Interaction Proxies for Runtime Repair and Enhancement of Mobile Application Accessibility

Xiaoyi Zhang[1*], Anne Spencer Ross[1*], Anat Caspi[1,2], James Fogarty[1], Jacob O. Wobbrock[3]
[1] Computer Science & Engineering, [2] Taskar Center for Accessible Technology, [3] The Information School
DUB Group | University of Washington
{xiaoyiz, ansross, caspian, jfogarty}@cs.washington.edu, wobbrock@uw.edu

ABSTRACT
We introduce interaction proxies as a strategy for runtime repair and enhancement of the accessibility of mobile applications. Conceptually, interaction proxies are inserted between an application's original interface and the manifest interface that a person uses to perceive and manipulate the application. This strategy allows third-party developers and researchers to modify an interaction without an application's source code, without rooting the phone, without otherwise modifying an application, while retaining all capabilities of the system (e.g., Android's full implementation of the TalkBack screen reader). This paper introduces interaction proxies, defines a design space of interaction re-mappings, identifies necessary implementation abstractions, presents details of implementing those abstractions in Android, and demonstrates a set of Android implementations of interaction proxies from throughout our design space. We then present a set of interviews with blind and low-vision people interacting with our prototype interaction proxies, using these interviews to explore the seamlessness of interaction, the perceived usefulness and potential of interaction proxies, and visions of how such enhancements could gain broad usage. By allowing third-party developers and researchers to improve an interaction, interaction proxies offer a new approach to personalizing mobile application accessibility and a new approach to catalyzing development, deployment, and evaluation of mobile accessibility enhancements.

Author Keywords
Interaction proxies; accessibility; runtime modification.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g., HCI); K.4.2. Assistive Technologies for Persons with Disabilities.

INTRODUCTION
Mobile devices and their applications (apps) have become ubiquitous in daily life. Ensuring full access to the wealth of information and services provided by such apps is a matter of social justice [39], but for the estimated 15% of the world population with a disability [60], many capabilities and services offered by apps remain inaccessible. Pursuing more complete access requires contributions from researchers, from developers of mobile platforms (e.g., Apple, Google), and from the developers of thousands of individual apps.

Although individual apps can improve their accessibility in many ways, the most fundamental is adhering to platform accessibility guidelines. For example, the Android Accessibility Developer Checklist guides developers to "*provide content descriptions for UI components that do not have visible text*" [1]. This in turn allows the Android TalkBack screen reader to provide meaningful feedback to a person using the screen reader to interact with the app. Unfortunately, many apps fail to implement accessibility guidelines, including flagship corporate apps (e.g., as of August 2016, this paper finds missing metadata in current Android apps for Wells Fargo and Yelp that makes key functionality inaccessible). The problem is analogous to developer failure to provide alternative text for images on the web [6] or to implement accessibility guidelines in desktop applications [28].

When an app fails to implement fundamental accessibility support, people seeking to use that app often have few or no options for resolving the failure. An app's developer may lack awareness or knowledge needed to implement a fix. They may delay a fix due to competing development priorities (e.g., prioritizing new features), and other app updates may even introduce new accessibility failures. Alternatively, it might be difficult or impossible to obtain updates if original development of an app was contracted out or if the app is otherwise abandoned by its developer. A new approach to deploying accessibility enhancements could allow individuals or communities to quickly address accessibility failures that an app's original developer is unable or unwilling to address.

Finally, mobile platforms also face significant challenges and tradeoffs in prioritizing, designing, and implementing potential platform-level accessibility improvements. Although the impact of built-in platform support can be large, many promising techniques never make it to platform-level adoption. At the same time, researchers cannot reasonably rebuild a platform's entire accessibility infrastructure as part of exploring a new approach, so research is often limited to prototype apps. A new approach to deploying accessibility enhancements could both: (1) improve research by enabling

* The first two authors contributed equally to this work.

CHI 2017, May 06 - 11, 2017, Denver, CO, USA

ACM 978-1-4503-4655-9/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3025453.3025846

deployment and evaluation of potential techniques, and (2) accelerate the impact of research by allowing promising accessibility enhancements to reach more people.

This paper introduces *interaction proxies* as a strategy for runtime repair and enhancement of mobile app accessibility. Interaction proxies are inserted between an app's original interface and the manifest interface [14] that a person uses to perceive and manipulate that app (e.g., the interface exposed by a screen reader like Android's TalkBack). As a strategy for third-party accessibility enhancements, interaction proxies therefore contrast with both: (1) fixes in individual apps (which can only be made by an app's developer), and (2) platform-level enhancements (which can only be made by the platform developer). The strategy is analogous to web proxies that modify a page between a server and a browser (including web proxies that modify pages to improve their accessibility [6,8,53]), but extended to address the design and implementation challenges of mobile app accessibility.

We demonstrate interaction proxies on Android, modifying interactions without an app's source code, without rooting the phone or otherwise modifying an app, while retaining all capabilities of the system (e.g., Android's full implementation of the TalkBack screen reader). Our implementations achieve this by modifying selected interactions using an accessibility service that creates overlays above an app while listening to and generating events to coordinate an interaction.

We then present a set of interviews with blind and low-vision people interacting with prototype interaction proxies, discussing their experiences and thoughts regarding such enhancements. As we will emphasize throughout this paper, our contribution is not intended to be any one specific accessibility enhancement. Rather, we aim to demonstrate the potential of interaction proxies as a *strategy* to support a variety of repairs and enhancements to the accessibility of mobile apps.

The specific contributions of our work therefore include:

- We introduce *interaction proxies* as a strategy for runtime repair and enhancement of the accessibility of mobile apps.
- We explore a design space of interaction re-mappings that can be implemented by an interaction proxy. This provides a framework for considering the design and implementation of mobile accessibility repairs and enhancements.
- We present a set of techniques for implementing interaction proxies and demonstrate implementations on Android without rooting the phone or otherwise modifying an app. A set of proof-of-concept Android implementations of interaction proxies demonstrate a range of technical techniques that developers and researchers can employ in implementing, deploying, and evaluating accessibility improvements.
- We present two sets of interviews with blind and low-vision people who use screen readers, first exploring accessibility barriers they encounter in apps and then exploring their reactions to using prototype interaction proxies as well as their thoughts on the potential of interaction proxies for enhancing the accessibility of mobile apps.

