inaccessible parts of existing web pages. We introduce *Arboretum*, a new architecture that enables any web user to seamlessly hand off controlled parts of their browsing session to remote users, while maintaining control over the interface via a “propose and accept/reject” mechanism. We illustrate the benefit of *Arboretum* by using it to implement *Arbility*, a browser that allows blind users to hand off targeted visual interaction tasks to remote crowd workers without forfeiting agency. We evaluate the entire system in a study with nine blind web users, showing that *Arbility* allows blind users to access web content that was previously inaccessible via a screen reader alone.

Author Keywords

Accessibility; Web Accessibility; Non-visual Access; Blind; Web Interfaces; Remote Collaboration; Crowdsourcing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

UIST 2018, October 14–17, 2018, Berlin, Germany

© 2018 ACM. ISBN 978-1-4503-5948-1/18/10...\$15.00

DOI: <http://dx.doi.org/10.1145/3242587.3242649>

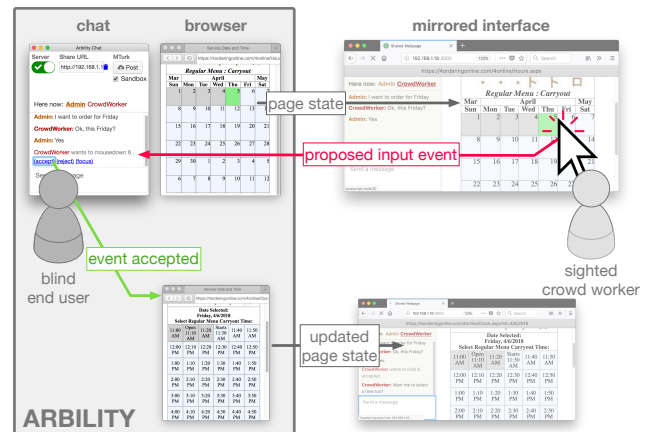


Figure 1. Arbility allows blind end users to interact with web page elements that are otherwise inaccessible via a screen reader. Here a blind end user would like to place a food order a day in advance, but the restaurant’s calendar web page is not accessible because it requires interacting with elements that are not keyboard-focusable and the interaction is listening for a mousedown event (rather than click), which not every screen reader application fires. Arbility allows the end user to hand off this targeted visual interaction task to a sighted crowd worker via the chat panel. The crowd worker interacts with the calendar page to select the end user’s desired order date, and Arbility sends the worker’s proposed action to the end user, who optionally accepts or rejects it. Throughout the task, the crowd worker interacts with a mirrored version of the end user’s web page.

INTRODUCTION

The World Wide Web (web) is a crucial resource for connecting people with services, information, and other people. For more than 39 million [52] blind people worldwide, however, many parts of the web are off-limits [15]. Most blind web users rely on keyboard navigation and screen readers, which convert textual web content into an accessible format (typically speech or Braille) [18]. However, many websites are designed for visual interaction; performing a task requires seeing, clicking, or dragging over dynamic visual content. Seemingly innocuous design decisions, like conveying information visually (e.g., color coding), not including Accessible Rich Internet Application (ARIA) labels, and requiring certain types of mouse-based interaction, can make sites difficult or even impossible for blind people to use [15, 44].

Engineering for accessibility is challenging, and improving access to web content for blind users is a long-standing prob-

lem in HCI. Researchers have proposed automated techniques to help guide blind users, such as enabling Natural Language (NL) control of browsing tasks [3] or automatically generating text labels [32]. However, accessibility issues are too varied to be fully solved by automated tools [15]. Similarly, systems that rely on user-created macros require them to be made in advance, limiting their usefulness for new or personalized tasks [14, 45]. The most reliable way to overcome accessibility barriers is with help from a sighted user for targeted portions of the task, but in-person help from sighted friends or coworkers is neither always available nor desirable.

Seeking assistance from sighted users for such targeted web tasks would normally be squarely in the realm of crowdsourcing [59] and hybrid-intelligence [9, 40] tools, which have both addressed similar accessibility problems [12, 20, 42, 65]. However, crowdsourcing is currently not feasible for web tasks because it is difficult to share browsing state and safely give controlled access to remote crowd workers. For example, sharing a link by copying and pasting does not capture the page state or personalized pages (such as tasks that involve logging in at some stage), while remote access tools like VNC require giving “all or nothing” screen or cursor control to the remote user.

In order to make controlled, stateful sharing possible, we introduce **Arboretum**, a new shared web architecture that makes it possible to seamlessly hand off controlled access to nearly any web browsing session and state. This would, for example, allow blind users to share their browsing context with a remote sighted user to ask a question about a visual element. Many accessibility barriers involve performing visiomotor tasks on the problematic page. For example, some information on a page might only be revealed after the end user moves their mouse over an element.

Toward this end, we also introduce **Arbility**, a web browser that builds on Arboretum to allow blind users to hand off targeted visual interaction tasks (tasks that involve seeing or spatial interaction) to crowd workers. When remote crowd workers join an Arbility browsing session, they can see the end user’s exact context and can communicate with them in natural language through a chat interface. Arbility allows crowd workers to “propose” actions to the end user by demonstrating them, as Figure 1 illustrates. For example, a crowd worker can propose to `mouseover` a menu element by simply moving their mouse over the element on the mirrored page. End users can also request ARIA labels from crowd workers for particular elements. Further, by storing past accepted actions, Arbility allows end users to re-use the page labels and actions proposed by crowd workers the next time they visit that page.

We make the following contributions with this work:

1. **Arboretum**, a shared web browser architecture for creating applications that can seamlessly hand off browsing state to remote users.
2. **Arbility**, a web browser that uses the Arboretum architecture to allow blind end users to request help from crowd workers for targeted visual interaction tasks. Arbility contains several novel features to enable effective communica-

tion between the blind end user and remote crowd workers. These features include allowing crowd workers to propose page actions, the ability to reference page content in chat messages from crowd workers, a feature that allows crowd workers to label unlabeled page elements, and allowing blind end users to re-use labels and actions generated by crowd workers.

3. An evaluation with nine blind participants on three inaccessible web pages showing that Arbility enables them to leverage crowdsourcing to perform tasks on these pages that would have otherwise been difficult, if not impossible. This evaluation also shed light on important design challenges for future work to address — most notably in dealing with privacy concerns.

We distinguish between the Arboretum architecture and the Arbility tool because Arboretum has many potential applications outside of accessibility, as we will describe. Arboretum opens up many exciting opportunities for applying crowdsourcing techniques to web tasks. Arboretum is publicly available as an extensible open source platform¹.

RELATED WORK

Arbility and Arboretum build on research from three vibrant areas: *multi-user/multi-device web browsing*, *end-user web scripting and automation*, and *crowdsourced control of interfaces*, with application to *web accessibility* for blind and low-vision end users.