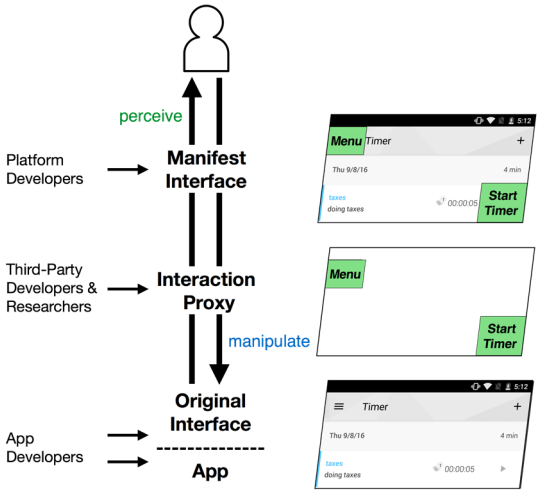

Figure 1. Interaction proxies are inserted between an app's original interface and the manifest interface a person uses to perceive and manipulate that app. This allows third-party developers and researchers to modify the interaction.

INTERACTION PROXIES

Effective interaction requires bridging both the gulf of evaluation (i.e., *perceiving and understanding* the state of an app) and the gulf of execution (i.e., *manipulating* the state of an app) [30]. These gulfs are bridged using an *interface*. For many people without disabilities, that interface is the original touch screen display. But it is important to clarify that *people commonly use different interfaces to access the same underlying app*. Many blind or low-vision people choose a *screen reader interface* (e.g., Slide Rule [34], Android's TalkBack, iOS's VoiceOver), which re-maps touch to support navigation separate from activation and may completely disable visual display of an app. Many people with motor impairments choose a *scanning interface*, which relies on visual perception but re-maps manipulation by scanning through potential targets that a person then activates using a switch. Even attaching an external keyboard or an additional display to a device can be considered a change in the interface used for interaction.

Borrowing from Cooper [14], we refer to the interface a person directly perceives and manipulates as the *manifest interface*. We contrast this with the *original interface* created by an app developer. For many people, the manifest interface may be the same as the original interface. However, many people with disabilities elect an alternative manifest interface. Conceptually, interaction proxies are inserted between an app's original interface and the manifest interface a person uses to perceive and manipulate that app, as illustrated in Figure 2.

We adopt the term "proxy" to be analogous to the web, where proxies can modify webpages between a server and their rendering in a browser (including web proxies with a purpose of improving web accessibility [6,8,53]). However, mobile app architectures do not allow the same approach to directly modifying the underlying representation of an interface prior to its rendering. Inserting an interaction proxy therefore aims to improve interaction by modifying how an app is perceived or

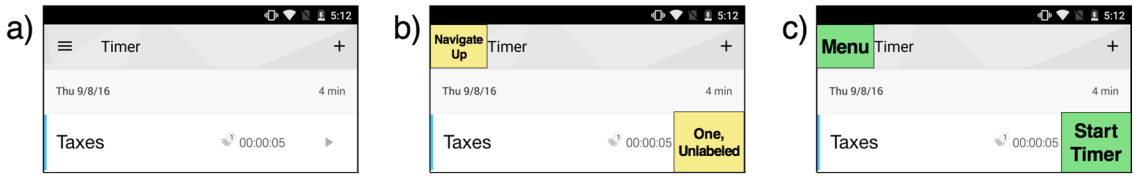

Figure 2. This interaction proxy repairs accessibility metadata than an app provides to a platform screen reader. (a) Toggl is a popular time-tracking app. (b) Its interface includes elements with missing or inappropriate metadata, which a screen reader manifests as "Navigate Up" and "One, Unlabeled". (c) Our third-party interaction proxy repairs the interaction with appropriate metadata for each element, so a screen reader manifests these elements as "Menu" and "Start Timer".

manipulated, without actually modifying the app. Key benefits are: (1) this does not require modifying the app, and (2) this does not require modifying the renderer of a manifest interface (e.g., a person continues to use Google's full implementation of Android's TalkBack screen reader). Figure 1 illustrates insertion of an interaction proxy to implement repairs of accessibility metadata in Figure 2. Using floating windows and other methods we detail in this paper, the interaction proxy provides corrected accessibility metadata. This can be implemented by a third-party developer or researcher, and the interaction proxy's modifications are then presented as part of the interface that is manifested by the platform screen reader.

There is significant overall promise in interaction proxies, which we examine in implementing several proof-of-concept enhancements and in interviews with blind and low-vision people interacting with those prototypes. But the interaction proxy strategy also has limitations. Specifically, the strategy is limited to interactions where a proxy can integrate itself into perception and manipulation of an app via a particular interface. A proxy must support both: (1) perception and understanding of the underlying app and the proxy's modifications via the manifest interface (e.g., Figure 1's upward arrow), and (2) manipulation of the underlying app and the proxy's modifications via the manifest interface (e.g., Figure 1's downward arrow). Proxies may simplify this by targeting only small portions of an interaction, leaving most of the original interaction unmodified. But care is required in developing an interaction proxy, as any seams or failures introduced by an interaction proxy can create new barriers to bridging the gulfs of evaluation and execution.

RELATED WORK

Our current work builds upon and is informed by prior research in mobile accessibility implementation, in accessibility repair and enhancement, and in runtime interface modification.

Mobile Accessibility Implementation

Designing and implementing support for mobile accessibility requires diverse contributions. One example can be seen in the design, adoption, and implementation of support for screen readers that make mobile apps more accessible to blind and low-vision people. Motivated by the research challenge of supporting blind people in using touch screens, Slide Rule introduced techniques for re-mapping gestures to support navigating and exploring the screen separately from activating targets in an interface [34]. These techniques were adopted by developers of screen readers for major mobile platforms (e.g., Android's TalkBack, iOS's VoiceOver), and now improve accessibility of apps on those platforms [42]. However, accessibility of apps on those platforms remains critically dependent upon the many developers of individual apps [1]. It is best for apps to be designed and developed to be broadly accessible [58], but third-party interaction proxies provide a complementary strategy when apps fall short of this goal.

One key requirement of the developers of individual apps is to provide metadata needed to support platform accessibility enhancements such as screen readers. Developers often fail to provide metadata for a variety of reasons (e.g., being unaware they should, lacking knowledge of how to do so, or failing to prioritize accessibility). This problem has been most studied on the web (e.g., with missing image alternative text [6]). Despite education and improved testing (e.g., [12,31,47]), such accessibility failures remain common on the web [25].

The same underlying challenge occurs in mobile and desktop interfaces, with individual app developers failing to implement platform accessibility support. The extent of the problem is difficult to measure, due to a relative lack of testing tools. Hurst et al. found roughly 25% of desktop elements are completely missing from the accessibility implementation of common desktop interfaces [28]. Milne et al. examined the accessibility of apps provided with mobile health sensors, finding none appropriately implemented platform accessibility standards [43]. The greater maturity of education and tooling in web accessibility makes it likely that mobile and desktop accessibility implementations are at least as poor as on the web, and blind and low-vision interview participants in this paper report missing accessibility metadata is a frequent barrier.

Accessibility Repair and Enhancement

Prior work repairing accessibility failures generally focuses on the web, where the representation rendered by a browser is available and can be directly modified. We use the term "proxy" to be analogous to prior research using web proxies to improve web accessibility [6,8,53], but our implementation mechanisms are quite different. Some accessibility repairs can be automated, such as attempting to automatically identify sources of alternative text for web images [6] or automatically increasing font size to improve readability [4]. Other approaches emphasize social annotation, wherein people contribute accessibility repairs that benefit additional people who encounter the same accessibility failure [37]. Such research has examined collaborative authoring of missing image alternative text and other metadata [51,54,55], sharing of scripts to implement site-specific repairs [7], and more recent work focusing on crowdsourcing contributions [27].

In addition to work on visual accessibility, many people with motor impairments choose accessibility techniques that modify how they manipulate an interface. SUPPLE modifies interface elements according to personal motor abilities [23,58]. Other systems leave interface layout unchanged, but apply techniques to ease pointing. Examples include making targets "sticky" [29,59], dynamically modifying cursor behavior according to a person's movement profile [57], and breaking a pointing interaction into multiple interactions that disambiguate the intended target [22,33,63]. Other research has explored alternative pointing for physically large devices [36].

Prior research examines how people adapt to accessibility barriers in mobile devices and how devices can enhance access in the physical world [35]. VizWiz examines supporting blind people in using a mobile device to take a picture and ask a crowd a question about that picture [5,11]. Extensions examine conversational interaction with the crowd [41] and soliciting volunteer assistance in social networks [9,10]. Related insights have been applied to supporting deaf students through real-time captioning for classroom activities [40].

There has been much less research in repairing or enhancing the accessibility of mobile apps. Notable examples include supporting macros motivated by accessibility needs [48] and implementations of pointing enhancements [63]. The SWAT framework examines system-level instrumentation of content and events to support developer creation of accessibility services [49], but requires rooting a device, which is a significant security risk and presents a technical expertise barrier. Our work explores a larger design space of re-mappings while working within the Android accessibility and security model. We thus extend the spirit of prior work in repair and enhancement, while developing new techniques for third-party accessibility enhancements in mobile apps.

Runtime Interface Modification

We also extend prior research in runtime modification of desktop interfaces. Such work employs different approaches than on the web, necessitated by a lack of ability to directly access or modify an interface's internal representation. Early work replaced a toolkit drawing object and intercepted commands (e.g., *draw_string*) [21,46]. More recently, Scotty examined runtime toolkit overloading, developing abstractions for implementing modifications in code injected directly into the interface runtime [20]. Complementary work explores input-output re-direction in the window manager [56], using information from the desktop accessibility API to make changes in an interface façade [52]. To overcome limitations of the desktop accessibility API and incomplete implementations of that API, recent work examines interpretation of interface pixels [28]. Runtime modification can be implemented using only pixel-level analysis [15,16,17,18,19,61] or in combination with information from an accessibility API [13]. Such approaches can allow authoring more specialized or accessible manifestations of underlying application functionality [62].

Our work builds on insights from the desktop, but the mobile app context does not allow prior approaches (i.e., does not allow direct modification of interface internals, nor surface modification via window manager redirection). We develop a new approach (i.e., using floating windows while coordinating perception and manipulation in the manifest interface), and we explore implications for mobile accessibility modifications.

RE-MAPPING INTERACTION

We now present a design space for supporting researchers and developers in the *design* of interaction proxies, considering *implementation* abstractions in the next section. Recall from Figure 2 that an interaction proxy helps a person to bridge the gulf of evaluation and/or the gulf of execution. A proxy achieves this by modifying: (1) perception in an interaction, (2) manipulation in an interaction, or (3) both. We consider such modifications in terms of how a proxy re-maps existing interaction into new interaction. This section considers each of five identified patterns of re-mapping and discusses potential interaction proxies in that region of the design space.

Although many potential enhancements fit cleanly into this design space, we note that some enhancements are better thought of as a combination of several interaction proxies, each of which re-maps an interaction according to this space.

From Zero to One

A re-mapping from *zero to one* adds a new interaction where there was none. By definition, this adds new information or functionality to a manifest interface, and we contrast it with later examples that correct or replace an existing interaction. Re-mappings might integrate new information obtained from an outside service or restore elements of an original interface that are inaccessible in a person's chosen manifest interface.

Figure 3 shows an example of adding information from an outside service. In interviews presented in this work, blind and low-vision people report they often do not have ready access to information about an app's accessibility prior to attempting to use the app. If a third-party service rated app accessibility, it would be preferable to make ratings available directly at the time that information is needed. Figure 3's proof-of-concept shows how a third-party interaction proxy could add a button for accessing such accessibility ratings from within the app store itself. Google's Accessibility Scanner app similarly inserts a floating button that allows invoking an accessibility checker for the current screen [24], but it targets developer inspection of an app rather than end-users seeking information about the accessibility of potential apps.

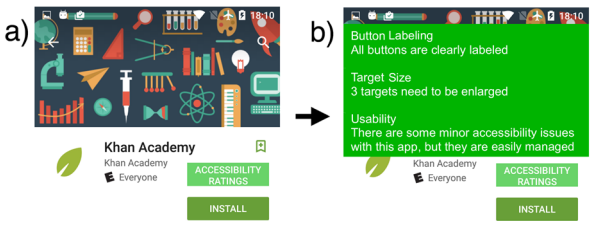

Figure 3. This interaction proxy adds third-party accessibility ratings directly within the app store. (a) An "Accessibility Ratings" button is added in empty space. (b) Selecting it shows third-party accessibility information for that app.

Later sections discuss Figure 7, an example of zero-to-one re-mappings that restore functionality missing in an interface manifested by a screen reader. Specifically, the Wells Fargo app fails to expose several of its interface elements, leaving them inaccessible to a person manifesting the interface with a screen reader. An interaction proxy repairs this, restoring access to each of the menu items from the original interface. Figure 8 presents the same problem occurring with the stars on Yelp's rating page. Inaccessible when using a screen reader, an interaction proxy restores these to the manifest interface.