Multi-User/Multi-Device Web Browsing

Early work by Greenberg and Roseman [24] explored ways of extending web browsers with groupware features to support co-browsing based on synchronized document views and telepointers. Researchers have also studied specific co-browsing interfaces for common web activities, including web search in both co-located [4] and remote [49] settings. A common approach is to implement “master-slave” functionality in which all interactions of one user who controls a session are mirrored for other users who are forced to follow along. Surfly [63] is a modern implementation of this in the form of a co-browsing web service combined with a discussion interface. Heinrich et al. also showed how elements of generic single-user web pages can be automatically converted to shared applications [28], with a focus on making editable text boxes sharable. Another common approach is to allow users to use a divide-and-conquer strategy by splitting up web pages and focusing their work on parts of the collaborative web activity. WebSplitter [26] was an early system that could split a web page among multiple users and devices. Research has then extensively studied sequential and parallel web browsing on multiple devices via multibrowsing support [33] and migratory interfaces [8] that allow users to easily switch and transfer (parts of) web tasks between devices. Apple’s Continuity features, such as Handoff [6], are modern implementations of this on Mac OS and iOS devices. More recent systems such as MultiMasher [30] and Webstrates [36] provide architectural support and visual tools for “mashing up” and re-authoring existing web applications for

¹<https://github.com/sonney/arboretum>

a wide variety of multi-user/multi-device shared web browsing scenarios. Finally, Subspace [61], PolyChrome [7], XD-Browser [50], and others [54, 51] can distribute web pages between devices while keeping the view and input states synchronized between multiple browser nodes. However, previous work does not enable *controlled* hand-offs of third-party content, as Arboretum does.

Synchronous interaction can greatly improve user satisfaction during customer service interactions as well. As Forrester finds in a 2011 study [34], “Live-assist communication channels (phone, chat, cobrowse) have much higher satisfaction ratings than asynchronous electronic channels (email, web self-service).” They found satisfaction ratings of: “phone (74%), chat (69%), cobrowse (78%), email (54%), and web self-service (47%).” Their highest satisfaction ratings were seen with “cobrowsing” (e.g., <https://www.olark.com/help/cobrowsing>), which is conceptually similar to our approach, but *highly* specialized to individual web sites, requiring that web developers use proprietary frameworks in their implementation. In contrast, Arboretum works on any website without any special accommodations from site developers. Users simply access the web using Arboretum as they would through their regular web browser.

Web Accessibility Standards and Solutions

People with disabilities, such as motor or visual impairments, face significant difficulties accessing the web when compared to most other users because of the web’s reliance on visual layout and small interaction targets (e.g., in-text URLs and drop-down menus). Existing access technology does not provide an equivalent web browsing experience. Screen readers convert textual content to speech for visually impaired users [11], but are tedious to use because users are often-times forced to traverse the Document Object Model (DOM) linearly, one element at a time. Blind end users might use navigational shortcuts (such as locating content on a web page using Ctrl+F or quickly scrolling through the heading levels of the DOM), but such strategies must be variously deployed, since no single strategy has any guarantee of success.

Given the diversity of web development paradigms and the Web 2.0 trend toward dynamic and interactive content, a website’s DOM is by no means structured to be parsed linearly by end users. In response to these trends toward visually dynamic—and hence, inaccessible—content, the World Wide Web Consortium (W3C) has developed and encouraged the use of standards for an accessible web, known as Web Content Accessibility Guidelines (WCAG), as well as standards for making rich, dynamic page content more easily parsed by a screen reader, ARIA. However, these guidelines have yet to be adopted as standard practice among many developer communities, and they are not retroactively applicable to websites that no longer have active developers, such as those of local stores, restaurants, or community centers [19].

Furthermore, even if a website does comply with WCAG standards, it is not guaranteed to deliver a satisfying user experience, and may still contain obstacles if certain expected information is missing [2]. In such cases, it may not be clear to end users whether they should try searching linearly through

the entire page, or look for the information on a different page. Recently, Bigham, et al. have called this the problem of “Not Knowing What You Don’t Know” for blind web users [15]. Essentially, not knowing if a particular piece of information is inaccessible via a screen reader, merely challenging to access, or not present on the page at all can lead to time draining searches through the DOM. Perhaps most prominently, Bigham, et al. have proposed a variety of solutions to the problem of web accessibility for blind end users, including a scripting frameworks for developers and users to collaborating improve accessibility [10, 14], real-time on-demand captions of images by remote crowd workers [13, 12, 66, 41], and screen readers designed to be accessible on-the-go [25, 16]. These solutions target important accessibility problems—lack of developer expertise in building accessible websites, lack of ways to get around barriers caused by visual information, and lack of access to keyboard-navigable screen readers—and inform our design goals for Arbility.

Crowdsourced User Interface Control

Arbility expands on previous research investigating the use of crowds to control existing user interfaces, often as solutions to accessibility challenges currently beyond the state-of-the-art of automated methods. Using a remote desktop access tool like VNC provides full access to and control of the target machine, meaning that it requires having access to a fully-trusted party as the remote user — a significant limiting factor in the availability of any such system.

Legion [40] mitigates this problem by filtering out potentially bad actors by requiring consensus between multiple crowd workers who click to control an interface. Legion makes aggregated control possible, but only captures mouse clicks and key presses, which limits the kinds of actions that crowd workers can take. For example, it is not possible to propose `mouseover` events or to scroll to a different part of the page. By letting the end user be the leader, Legion was successfully used for Programming by Demonstration (PBD) applications creating macros for Google spreadsheets and creating mash-ups and controlling existing desktop applications with the crowd. Further, it is not possible to replay crowd workers’ previous interactions using only pixel/coordinate information without any semantic information, unless subsequent browsing sessions have the exact same page state and window configuration (location and dimensions, scroll position, etc.).

Salisbury et al. [60] and Loparev et al. [47] explored alternative real-time mediation strategies for integrating the input of multiple crowd workers on a control task. Researchers have experimented with asking crowd workers to recognize interaction patterns from the users’ completion of a range of different web browsing tasks [39]. Arbility builds on this research by enabling the hand-off of web browsing tasks so workers can complete these tasks on the end user’s behalf.

Web Automation and Scripting

Arbility includes a PBD component that records and can replay actions that remote crowd workers take. Arbility is one of several PBD web activity recording and automation tools, such as WebVCR [5], ActionShot [46], PLOW [3],

and CoCo [43]. Extensive reviews of existing PBD systems can be found in [21, 58]. There are also several non-PBD web automation tools. Chickenfoot [17] allows users to script browser automation using a high-level programming language. Smart Bookmarks [29] can be generated that are essentially replay scripts of web browsing sessions to restore an entire bookmarked session state. Inky [48] allows users to interact with the web via a relaxed command-line interface. However, prior web automation systems require that macros be created (either through programming or by demonstration) *before* the end user can perform the task, meaning they cannot be used when users encounter new accessibility barriers.

SYSTEM DESIGN & FEATURES

We divide our discussion of Arbility into two sections: our design goals and the resulting design.

Design Goals

The design of Arbility was influenced by prior studies on the types of challenges that blind web users face, a set of guiding principles grounded in user-centered design, and feedback from pilot studies.