From One to Zero

A re-mapping from *one to zero* removes an interaction. Examples include: (1) modifying a manifest interface to remove information or functionality from the original interface, and (2) modifications to correct or improve undesirable interactions introduced by a person's chosen manifest interface. For example, stencils-based tutorials remove access to much of an interface by limiting interaction to the current step in the tutorial [26,38]. iOS's Guided Access feature similarly allows disabling a phone's motion detection or disabling touch in a region of an interface [3]. Additional examples can include removing interactions that provide no value or are problematic in a person's chosen manifest interface (e.g., advertising, interstitial screens, spurious animations that generate distracting notifications with a screen reader).

From One to One

A re-mapping from *one to one* replaces an existing interaction with another. Figure 1 shows an example of elements with missing or incorrect accessibility metadata, which are therefore inappropriately manifested by a screen reader (e.g., presented as "One, Unlabeled"). Manipulation thus works as desired, but elements are difficult to correctly perceive. A third-party interaction proxy can correct this (e.g., presenting the button as "Start Timer"), proxying any manipulation to the underlying original interface. Although such a correction could also be considered a combination of a one-to-zero re-mapping (i.e., removing the incorrect data) and a zero-to-one re-mapping (i.e., adding the correct data), it is more straightforward to consider such direct replacement of an existing interaction as a one-to-one re-mapping.

Alternatively, an interaction proxy can modify manipulation while leaving information in individual elements unchanged. For example, our blind and low-vision interview participants report needed functionality can sometimes be difficult to reach when manifested by a screen reader. Because swiping gestures traverse elements serially, an example difficulty is when a screen reader manifests a target at the end of a list that must be traversed to access it (e.g., Toggl's "Create Timer" button prominently floats above the interface but the screen reader manifests it after all existing timers, Yelp's search box is readily available in the original interface but the screen reader manifests at the end of a list). Later sections discuss Figure 8 and our implementation of interaction proxies to modify navigation order in such interfaces.

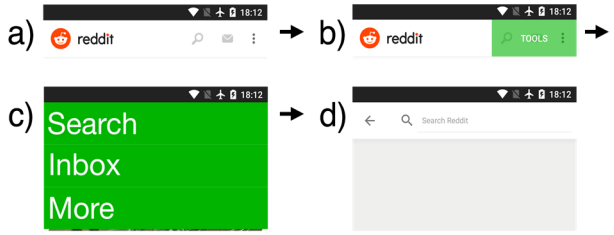

Figure 4. This interaction proxy replaces one interaction with a sequence of two interactions. (a) An interface contains several small adjacent targets. (b) Replaced with a single larger "Tools" button. (c) Selection displays a menu of the original targets. (d) A selected item is activated.

We note existing platform support for re-mapping interaction tends to be strongest for one-to-one re-mapping (e.g., both Android and iOS support interactively labeling elements that are missing screen reader metadata). But interaction proxies further allow third-party developers and researchers to develop, deploy, and evaluate new approaches (e.g., neither Android nor iOS provide integrated support for social annotation approaches to crowdsourcing accessibility text, as has been proposed and examined in web-based systems [27,51,54,55]).

From One to Many

A re-mapping from *one to many* replaces a single interaction with multiple interactions. This is often because the original interaction makes ability assumptions that are inaccessible to some people, such as fine-grained motor assumptions [58]. Unpacking such an interaction into a sequence of interactions can make it more accessible to more people. This generally requires designing for both perception and manipulation, as a person must navigate the new sequence of interactions. For example, prior work explores two-stage selection of small targets [22,33,63]. Figure 4 shows an interaction proxy that replaces a set of small adjacent targets with a single larger target that invokes a menu for choosing among them.

From Many to One

A re-mapping from *many to one* replaces multiple interactions with a single interaction. For example, prior work has explored this in accessibility macros [48]. Such macros can provide a new method for manipulating an app through a sequence of interactions that might otherwise be difficult or laborious (e.g., interfaces that rely upon timing constraints or active modes that are difficult for a person with a motor impairment to perform). A macro might also be invoked using a different modality, as with CoFaçade's support for configuring devices so an older adult can access functionality with on-screen buttons, with physical buttons, or with physical gestures like scanning an RFID tag [62]. In contrast to many-to-one re-mapping of manipulation, many-to-one re-mapping of perception can be implemented by summaries. For example, a home automation app might contain many visual indicators of devices, which a screen reader manifests by serially scanning through each device and its status, but a proxy could add support for a summary of which devices are currently on.

Composing Multiple Re-Mappings in an Enhancement

Apps and potential enhancements to those apps are composed of many interactions. In applying this design space to consider current and potential enhancements, it is helpful to consider enhancements as composed of many re-mappings, each of which is characterized by one of these five patterns.

A simple case is when an enhancement applies the same type of re-mapping to multiple interactions (e.g., modifying screen reader metadata on multiple elements, modifying navigation order among multiple elements, disabling multiple elements as part of a stencils-based tutorial). But complex enhancements can also be understood as compositions of many re-mappings. Later sections discuss personalized interface layout in Figure 10, as motivated by SUPPLE's generation of interface layouts adapted to an individual's motor abilities [23,58]. Considering such a complex enhancement in terms of its underlying re-mapping of many interactions can provide a useful framework for approaching its design and implementation.

IMPLEMENTATION IN ANDROID

The prior section provides a framework for considering *design* of interaction proxies. We now present a set of *implementation* abstractions and how each is implemented in Android. Our abstractions are sometimes similar to prior examinations of runtime interface modification (e.g., [20,52]), so we focus on details for interaction proxies in Android. Abstractions provide a higher-level language for discussing enhancements and convey what is needed to achieve these behaviors on other platforms. After introducing the primary abstractions, we discuss their composition to coordinate perception and manipulation over the course of modifying an interaction. Maintaining this coordination of the interaction is critical for an effective proxy.

Our interaction proxies are installed as Android accessibility services. This provides a proxy with some ability to inspect and manipulate other apps using privileged Android APIs. Installing an accessibility service requires explicit consent, and none of our enhancements require rooting the phone or otherwise modifying the operating system, as we believe it is inappropriate to require a person to compromise basic security of their device to access an app. Nevertheless, protecting against malicious accessibility services is an important topic for additional security research [32]. Prior research has explored composing interfaces from mutually distrustful elements [50], and the development of effective abstractions is important to improving the security of accessibility enhancements.

Abstractions for Android Interaction Proxies

Our basic strategy is to minimize the scale and complexity of an interaction proxy by intervening as little as necessary in an interaction. We achieve this using a combination of *floating windows* with the *introspection* and *automation* capabilities of an accessibility service. We note the percentage of Android devices on which each of the core requirements are available, reporting availability as of December 2016 [2]. Some capabilities are limited to more recent versions of Android, which have more limited market share. These will become more available as people transition to newer devices.

***Floating Windows*:** Our interaction proxies intervene between an app's original interface and the manifest interface by creating floating windows that the interface manifests instead of the underlying app. Similar to overlapping windows on the desktop, floating windows sit above the app in z-order (e.g., Facebook Messenger's floating chat heads allow reading and replying to messages from the context of other apps). The floating windows capability has been included since Android 1.0 and is therefore available on all Android devices.

We further define a *Full Overlay* as a floating window that occludes the entire underlying app, and a *Partial Overlay* as occluding one or more elements while leaving other interactions unmodified. Enhancements composed of multiple re-mappings will often coordinate multiple partial overlays within an app.

Event Listeners: Some enhancements require detecting events in the underlying app (e.g., to trigger an update in an overlay). Android allows an accessibility service to listen to interface accessibility events (e.g., a button click, a text field focus, a view update, an app screen switch, a device app switch). Although limited to accessibility events that are invoked by the app, many necessary events are implemented by the default Android tools. This capability has been included since Android 1.6 and is available on 99.9% of Android devices [2].

Content Introspection: Some enhancements require knowledge of the content of elements in an underlying app. Proxies can inspect the app accessibility service representation. This provides a tree describing an app, where each node provides information about an element (e.g., content, size, state, possible actions), though we have noted that apps do not always expose correct or complete information via this representation. This capability has been included since Android 4.1 and is available on 97.5% of Android devices [2].

Automation: Some enhancements require manipulating an underlying app. Accessibility service automation support allows programmatic invocation of common manipulations of elements exposed via the accessibility service representation (e.g., click, long press, select, scroll, or text input directed to a node in the tree). This has been included since Android 4.1 and is currently available on 97.5% of Android devices [2].

Screen Capture: Some enhancements require information or presentation details not available through content introspection. This can include details of a view that the corresponding accessibility representation does not include in its model (e.g., pixel-precise positioning of content). The accessibility metadata provided by an app may also be incomplete or incorrect. Screen capture allows application of pixel-based methods developed in prior work (e.g., [13,15,16,17,18,19]), though some apps intentionally disable screen capture (e.g., banking apps). Screen capture has been included since Android 5.0 and is available in 60.7% of Android devices [2].

Gesture Dispatch: Some enhancements require simulating a gesture or touch. This can be because the automation events are not expressive enough for an enhancement to obtain a desired behavior, or it can be because an enhancement needs

to manipulate an element that is not properly exposed by the accessibility representation. Gesture dispatch allows simulating a gesture or touch at any screen location. This capability is new in Android 7.0, but we have noted such capabilities will become more common as people transition to newer devices.

Current Implementation Limitations: We focus on Android, but support for these abstractions in other mobile platforms should similarly enable interaction proxies. We have also identified a few limitations of Android's current support. First, Android does not provide a robust unique identifier for each screen within an app. Similarly, the field used for differentiating among elements within a screen is optional and often not specified by developers. It can therefore be challenging to reason about the state of an app. Our current proxies use signatures computed from the tree exposed for content introspection. Pixel-based methods for annotation of interface elements could also be valuable [19]. Second, although an accessibility service can observe events, it cannot consume them (i.e., cannot redirect or prevent an event from occurring). The next section presents one implication of this in coordinating perception and manipulation. Finally, we have noted that accessibility services rely on app developers providing necessary events and metadata. Default tools provide this automatically whenever possible, but it is often missing. Our inclusion of screen capture and gesture dispatch provide additional options for interaction proxy implementation.

Coordinating Perception and Manipulation

A person perceives and manipulates the manifest interface, but that interaction must be re-mapped to the underlying original interface. Coordination of perception and manipulation in a re-mapping is critical to the illusion of the manifest interface remaining seamless (i.e., the interaction proxy being perceived as part of the interface, as opposed to itself being disruptive). The complexity of this coordination will vary according to the nature of the interaction and how it is re-mapped.

We have found that *direct* interaction is generally most straightforward to coordinate, as illustrated in Figure 5. Proxy implementation using floating windows means an interface is composed of layers. Projecting these into the display (i.e., flattening the layers along the z-axis) naturally results in interaction proxies occluding anything below. Manipulation is similarly straightforward. Figure 5 (left) shows selecting a button in the manifest interface, with Figure 5 (right) showing that hit-testing finds nothing at that location in the proxy layer and so selection passes to the underlying original interface. This straightforward coordination similarly extends to other *direct* interaction in the flattened interface. For example, the expected elements are naturally presented by Android TalkBack's exploration mode (i.e., as a person moves a finger around the screen to browse an interface with the screen reader, the expected element is naturally presented).

Coordination can be more challenging for *indirect* interaction. Figure 6 presents an example of a gesture-based interaction (i.e., swiping right to the next element in the navigation order). The manifest interface includes a button with correct metadata

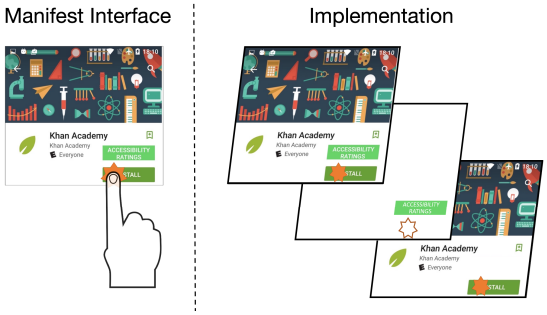

Figure 5. Direct interaction is generally straightforward to coordinate, with interface layers behaving as expected in their occlusion and in mapping input to the appropriate element.