Challenges Blind Users Face when Using the Web

Researchers have studied and categorized the types of accessibility barriers that blind users face on inaccessible web sites [56, 18, 57, 15]. Broadly, there are three primary types of accessibility barriers that we designed Arbility to address:

- **Barriers caused by visual information.** Many websites lack ARIA labels, convey information in images, or embed information in visual style. These types of mistakes can occur even on websites that are otherwise usable and accessible [56]. For example, a restaurant might use **red text** to identify spicy items on their menu, [15] or a program guide for an HCI conference might use background images to indicate best paper awards. Both conventions are invisible to screen readers.
- **Known unknowns.** Blind web users who are unable to find a given piece of content on a page cannot be sure if they are unable to find it because the page is inaccessible, or because the content does not exist on the page [15]. This applies even to sites that are completely accessible, as there is no way for users to be certain they have complete information, short of navigating the page’s source code.
- **Lacking keyboard navigability.** Blind users typically rely on keyboard navigation to interact with a page. However, some pages might not be keyboard navigable for three primary reasons. Some sites require mouse interaction because they were programmed to listen to mouse events (press, release, move, etc.) Other sites (including the latest versions of the UIST and CHI program guides) might require interaction on elements that are not typically keyboard-selectable or clickable, such as a generic `<div/>` or heading, respectively. Alternatively, a site might lack keyboard navigability because the information is not structured in a way that is easy to digest (e.g., misleading tab ordering) or from web developers confusing structure with content.

Arbility helps users overcome all three types of barriers.

Guiding Principles

Broadly speaking, the goal of access technology is to increase its users’ independence and agency. Blind users generally place a high value on autonomy [1]. Thus, the first guiding principle of Arbility was to *ensure that the end user retains control* of their browsing session even as remote crowd workers provide assistance. Interactions with Arbility should reflect the fact that the blind end user is the task expert, whose goal is to guide remote helpers through a rote task.

We also wanted to try to ensure that end users could *trust* the actions proposed by remote users. As we will discuss in the future work section, we treat trust as a different design goal than ensuring privacy, which is a feature that Arbility leaves to future work. This means that when an end user receives a proposed action from a crowd worker, they can trust that it will not be nefarious. We address the issue of trust by allowing end users to examine crowd workers’ proposed actions — being able to see what elements they affect and how.

Finally, we wanted Arboretum to *fit users’ existing workflows* — to allow them use their preferred screen reader and navigation methods. We designed Arboretum to allow users to interact with it like any other browser, except that they can also easily toggle a shared browsing session as needed.

Guidance from Pilot Studies

In addition to the above considerations, there were several practical interface design guidelines from pilots of Arbility with a blind web user. These pilots helped ensure that Arbility itself is accessible and usable.

Arbility and Arboretum Features

The resulting design of Arbility is illustrated in Figure 1. Arbility consists of two windows: a *browser window* and an *administrative panel*. The browser window mostly behaves like a standard web browser. End users interact with web content as normal, through a third-party screen reader like JAWS, NVDA, or VoiceOver. When the end user wants to seek help from crowd workers, they use the administrative panel, where they can toggle web session sharing, communicate with crowd workers, remove specific workers from the shared browsing session, or mark a task as completed.

Arbility also embeds a web *server* as part of the browser. This server serves a page that mirrors the DOM state of the end user’s browser, without sharing the underlying code. When remote users visit the served page, it appears to be exactly the same as the page the end user is using, augmented with a chat window panel, where they can interact with the end user. This page mirroring is done through Arboretum, which we will describe in more detail below.

Mirroring Web Pages with Arboretum

Everything that is rendered by a web browser (what end users interact with and see) is specified by the page’s Document Object Model (DOM), a tree structure where every node is an element on the page. Developers write web pages by writing code that creates and manipulates the DOM, using the three fundamental web languages. The *HyperText Markup Language (HTML)* specifies the initial content and structure

of the page’s DOM. *Cascading Style Sheets (CSS)* control the visual appearance of the DOM, such as colors and positions. *JavaScript* defines the page’s behavior by specifying how the DOM should change in reaction to user input and other events.

When a page is shared with a remote user through Arboretum, the page’s DOM and appearance are shared². Unlike the naïve approach of sending a link to remote helpers, sharing the current page’s DOM allows browsing sessions to be shared even if they involve password-protected pages. Another naïve approach to page sharing would be to share the page’s source (HTML, CSS, and JavaScript). However, this would lead to diverging DOM states as the end user and remote helper perform different actions on the page. Instead, Arboretum strips the page’s JavaScript code and propagates DOM changes to remote clients dynamically.

However, in testing Arboretum with external websites, we found that simply sharing the DOM with the JavaScript stripped from it can lead to access issues with some types of external resources, such as images or style sheets, at the original web server’s discretion. Thus, in addition to sharing the DOM, Arboretum also re-routes references to external resources so that they are served directly from the Arboretum server. This ensures that remote workers will be able to see the same content as the end user.

Finally, in order to allow remote users to interact with the page content and with the end user, Arboretum’s web server attaches extra snippets of JavaScript to the pages that it serves up. These extra snippets of JavaScript: 1) add a chat widget to the side of the served webpage that lets workers interact with the end user, 2) modify the remote worker’s DOM to always reflect the DOM content of the end user, and 3) capture the remote worker’s input events and send them back to the end user’s browser, where they can decide how to act on them.

In sum, Arboretum creates a “mirror” DOM tree that is modified to strip out JavaScript that would keep its DOM out of sync with the end user’s, adds code to allow remote workers to communicate back with end users, keeps the DOM of remote users and the end user in sync, and re-routes any external resources to ensure the remote workers see the same content as end users.

Chatting with Remote Crowd Workers

In order to allow end users to effectively convey their goals to remote crowd workers, Arbility includes a text-based chat channel connecting the end user and remote crowd workers. This chat channel remains open throughout the shared browsing session, which makes it easy for crowd workers to ask clarification questions for poorly worded requests. Whenever crowd workers post a new message or join the channel (after choosing a username), Arbility uses audio notifications to notify the end user. End users can also remove crowd workers from the browsing session (by typing `/boot <user>`) or mark a task as successfully completed (`/done`).

²This explanation is slightly simplified — Arboretum can also share important variables that are not technically part of the DOM, such as the value of an `<input/>` element or the visual contents of a `<canvas/>` element.

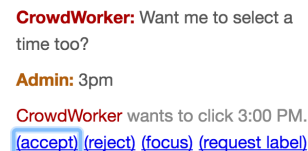


Figure 2. When a remote client worker proposes a page action, a description of that action is sent to the end user for approval. The end user can perform one of four actions: 1) *accept* the action, which will perform it on their browser; 2) *reject* the action; 3) *focus* which will direct their keyboard focus and screen reader to the target element; or 4) *request a label*, which will ask a crowd worker to replace the ARIA label of the target element.

Proposing Page Actions

For information seeking tasks, such as asking a question about the content of an image or whether the page contains a piece of information, the Arbility chat feature combined with Arboretum’s session sharing is sufficient for end users to ask and answer questions. However, many kinds of tasks also require users to *interact* with page elements (e.g., when information is hidden behind collapsible panels or only appears when the cursor is hovering on a page element. Thus, Arbility allows remote users to *propose* actions for the end user to perform. These actions can include any user interaction (e.g., `mouseover`, `touchstart`, etc.).

Retaining Control and Trust for End Users

One of the design goals of Arbility is to give the end user ultimate control over their browser. Thus, rather than allowing crowd workers to directly interact with the end user’s page, any action that a crowd worker proposes must be approved by the end user, as Figure 2 shows. If the end user approves that action, then Arbility emulates the action proposed by the crowd worker on the end user’s browser.