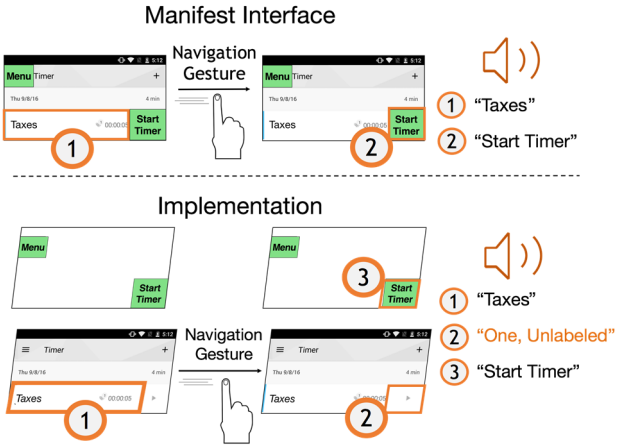

Figure 6. Indirect interaction can require more coordination. Here a gesture-based navigation requires an event listener to watch for "One, Unlabeled" to get focus and then immediately gives focus to "Start Timer". Perception in the manifest interface is seamless because the screen reader truncates reading of "One, Unlabeled", but this illustrates the type of coordination an interaction proxy may need to implement.

(i.e., "Start Timer") inserted to replace an original interface button that lacked metadata (i.e., "One, Unlabeled"). Figure 6 (top) illustrates the interaction experienced, swiping right to hear Android's TalkBack read "Start Timer". Figure 6 (bottom) reveals the navigation gesture is received by the original interface, which is unaware of the interaction proxy and gives focus to "One, Unlabeled". The proxy detects this using an event listener and immediately gives focus to "Start Timer", which is read aloud. In the instant that "One, Unlabeled" has focus, the screen reader begins to read it. However, this is imperceptible because focus moves to "Start Timer" and the screen reader aborts the prior reading (by design, ensuring prompt perception of interaction state during rapid navigation). The illusion of the manifest interface is therefore maintained. Although we do not present all of our interaction proxy implementations in this same detail, this example is intended to illustrate a typical coordination of an indirect interaction.

DEMONSTRATION IMPLEMENTATIONS

Prior sections have introduced several demonstration interaction proxies as part of conveying the strategy, design space, and implementation abstractions. This section presents additional

details and demonstrations. All of our demonstrations were developed as proof-of-concept prototypes. By showing and explaining their key technical approaches, we aim to inform future development of the interaction proxy strategy. Details of these proof-of-concept implementations can also be found in their code, available at: https://github.com/appaccess.

Adding or Correcting Accessibility Metadata

Developers often fail to provide appropriate accessibility metadata for interface elements (e.g., labels for text fields). Although default tools provide this automatically whenever possible, people often encounter apps that have incomplete or incorrect metadata. Platforms have begun to support interactive correction, allowing a person to apply a custom label. But support is limited (e.g., Android only supports custom labels for elements with a `ViewIDResourceName`, which is itself optional and often not specified by app developers). Interaction proxies offer a strategy for third-party developers and researchers to develop and explore new approaches (e.g., social annotation approaches that have been proposed and examined in web-based systems [27,51,54,55]).

Figure 1 shows an example from Toggl, a popular time tracking app. The "Start Timer" button is missing metadata (i.e., the screen reader announces it as "One, Unlabeled") and the menu button has metadata resulting from an implementation artifact (i.e., is read as "Navigate Up"). Figure 7 shows a similar example in the Wells Fargo banking app (i.e., the label "Hamburger Button" is an implementation term better presented as "Menu"). Our interaction proxies identify these failures through *content introspection*, then obtain an image of the element using *screen capture*. Captured images can be used to obtain content descriptions (e.g., our proof-of-concept prototype uses a local database, envisioning social annotation mechanisms in future systems). Each failure is then repaired using a *floating window* to create a *partial overlay* that replaces the element in the manifest interface. Any manipulation of the element in the floating window is proxied to the original interface using *automation* to activate the appropriate element.

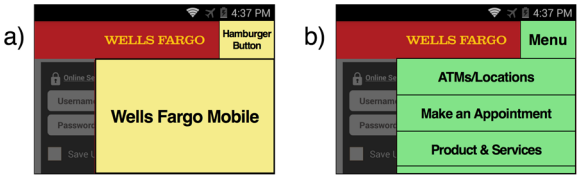

Figure 7. This interaction proxy corrects a "Menu" label and repairs interaction with the app's dropdown menu, which is otherwise completely inaccessible with a screen reader.

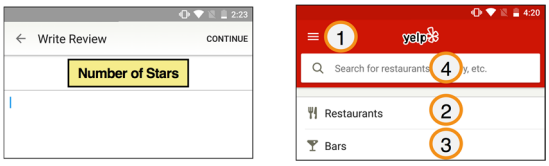

Figure 8. The Yelp app manifests its five-star rating system as a single element that cannot meaningfully be manipulated using a screen reader, and its navigation order using a screen reader makes it needlessly difficult to access "Search".

Restoring Missing Interactions

Figure 7 also illustrates repairing a similar but more severe failure in the Wells Fargo banking app. The original interface's dropdown menu includes several important functions, but does not correctly present itself to accessibility services. It therefore manifests as a single large element, is read as "Wells Fargo Mobile", and activates whatever menu item is in its physical center (e.g., "Make an Appointment"). Figure 8 shows a similar flaw in the Yelp app, which exposes its five-star rating system as a single element that cannot be meaningfully manifested by a screen reader. An interaction proxy repairs these using *screen capture*, a *floating window*, and *content introspection*. The proxy cannot use *automation* to manipulate underlying items, as this will again activate whatever item happens to be in the physical center of the erroneously monolithic element. The proxy instead activates the correct underlying item using *gesture dispatch* (i.e., sending a two-finger touch to the correct screen coordinate, which is consumed by the screen reader, generating a touch in the underlying original interface).

Modifying Navigation Order

Navigation order is a significant aspect of an interface, and optimal navigation may differ for different manifest interfaces (e.g., parallel visual scanning, touch-based exploration using a screen reader, serial navigation using a switch interface). Inappropriate orders can also result from an implementation failure, similar to other incorrect accessibility metadata. Figure 8 shows an example where an implementation error means the Yelp search box visually appears before the scrolling list of businesses, but is manifested after that list in a screen reader (i.e., making it difficult to access via serial navigation). Toggl similarly includes a "Create Timer" button that visually floats above the list of existing timers for easy and prominent access, but is manifested to a screen reader at the end of the list of existing timers. Our interaction proxies modify these navigation orders, moving the appropriate elements to the beginning of the manifest interface, using *content introspection*, *event listeners*, and *automation*, coordinating the interaction similar to the manner discussed in association with Figure 6.