In order to allow the end user to make an informed decision about whether they should accept a proposed action, Arbility automatically generates a textual description of the proposed action. This description includes the type of event (e.g., `mousedown`, `mouseover`, etc.), the event target, and any other relevant information. In order to describe the event target, Arbility uses (in order of precedence): ARIA labels, text content, or tag names. If the end user needs more information about a given element, they can also quickly give the target element keyboard focus in their screen reader via a “focus” shortcut in the chat interface or request a label from remote workers. All of these features are designed to allow the end user to trust that any actions they approve will not have any unintended consequences.

Minimizing the Learning Curve for Crowd Workers

In order to minimize the learning curve for crowd workers, we designed Arbility to allow them to propose page actions in as natural a way as possible — by interacting directly with the mirrored page. Thus, when a crowd worker clicks a button or moves their mouse over a relevant element, Arbility automatically sends an action proposal to the end user.

However, if implemented naïvely, when a crowd worker clicked a button on the page, this feature would fire a series of `mousemove` and `mouseover` events (as the worker

moves their mouse to the target element) and `mousedown`, `mouseup`, and `click` events as the worker is clicking the element. Assuming the end user only cared about the click event, there would be many false positives and erroneous intermediate events. To address this issue, Arbility only proposes events for elements and events that are associated with at least one JavaScript event listener. Workers do not need to understand different event types; when they demonstrate an action on the page, Arbility’s event hooks only listen for events that have associated callbacks. Although this does not fully solve the issue of false positives (web pages might have erroneous event listeners or listeners that could be triggered when the remote user intended to perform another action), it does mitigate it greatly. Remote crowd workers can also delete actions that they did not intend to propose.

Storing and Recalling Previous Actions

After a shared browsing session is complete, Arbility stores the actions that were approved. The next time that user loads the same page, Arbility will offer to repeat these actions on the newly loaded page. As the implementation section below discusses, these new events are re-aligned to be robust with respect to page changes. A list of suggested commands is displayed above the chat panel, as Figure 4.4 shows.

IMPLEMENTATION

Arbility is built as an Electron [31] application that builds on Arboretum, a Node.JS [22] application. Both systems are implemented with the TypeScript programming language and ReactJS (in the case of Arboretum, the ReactJS code implements in the worker-side pages).

As Figure 3 illustrates, Arbility has two components:

- 3A: A chat interface for interacting with crowd workers
- 3B: a chromium browser that the end user interacts with through their preferred screen reader.

Arbility interfaces with Arboretum, which itself has two separate components:

- 3C: A Web Server that serves a dynamic page for remote crowd workers. The page is a transformed version of the contents of the end user’s Chromium browser.
- 3D: A DOM state tracker that interacts with the Arbility Chromium browser through the DevTools Protocol to track and update the DOM state and updates, pull any necessary external resources, and simulate input events from remote workers. This component handles many complexities of document mirroring, including dealing with nested frames, retargeting resources, removing JavaScript, and more.

Communicating via the DevTools Protocol

Arboretum uses the Chrome DevTools Protocol [23] (formerly known as the Remote Debugger Protocol). This protocol gives Arboretum access to the internal state of every DOM element on the end user’s browser. Because it uses the DevTools protocol, Arboretum is robust with respect to internal browser changes and can work with any browser that implements this protocol.

Arbility also uses the DevTools protocol to determine which parts of a page listen to user input events (which in turn determines whether an action from a remote user is ignored or should propose an action on the end user’s page). When the end user “accepts” an action proposed by a remote crowd worker, Arboretum emulates that event on the end user’s machine by injecting the end user’s page with code that simulates the event on their client.

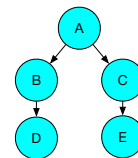
Synchronizing Distributed Clients

Arboretum uses WebSockets to communicate between the end user and remote clients. These WebSockets communicate both chat messages and DOM state changes dynamically. Arboretum also uses ShareDB [62] to synchronize the DOM between the end user and remote crowd workers.

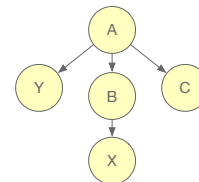
Remembering and Retargeting Prior Page Actions

Whenever an end user accepts a proposed action from a crowd worker, Arbility stores the details of that action (the event type, target, and other necessary details) and a snapshot of the DOM tree when that action was performed in a JSON file on the end user’s browser. However, pages change over time, which can invalidate the stored actions if implemented naïvely.

For example, suppose the end user visits a page that has the following DOM tree, which is shortened and labeled for the sake of simplicity:



and the remote crowd worker proposes an action on node **B**. The next time that user visits the same page, the DOM tree has been modified and now has the following DOM tree:



Arbility—which does not have the benefit of clear labels like those in these diagrams and must work with DOM trees that are significantly larger—must then determine what node is equivalent to node **B** in this new DOM tree. In order to do so, it first flattens both trees using a depth first traversal.

It then computes a “similarity” score between pairs of DOM nodes in the different trees. In our current implementation, nodes that have the same tag name are considered the most similar (+100 in the similarity score). Nodes with similar DOM attribute names and values are also scored highly (+7 per matching name/value pair and –7 for every non-matching name/value pair). Arbility then uses the Needleman-Wunsch sequence alignment algorithm (most frequently used to match DNA sequences) to determine the best mapping between DOM nodes in the new and old trees, with a gap penalty of –2.

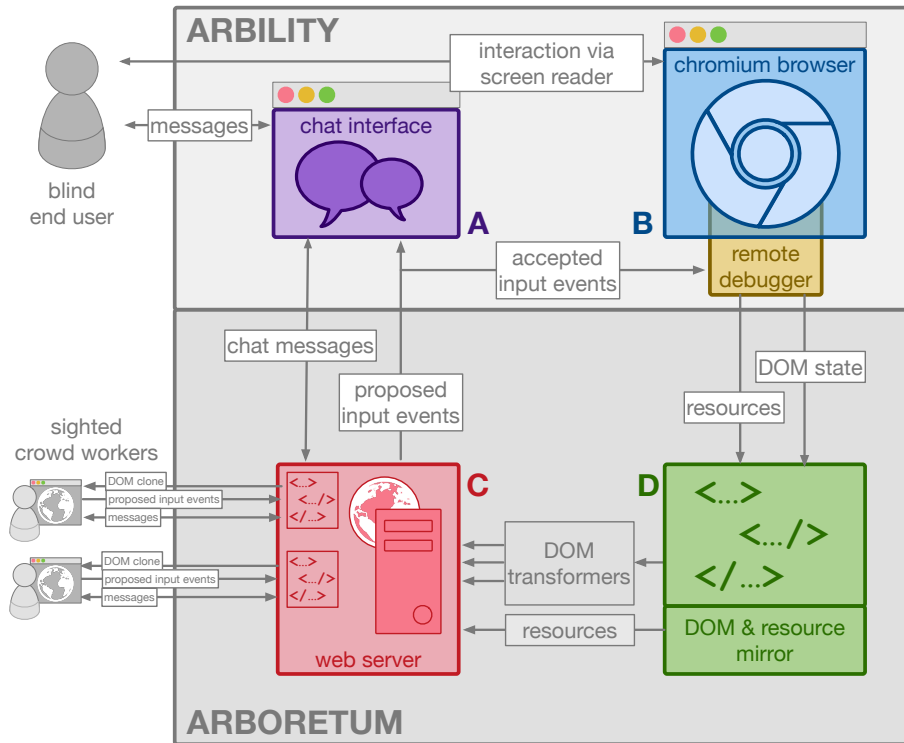
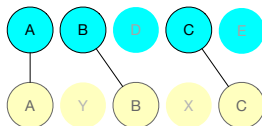


Figure 3. A system diagram of Arbility, Arboretum, and their interactions. Arbility bundles a chat interface (A) and web browser (B). Arboretum includes a web server (C) and module for mirroring pages for remote users (D). The end user can interact with the browser as normal or by accepting input events that are proposed by remote crowd workers. The end user can also discuss task specifics with remote workers through the chat interface.