Fully-Proxied Interfaces

Our prior interaction proxies have been minimal, emphasizing the ability to repair or enhance the accessibility of an interaction without needing to re-implement unrelated portions of the original interface. Such targeted re-mappings also highlight challenges of coordinating an interaction so that it blends in to the surrounding interface. Our same abstractions can also be applied in enhancements that proxy the entire interface, more completely changing the interface's manifestation.

Figure 9 shows an example of re-mapping interaction to implement stencils-based tutorials, a technique designed to help guide and focus a person through an interaction using "*translucent colored stencils containing holes that direct the user's attention to the correct interface component and prevent the user from interacting with other components*" [26,38]. Stencils could complement features like iOS Guided Access, providing additional capabilities, and enabling third-party development of different potential guides for varying needs

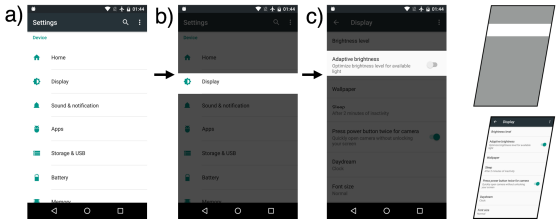

Figure 9. This interaction proxy implements a stencils-based tutorial. At each tutorial step, only the appropriate element is available. All other elements are obscured and disabled.

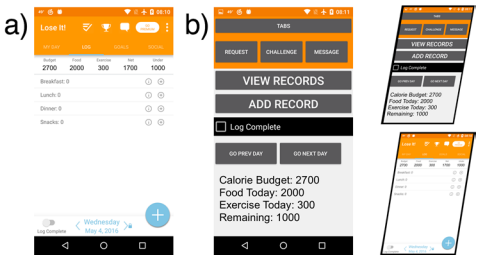

Figure 10. Motivated by research in personalized interface layout, this interaction proxy creates a completely new personalized layout for interacting with the underlying app.

(e.g., stencils in Figure 9 guide a person to a device setting). Our proxy is implemented using a *floating window* to create a *full overlay*, displayed as a translucent overlay and capturing all input. For each tutorial step, it uses *content introspection* to determine the bounds of elements that should be visible through the overlay (i.e., holes in the stencil), then *automation* to proxy manipulation of those elements. An *event listener* ensures the proxy's prompt response to interface changes, as when the interface advances between each step in the tutorial.

Figure 10 shows an example of a personalized interface layout, motivated by SUPPLE's approach to arranging entire interfaces to match an individual's motor abilities [23,58]. Although such personalization is promising for many accessibility needs, adoption of such methods is limited by a need for interfaces to be re-written as abstract specifications [45]. We instead propose an interaction proxy could re-map an original interface into an abstract specification which is then used to generate a personalized interface. As a proof-of-concept, our interaction proxy pictured in Figure 10 re-maps the original interface for Lose It! (a popular food journaling app) into model-level variables, then manifests those in a new interface. The new interface uses a different layout strategy to support larger text and buttons, and it presents underlying functionality differently (e.g., replacing a small sliding widget toggle with a larger checkbox element). Full implementation of this strategy is future research, but our proof-of-concept demonstrates the necessary combination of a *floating window* as a *full overlay*, perceiving the original interface using *content introspection* with *event listeners*, and manipulating it using *automation*.

INTERVIEWS REGARDING INTERACTION PROXIES

We conducted two sets of interviews with blind and low-vision people who use screen readers. Interviews were overall focused on: (1) accessibility barriers and the contexts in which they are encountered, (2) the experience of using an interface with an interaction proxy, (3) usefulness and potential of interaction proxies, and (4) potential for adoption of such enhancements.

Method

Our first set of interviews included eight people who are blind or low-vision and use a screen reader. We discussed what types of accessibility barriers these participants encounter in apps, how they navigate the barriers, how barriers could be addressed, and specific apps in which barriers are encountered. These interviews informed many of the interaction proxies developed in this paper. Specifically, participants identified three common barriers that potentially could be addressed with interaction proxies: mislabeled elements, inaccessible functionality, and challenging navigation. Participants also identified three major categories of app to be of interest: community engagement, productivity, and banking apps. Our proof-of-concept demonstrations and our second set of interviews therefore focused on these needs and opportunities.

Our second set of interviews included six people who are blind or low-vision and use a screen reader (including two from the first set). We developed three proof-of-concept enhancements to present to participants in support of these interviews: (1) Yelp: As illustrated in Figure 8, we repaired the stars on the business rating page to make it possible for a person using a screen reader to rate a business and repaired the navigation order to make the search box easier to reach. (2) Toggl: As illustrated in Figure 1, we repaired missing screen reader labels for elements associated with each existing timer. We also repaired navigation order to make the "Start New Timer" button easier to reach. (3) Wells Fargo: As illustrated in Figure 7, we repaired the items in the dropdown menu to be accessible with a screen reader and repaired the label of the menu button.

Interviews with these participants focused on the potential of interaction proxies to support accessibility enhancements. We asked each participant to use the Android Talkback screen reader to interact with the above three apps (on a Google Nexus 6P with Android 7.0). For each app, participants first completed simple tasks while our interaction proxy applied its repairs, then again with the screen reader manifesting the original interface. We chose this approach (e.g., instead of counterbalancing), in part because tasks generally could not be completed without the interaction proxy. Our focus was therefore on qualitative reactions rather than task metrics, and asking participants to begin with a task we knew was impossible would have undermined the interview. Participants discussed the feeling of the interaction with the interaction proxy active, how well it addressed accessibility barriers, and their ideas for the potential of accessibility enhancements. Interviews were transcribed and then analyzed using open coding.

Participants primarily reported using an iPhone, the more popular choice for people in the United States who use accessibility services [44]. Two reported using Android. Interaction with iOS's VoiceOver screen reader is similar to Android's TalkBack, and the barriers within apps are similar between platforms. We therefore believe these participants provided useful insight into the interaction proxy strategy.

Results

Interview participants reported our proof-of-concept prototype interaction proxies worked well for improving the accessibility of interactions. P3 said "*I think in every case [the enhancement] made it a much better experience than it would normally be*", while P5 said "*I think the enhancements have made it better*".

One goal for participant interaction with our prototypes during interviews was to examine seamlessness of the interaction (i.e., an interaction proxy being perceived as part of the interface, as opposed to itself being disruptive). Interviews explicitly probed this, in part by having participants first interact with the enhanced interface and then the app with its underlying accessibility failures. Participants ideally would not be able to distinguish between a natively accessible interface versus an interface that had been repaired or enhanced by an interaction proxy, and several participants commented that interactions were seamless. P3 said "*[the enhancements] made it behave as I would expect it to. I think, when the enhancements were on, I generally didn't have any trouble completing the tasks, which definitely means it's working*", while P2 said "*[the enhanced Yelp app] acted the way I would expect it to act*".