We tuned these constants based on preliminary experiments using Arbility on several frequently-changing webpages. For the two trees described earlier, assuming that nodes that are in both trees (A, B, and C) have high similarity scores, we would end up with the following alignment:



We chose to use sequence matching, rather than a global similarity computation, to ensure that the DOM structure and the order is accounted for in the matching process.

EVALUATION

In order to test Arboretum’s ability to seamlessly share web page content and interactions between end users and groups of remote workers, and to test the usability and benefit of Arbility for blind participants, we performed a laboratory evaluation with 9 blind participants consisting of 3 interactive web tasks followed by a post-study survey.

Participants

We recruited 10 blind participants by posting on Twitter and through mutual connections in the blind community. We omitted one participant who—unbeknownst to the study coordinator—completed the user study on their mobile phone. Because both Arbility and our study were designed

for desktop browsers, this participant faced navigational challenges that other participants did not—specifically, the participant accidentally closed a relevant tab during the task. However, this participant did successfully complete all of the study tasks using Arbility through their phone.

Of the remaining 9 participants, 8 of them had 16 or more years of experience using a screen reader. Participants were compensated \$35.00 for an hour-long remote study. This rate of pay is commensurate with participants’ specialized skill in using a screen reader, a necessary and hard-to-fulfill prerequisite for our study. Additionally, we recruited crowd workers from Amazon Mechanical Turk (MTurk). Crowd workers were required to have a 95% approval rate and be located in the United States. We recruited these workers using the retainer model [9, 12] via LegionTools [38]. The retainer model automatically posts tasks to MTurk as needed, and continuously adjusts worker compensation based on demand (i.e. if the retainer is empty then compensation will be higher, if the retainer is full then compensation will be lower). Workers were compensated 50–100 cents for a task taking 300 seconds on average for an effective pay rate of \$6–12 per hour.

Setup

Every instance of the study was conducted with remote participants, each of whom interacted with a version of Arbility that was slightly modified to work within the browser. Using a browser-based version of Arbility allowed our participants to use the tool without needing to install it. All of the interactions with this browser-based were the same as those in the desktop version. The study asked the blind participants to

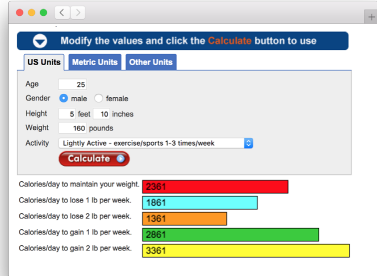
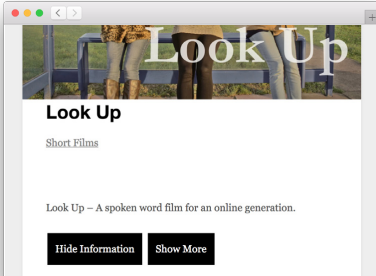
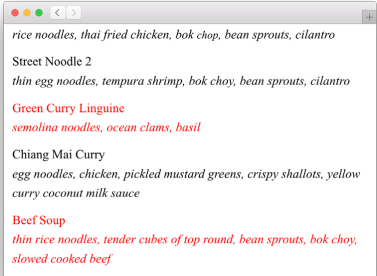
Task Web Page		
<p>(1) Calorie Counter</p> 	<p>(2) Gary Turk Video</p> 	<p>(3) Noodlehead Menu</p> 
Task Question		
How many calories are needed to lose 1 pound?	Name the person who did sound engineering for the video.	Name one \$8 noodle dish that is indicated as spicy.
Reason for Inaccessible Content		
Uses inaccessible <canvas> elements to display information.	Requires mousedown event whereas screen readers simulate click events.	Spiciness is indicated by red text styling only.

Figure 4. The web pages (1) Calorie Counter, (2) Gary Turk Video, and (3) Noodlehead Menu were selected from Bigham, et al. to provide a representative sample of inaccessible web content. Page interactions proposed by crowd workers, such as clicking the ‘Show More’ <div/> in (2), are recorded and replay-able from the Arbility chat panel upon visiting the same page later on. This allows blind end users to overcome the same obstacles in the future without having to call on crowd workers again, reducing cost and increasing independence in the long term.

complete three information finding tasks. Each task consisted of a page with task instructions, a link to the Arbility shared browser page, and a task question whose answer was located on the shared browser page. To avoid biasing participants in favor of using Arbility, the task instructions explicitly stated that the answer might not be inaccessible (i.e., participants did not know ahead of time if a given task posed an accessibility challenge or not). The Arbility shared browser page contained two panels: a chat panel for communicating with remote workers and a content panel containing the original content of the in-the-wild web page. From the task instructions page, participants were asked to click a link to launch the shared web page in another browser tab.

Tasks

In order to choose a representative set of tasks, we first asked our pilot participants for examples of inaccessible page elements they typically encountered. We found that the sites they found to be most problematic included pages with content embedded in untagged images/canvases, encoded using CSS styling, or hidden behind improperly formatted page elements. These types of problems were represented in the tasks used by a study from Bigham et al. [15], so we chose a subset of the tasks from that study that were deemed inaccessible by WCAG 2.0 standards. The specific inaccessible elements were (1) important information contained in images lacking alternative text, (2) poorly constructed forms and buttons, and (3) conveying information through the visual styling of text. The specific pages corresponding to these inaccessible elements were (1) a Calorie Counter page, (2) a video page for the musician Gary Turk, and (3) a menu page for the

Noodlehead restaurant (Figure 4). Blind participants were instructed to retrieve a specific piece of information from each web page (i.e., the answer to the task question). In all cases, the requested information was inaccessible based on WCAG 2.0 standards (Figure 4), which meant that blind participants would most likely need to collaborate with remote sighted workers in order to retrieve the piece of information.

In running our study, we closely followed the methodology of Bigham et al. [15]:

- Tasks were run remotely, allowing participants to use their preferred screen reader and environment, which prior work has considered more ecologically valid [11, 55].
- All widely-used screen readers (Jaws, NVDA, VoiceOver) were represented in our group of recruited participants.
- Task instructions were identical to those in [15] except for added instructions about requesting crowd assistance via the Arbility shared browser page.

Collaboration

In order to retrieve the inaccessible information, blind end users collaborated with sighted remote workers, primarily through two interaction types: natural language text-based chat and proposed page interactions (i.e., clicking, scrolling, hovering on particular page elements). Interactions had to be proposed by crowd workers via the chat panel and were optionally accepted, rejected, or ignored by end users. For example, in the Gary Turk Video task, the requested information could only be retrieved by clicking an incorrectly specified DOM element that was listening for the `mousedown` event—specifically, a “Show More” <div/> element that was styled

Accuracy (%)	Counter	Video	Menu
Blind (Solo)	0	63	14
Sighted (Solo)	100	90	86
Blind+Sighted (Arbility)	100	89	89
Average Time to Success (s)	Counter	Video	Menu
Blind (Solo)	n/a	108	133
Sighted (Solo)	62	93	82
Blind+Sighted (Arbility)	418	240	304

Figure 5. When blind users collaborate with sighted workers via Arbility, their information finding accuracy becomes comparable with that of solo sighted workers (Upper Table). However, these accuracy gains come at a cost in speed, taking on average 3–4 times as long as sighted workers acting alone (Lower Table).

as if it were a `<button/>` element (Figure 4). Activating this element would not usually require a `click` interaction via the keyboard (but merely a `keypress`), and so executing a `click` may have been unintuitive when navigating via a screen reader. However, since the element visually resembled a button, clicking would have been an intuitive interaction when navigating via visual-motor skills and the mouse. Hence, crowd workers were able to propose the `mousedown` event for blind end users to accept and retrieve the requested piece of the information.

Results

In comparison with the baseline performance of solo blind and sighted participants from Bigham, et al., blind participants who used Arbility were dramatically more accurate on every task when compared with those who did not [15]. In particular, Arbility allowed blind participants to come very close to matching the performance of their sighted counterparts (Figure 5). This is most evident on the Calorie Counter task, for which solo blind participants *never* reported the correct answer (0% accuracy), solo sighted participants *always* reported the correct answer (100% accuracy), and blind participants collaborating with sighted participants via Arbility also *always* reported the correct answer (100% accuracy). Effectively, Arbility removed the barriers to information access by transferring web navigation capability from sighted to blind users, at a cost of time and money. Although Arbility allows blind end users to successfully complete information finding and navigation tasks previously impossible via a screen reader, this transference of web browsing capability—from remote workers to end users—is by no means instantaneous. On average, blind participants using Arbility took 3–4 times longer than their sighted counterparts on the same tasks (Figure 5), most likely because collaboration takes time. Blind participants needed to give directions to crowd workers via the chat panel, and to accept or reject any of their proposed actions. However, if the information is valuable enough to the end user, this cost could be worth paying, as it is in the case of remote video assistance systems like Aira [20]. In the next section, we report blind participants’ subjective assessments of Arbility’s usability and discuss their recommendations for improvements.

FEEDBACK AND DISCUSSION

In a post-study survey, we asked participants to rate their agreement with a set of statements based on the Technology Acceptance Model (TAM). TAM is a popular information systems acceptance model intended to predict and explain why end users end up adopting tools, based on two primary criteria: ease of use and perceived usefulness [53, 27]. Participants rate aspects of the tool’s usability on a scale of 1–7 where 1 is “strongly disagree” and 7 is “strongly agree”. In addition to the TAM survey, we also asked participants to give open-ended feedback about the positive and negative aspects of Arbility. The following sections summarize the key quantitative and qualitative trends that emerged from these two different forms of feedback.

Practical and Real-World Applicability

Responding to Arbility’s perceived usefulness (specifically, the statement: “Using a shared web browser could make it easier to navigate the web”), participants expressed a mean level of agreement of 5.67 (SD=0.87), indicating a positive view of the tool’s practical applicability to real-world scenarios, which participants felt were well-represented by our selection of tasks from [15]. Indeed, participants had the following to say:

“It offers a practical way to get sighted help, when that help may not be available or desirable in person.” (P4)

“The problems posed in the tasks were very realistic. I have either encountered similar issues on web pages, or could easily imagine them happening. The assistants were able to provide answers that the screen reading software had no way to find.” (P5)

“It is great to get quick answers to questions that can’t be answered on an inaccessible page. A lot of time could be saved, and it could save me a lot of frustration.” (P6)

These comments touch on our guiding principles in developing Arbility: preserving independence and agency for blind end users who may not want to request in-person assistance from friends or family, overcoming frequently-occurring web navigation obstacles, and saving blind end users time (in comparison with the time required to request and receive in-person assistance). Additionally, participants were enthusiastic enough about the idea to make suggestions for future applications:

“This is a good idea, especially for use on-the-fly, possibly in travel or other business settings. . . . Another way I could see it being really useful is for people needing to access government services that have been moved to an online-only model but they don’t have the access and/or skills. This could be a really cool part of any ‘assisted digital’ model! Having a real person helping instead of a chatbot would be a big draw!” (P3)

“Could this concept be expanded to things such as help with filling out problematic forms, or perhaps Captchas that don’t have an audio alternative?” (P7)

Protecting End User Privacy

In considering their behavioral intent to use a shared web browsing tool like Arbility (specifically, the statement: “I would be a frequent user of a shared web browser”), participants expressed a mean level of agreement of 3.89 (SD=1.17), indicating a slightly negative view of how frequently they might need to—or want to—rely on such a tool. This is somewhat contrary to their positive-leaning attitude toward the idea of shared browsing (specifically, “Web navigation through shared browsing is a good idea”; $\mu=4.56$, SD=1.33) as well as the tool’s ease-of-use (specifically, “I find the shared web browser easy to use”; $\mu=4.89$, SD=0.78). Although positive-leaning, one common reason for hesitation is that Arbility is not entirely privacy preserving. Of the 9 participants, 4 expressed reservations about the implications for web browsing privacy:

“Privacy is a concern. Although the mechanism itself ensures that no personal data is shared, the content of the website may do so (as was the case with the calorie counter).” (P4)

“I’m strange, but I feel under pressure when being observed while someone waits helpfully. It isn’t a problem for someone else to see many of sites that I browse, but I find I tend to have the most trouble when I am on sites that I would hesitate to share because of privacy concerns.” (P6)

Deprioritizing Accessible Web Development

In considering the extent to which using a tool like Arbility aligns with their values (specifically, the statement: “I like the idea of shared web browsing based on the similarity of my values and the societal values underlying its use”), participants expressed a mean level of agreement of 4.67 (SD=1.58), suggesting some ambivalence about the society-wide implications of the development and use of a shared browser for overcoming accessibility obstacles. Of the 9 participants, 3 expressed concerns that a shared web browsing system like Arbility—if widely deployed—would discourage or demotivate the development of web content that is accessible from the start.

“Such a system, in general, would possibly allow developers to avoid making creating accessible content a priority.” (P2)

“I think this type of system sends the wrong message to the non-disabled web developer community. It suggests that they don’t have to solve accessibility problems, because someone else will do it. The real solution is to put more effort into accessible and inclusive design across the web industry.” (P5)

In principle, the authors agree that it is preferable to develop standards and best practices to guarantee that newly developed web content prioritizes accessibility. We do not propose Arbility as a universal solution to the problem of web accessibility, but rather as an ad-hoc solution to a problem that undeniably exists today. In the next section, we discuss some of the future improvements and design challenges opened by Arbility and Arboretum.

FUTURE WORK

Arbility and the underlying Arboretum architecture open up many opportunities for future research, both inside and outside the domain of accessibility. Much of the feedback received from blind participants during this study would be equally applicable to sighted users of a shared browser tool, especially with respect to privacy concerns and workflow integration and automation.

Addressing Privacy Concerns

The most common response from user study participants was that they would like to see privacy concerns addressed to ensure that remote users would not be able to see sensitive information that might be on the page. While privacy was not the focus of this iteration of Arbility, there are several non-technical solutions that could work with Arbility’s existing architecture: using an organized set of trusted crowd workers specifically for accessibility tasks, like Aira’s professional agents [20] or encouraging users to share content with family and friends in situations where they are concerned about privacy. Privacy concerns are highly subjective, and it is unlikely that 100% of privacy issues can be solved with technology alone, but we plan to explore how automated techniques might alleviate privacy concerns, as has been explored in other crowdsourcing applications [35, 64]. In particular, we plan to explore ways to give end users fine-grained control over which elements remote users can and cannot see.

Better Automation via Hybrid Intelligence

Arboretum could also be used to create hybrid intelligence workflows that coordinate actions between AI agents, crowd workers, and end users when trying to complete a task. In this model, crowd workers and the end users would fill in where automated techniques fall short, allowing both maximum robustness while requiring minimum human effort [37].

CONCLUSION

In this paper, we introduced Arboretum, a novel shared web browsing architecture for seamlessly transferring web browsing tasks and Arbility, a web accessibility tool that allows blind end users to hand off targeted visual interaction tasks to remote crowd workers. Our evaluation of Arbility showed that it allows blind users to perform web tasks that would have otherwise been difficult or impossible. This demonstrates Arboretum as an open source platform capable of making future progress on real-world problems via interactive, hybrid-intelligent systems and general web automation.

ACKNOWLEDGEMENTS

The design of Arboretum and Arbility benefited greatly from the valuable feedback provided by Emilie Gossiaux. Additionally, we thank Maximilian Speicher for his work on an earlier iteration of this project, Stephanie O’Keefe for her help in drafting this paper, the LightHouse for the Blind and Visually Impaired for assisting with participant recruitment. We also thank our participants (both the blind and low-vision end users, and Mechanical Turk crowd workers) for their time and feedback during our user studies. This work was supported in part by IBM and the University of Michigan.

REFERENCES

1. Tânia Medeiros Aciem and Marcos José da Silveira Mazzotta. 2013. Personal and social autonomy of visually impaired people who were assisted by rehabilitation services. *Revista Brasileira de Oftalmologia* 72, 4 (2013), 261–267.
2. Amaia Aizpurua, Myriam Arrue, and Markel Vigo. 2013. Uncovering the Role of Expectations on Perceived Web Accessibility. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 74:1–74:2.
3. James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. Plow: A Collaborative Task Learning Agent. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 1514.
4. Saleema Amershi and Meredith Ringel Morris. 2008. CoSearch: a system for co-located collaborative web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1647–1656.
5. Vinod Anupam, Juliana Freire, Bharat Kumar, and Daniel Lieuwen. 2000. Automating Web navigation with the WebVCR. *Computer Networks* 33, 1 (2000), 503–517.
6. Apple, Inc. 2015. iOS Handoff Programming Guide. (2015). <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/Handoff/HandoffFundamentals/HandoffFundamentals.html> Accessed: April 2018.
7. Sriram Karthik Badam and Niklas Elmqvist. 2014. PolyChrome: A cross-device framework for collaborative web visualization. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 109–118.
8. Renata Bandelloni and Fabio Paternò. 2004. Flexible interface migration. In *Proceedings of the 9th international conference on Intelligent user interfaces*. ACM, 148–155.
9. Michael S Bernstein, Joel Brandt, Robert C Miller, and David R Karger. 2011. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 33–42.
10. Jeffrey P Bigham, Jeremy T Brudvik, and Bernie Zhang. 2010. Accessibility by demonstration: enabling end users to guide developers to web accessibility solutions. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 35–42.
11. Jeffrey P Bigham, Anna C Cavender, Jeremy T Brudvik, Jacob O Wobbrock, and Richard E Ladner. 2007. WebinSitu: a comparative analysis of blind and sighted browsing behavior. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 51–58.
12. Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samuel White, and Tom Yeh. 2010. VizWiz: Nearly Real-time Answers to Visual Questions. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 333–342. DOI : <http://dx.doi.org/10.1145/1866029.1866080>
13. Jeffrey P. Bigham, Ryan S. Kaminsky, Richard E. Ladner, Oscar M. Danielsson, and Gordon L. Hempton. 2006. WebInSight: Making Web Images Accessible. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 181–188.
14. Jeffrey P Bigham, Tessa Lau, and Jeffrey Nichols. 2010. Trailblazer: enabling blind users to blaze trails through the web. In *No Code Required*. Elsevier, 367–386.
15. Jeffrey P. Bigham, Irene Lin, and Saiph Savage. 2017. The Effects of "Not Knowing What You Don't Know" on Web Accessibility for Blind Web Users. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '17)*. ACM, New York, NY, USA, 101–109. DOI : <http://dx.doi.org/10.1145/3132525.3132533>
16. Jeffrey P. Bigham, Craig M. Prince, and Richard E. Ladner. 2008. WebAnywhere: A Screen Reader On-the-go. In *Proceedings of the 2008 International Cross-disciplinary Conference on Web Accessibility (W4A)*. ACM, 73–82.
17. Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C Miller. 2005. Automation and Customization of Rendered Web Pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*. ACM, 163–172.
18. Yevgen Borodin, Jeffrey P Bigham, Glenn Dausch, and IV Ramakrishnan. 2010. More than meets the eye: a survey of screen-reader browsing strategies. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*. ACM, 13.
19. Andy Brown and Simon Harper. 2013. Dynamic Injection of WAI-ARIA into Web Content. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility (W4A '13)*. ACM, New York, NY, USA, Article 14, 4 pages. DOI : <http://dx.doi.org/10.1145/2461121.2461141>
20. Aira Tech Corp. 2014. Aira. (2014). <https://aira.io/> Accessed: July 2018.

21. Allen Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols. 2010. *No Code Required: Giving Users Tools to Transform the Web*. Morgan Kaufmann.
22. Ryan Dahl. 2009. Node.js. (2009). <http://nodejs.org> Accessed: April, 2018.
23. Google, Inc. 2018. Chrome DevTools Protocol. (2018). <https://chromedevtools.github.io/devtools-protocol/> Accessed: April, 2018.
24. Saul Greenberg and Mark Roseman. 1996. GroupWeb: A WWW Browser As Real Time Groupware. In *Conference Companion on Human Factors in Computing Systems (CHI '96)*. ACM, 271–272. DOI : <http://dx.doi.org/10.1145/257089.257317>
25. Anhong Guo, Xiang 'Anthony' Chen, Haoran Qi, Samuel White, Suman Ghosh, Chieko Asakawa, and Jeffrey P. Bigham. 2016. VizLens: A Robust and Interactive Screen Reader for Interfaces in the Real World. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 651–664.
26. Richard Han, Veronique Perret, and Mahmoud Naghshineh. 2000. WebSplitter: a unified XML framework for multi-device collaborative Web browsing. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 221–230.
27. Hans Heijden. 2004. User Acceptance of Hedonic Information Systems. *MIS Q.* 28, 4 (2004), 695–704.
28. Matthias Heinrich, Franz Lehmann, Thomas Springer, and Martin Gaedke. 2012. Exploiting single-user web applications for shared editing: a generic transformation approach. In *Proceedings of the 21st international conference on World Wide Web*. ACM, 1057–1066.
29. Darris Hupp and Robert C. Miller. 2007. Smart bookmarks: automatic retroactive macro recording on the web. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, Newport, Rhode Island, USA, October 7-10, 2007*. 81–90. DOI : <http://dx.doi.org/10.1145/1294211.1294226>
30. Maria Husmann, Michael Nebeling, Stefano Pongelli, and Moira C Norrie. 2014. MultiMasher: providing architectural support and visual tools for multi-device mashups. In *Web Information Systems Engineering—WISE 2014*. Springer, 199–214.
31. Github Inc. 2003. Electron. (2003). <http://www.electron.atom.io/> Accessed: April 2018.
32. Muhammad Asiful Islam, Yevgen Borodin, and IV Ramakrishnan. 2010. Mixture model based label association techniques for web accessibility. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. ACM, 67–76.
33. Brad Johanson, Shankar Ponnekanti, Caesar Sengupta, and Armando Fox. 2001. Multibrowsing: Moving web content across multiple displays. In *UbiComp 2001: Ubiquitous Computing*. Springer, 346–353.
34. Forrester Research Kate Leggett. 2011. Forrester Technographics Data Points To Increased Communication Channel Usage With Inconsistent Satisfaction Ratings. (2011). Accessed: April, 2017.
35. Harmanpreet Kaur, Mitchell Gordon, Yi Wei Yang, Jeffrey P. Bigham, Jaime Teevan, Ece Kamar, and Walter S. Lasecki. 2017. CrowdMask: Using Crowds to Preserve Privacy in Crowd-Powered Systems via Progressive Filtering. In *Proceedings of the Fifth AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2017, 23-26 October 2017, Québec City, Québec, Canada*. 89–97.
36. Clemens N Klokmoose, James R Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 280–290.
37. Walter S Lasecki and Jeffrey P Bigham. 2013. Interactive crowds: Real-time crowdsourcing and crowd agents. In *Handbook of human computation*. Springer, 509–521.
38. Walter S Lasecki, Mitchell Gordon, Danai Koutra, Malte F Jung, Steven P Dow, and Jeffrey P Bigham. 2014. Glance: Rapidly coding behavioral video with the crowd. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 551–562.
39. Walter S Lasecki, Tessa Lau, Grant He, and Jeffrey P Bigham. 2012. Crowd-based recognition of web interaction patterns. In *Adjunct proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 99–100.
40. Walter S Lasecki, Kyle I Murray, Samuel White, Robert C Miller, and Jeffrey P Bigham. 2011. Real-time crowd control of existing interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 23–32.
41. Walter S Lasecki, Phyo Thiha, Yu Zhong, Erin Brady, and Jeffrey P Bigham. 2013a. Answering visual questions with conversational crowd assistants. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 18. DOI : <http://dx.doi.org/10.1145/2513383.2517033>
42. Walter S Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F Allen, and Jeffrey P Bigham. 2013b. Chorus: a crowd-powered conversational assistant. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 151–162.
43. Tessa Lau, Julian Cerruti, Guillermo Manzato, Mateo Bengualid, Jeffrey P Bigham, and Jeffrey Nichols. 2010. A Conversational Interface to Web Automation. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. ACM, 229–238.

44. Jonathan Lazar, Aaron Allen, Jason Kleinman, and Chris Malarkey. 2007. What frustrates screen reader users on the web: A study of 100 blind users. *International Journal of human-computer interaction* 22, 3 (2007), 247–269.
45. Gilly Leshed, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & Sharing HowTo Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1719–1728.
46. Ian Li, Jeffrey Nichols, Tessa Lau, Clemens Drews, and Allen Cypher. 2010. Here’s What I Did: Sharing and Reusing Web Activity with ActionShot. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 723–732.
47. Anna Loparev, Walter S Lasecki, Kyle I Murray, and Jeffrey P Bigham. 2014. Introducing shared character control to existing video games. In *Proceedings of the International Conferences on the Foundations of Digital Games*.
48. Robert C Miller, Victoria H Chou, Michael Bernstein, Greg Little, Max Van Kleek, David Karger, and others. 2008. Inky: A Sloppy Command Line for the Web with Rich Visual Feedback. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. ACM, 131–140.
49. Meredith Ringel Morris and Eric Horvitz. 2007. SearchTogether: an interface for collaborative web search. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, 3–12.
50. Michael Nebeling and Anind K Dey. 2016. XDBrowser: User-Defined Cross-Device Web Page Designs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.
51. Michael Nebeling, Fabio Paternò, Frank Maurer, and Jeffrey Nichols. 2015. Systems and tools for cross-device user interfaces. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 300–301.
52. World Health Organization and others. 2012. Global data on visual impairments 2010. *Geneva: World Health Organization Organization* (2012).
53. Sung Youl Park. 2009. An Analysis of the Technology Acceptance Model in Understanding University Students’ Behavioral Intention to Use e-Learning. *Journal of Educational Technology & Society* 12, 3 (2009), 150–162. <http://www.jstor.org/stable/jeductechsoci.12.3.150>
54. Fabio Paternò, Carmen Santoro, and Antonio Scordia. 2008. Preserving Rich User Interface State in Web Applications across Various Platforms. In *Engineering Interactive Systems*. Springer, 255–262.
55. Helen Petrie, Fraser Hamilton, Neil King, and Pete Pavan. 2006. Remote usability evaluations with disabled people. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 1133–1141.
56. Helen Petrie and Omar Kheir. 2007. The relationship between accessibility and usability of websites. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 397–406.
57. Christopher Power, André Freire, Helen Petrie, and David Swallow. 2012. Guidelines are only half of the story: accessibility problems encountered by blind users on the web. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 433–442.
58. Yury Puzis, Yevgen Borodin, and IV Ramakrishnan. 2015. Complexities of practical web automation. In *Proceedings of the 12th Web for All Conference*. ACM, 11.
59. Alexander J Quinn and Benjamin B Bederson. 2011. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 1403–1412.
60. Elliot Salisbury, Sebastian Stein, and Sarvapali Ramchurn. 2015. Real-time opinion aggregation methods for crowd robotics. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 841–849.
61. Yasushi Shinjo, Fei Guo, Naoya Kaneko, Takejiro Matsuyama, Tatsuya Taniuchi, and Akira Sato. 2011. A distributed web browser as a platform for running collaborative applications. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*. IEEE, 278–286.
62. Nate Smith. 2012. ShareDB. (2012). <https://github.com/share/sharedb> Accessed: April, 2018.
63. Surfly. 2012. <https://www.surfly.com>. (2012). Accessed: January 2, 2016.
64. Saiganesh Swaminathan, Raymond Fok, Fanglin Chen, Ting-Hao Kenneth Huang, Irene Lin, Rohan Jadvani, Walter S Lasecki, and Jeffrey P Bigham. 2017. WearMail: On-the-Go Access to Information in Your Email with a Privacy-Preserving Human Computation Workflow. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 807–815.
65. Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. 2003. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology (EUROCRYPT) 2003*. Springer, 294–311.
66. Yu Zhong, Walter S Lasecki, Erin Brady, and Jeffrey P Bigham. 2015. Regionspeak: Quick comprehensive spatial descriptions of complex images for blind users. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2353–2362.