Two participants commented on swipe-based navigation when using a proxy that repairs metadata by changing screen reader focus (as discussed with Figure 6). P1 described "*lagginess*", and we did note the app and enhancement were unusually slow for this participant. P2 described "*oversensitivity*" that made it more difficult to use swipe-based navigation to select a target without skipping it. However, P2 also explicitly noted that the value provided by the enhancement was enough to outweigh "*oversensitivity*". Even when prompted, participants did not mention any other unusual or bothersome interactions.

Participant responses to the specific enhancements we showed were varied. For example, all participants agreed it was an improvement for the Wells Fargo app menu to be accessible, but all felt the "Forgot Password" item was still difficult to find because the app hid it in the dropdown menu. Even when an interaction proxy improves accessibility of an interface, there can still be usability barriers due to poor interface design choices.

All participants described how the types of enhancements we demonstrated could be made to address barriers they encounter, and all expressed interest in using such enhancements. P1 said "*In email…forward and reply all are at the very bottom of the message and if it's a really long message, it's really a pain to have to scroll all the way to the bottom of the message*". P5 said "*[an enhancement] would definitely be a value to be able to get the Greyhound app accessible so that I could be able to purchase tickets and look at the schedules and so forth*".

In discussions regarding the potential for broad deployment, all participants said they would be willing to submit apps needing repair or enhancement if they thought it was likely the enhancement would be created. P3 is a software developer and indicated he would be willing to create enhancements if good tools were available. P3 and P5 also expressed concern over whether enough people would contribute enhancements. P6 was more optimistic about participation, saying "*I do think that people would be very interested in it, and I think people would want to help contribute to the actual programming, and then people would also be interested in making suggestions.*" Finally, participants discussed their trust of third-party enhancements. They reported that primary factors in whether they would trust an enhancement enough to download it are the reputation of the source, endorsements from known organizations (e.g., the National Federation for the Blind), and feedback from other people who use screen readers.

DISCUSSION AND CONCLUSION

We have introduced interaction proxies as a strategy for runtime repair and enhancement of the accessibility of mobile apps. Inserted between an app's original interface and a manifest interface, an interaction proxy allows third-party developers and researchers to modify an interaction without an app's source code, without rooting the phone or otherwise modifying an app, while retaining all capabilities of the system. We have examined the interaction proxy strategy through a design space of interaction re-mappings, by defining and developing key implementation abstractions, and in Android implementations of proof-of-concept interaction proxies. Details of these proof-of-concept implementations can also be found in their code, available at: https://github.com/appaccess.

These conceptual and technical contributions are our primary contributions, and our interviews with blind and low-vision people who use screen readers provide support for further developing this strategy. Participants were enthusiastic for the strategy, based on our proof-of-concept prototypes repairing accessibility failures in popular real-world apps. Including these prototypes in our interviews provided a real-world context for discussing the potential of interaction proxies, and participants used this as a starting point for discussing other apps in which they have encountered accessibility barriers that might be addressed. Participants saw a need and potential for third-party enhancements, expressing interest in the value they could provide even if a repaired interaction was not quite seamless.

Our ultimate goal is to help catalyze advances in mobile app accessibility. Where contributions have previously been limited to developers of individual apps or the underlying mobile platform, interaction proxies open an opportunity for third-party developers and researchers to develop and deploy accessibility repairs and enhancements into widely-used apps and platforms. P1 motivated a multi-faceted approach by saying "*I mean I think what you did is great, to make some more improvements, but also how we can work with community people and ideally Google and [app] developers*". Interaction proxies therefore provide an additional tool that complements efforts to educate and support developers in improving the accessibility of their apps, as well as improvements in platform accessibility support.

ACKNOWLEDGEMENTS

We thank our participants for their contributions, and reviewers for their feedback on an earlier draft of this paper. This work was supported in part by the National Science Foundation under awards IIS-0811063 and IIS-1053868.

REFERENCES

1. Android Open Source Project. Accessibility Developer Checklist. http://developer.android.com/guide/topics/ui/accessibility/checklist.html#requirements
2. Android Open Source Project. Dashboards. http://developer.android.com/about/dashboards/index.html
3. Apple Inc. Use Guided Access with iPhone, iPad, and iPod touch. https://support.apple.com/en-us/HT202612
4. Jeffrey P. Bigham. (2014). Making the Web Easier to See with Opportunistic Accessibility Improvement. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2014)*, 117–122. http://doi.org/10.1145/2642918.2647357
5. Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Aubrey Tatarowicz, Brandyn White, Samuel White, and Tom Yeh. (2010). VizWiz: Nearly Real-time Answers to Visual Questions. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2010)*, 333–342. http://doi.acm.org/10.1145/1866029.1866080
6. Jeffrey P. Bigham, Ryan S. Kaminsky, Richard E. Ladner, Oscar M. Danielsson, and Gordon L. Hempton. (2006). WebInSight: Making Web Images Accessible. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2006)*, 181–188. http://doi.org/10.1145/1168987.1169018
7. Jeffrey P. Bigham and Richard E. Ladner. (2007). Accessmonkey: a Collaborative Scripting Framework for Web Users and Developers. *Proceedings of the International Conference on Web accessibility (W4A 2007)*, 25–34. http://doi.acm.org/10.1145/1243441.1243452
8. Jeffrey P. Bigham, Craig M. Prince, and Richard E. Ladner. (2008). WebAnywhere: a Screen Reader On-the-Go. *Proceedings of the International Workshop on Web Accessibility (W4A 2008)*, 73–82. http://dx.doi.org/10.1145/1368044.1368060
9. Erin L. Brady, Yu Zhong, Meredith Ringel Morris, and Jeffrey P. Bigham. (2013). Investigating the Appropriateness of Social Network Question Asking as a Resource for Blind Users. *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW 2013)*, 1225–1236. http://dx.doi.org/10.1145/2441776.2441915
10. Erin Brady, Meredith Ringel Morris, and Jeffrey P. Bigham. (2015). Gauging Receptiveness to Social Microvolunteering. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2015)*, 1055–1064. http://doi.acm.org/10.1145/2702123.2702329
11. Erin Brady, Meredith Ringel Morris, Yu Zhong, Samuel White, and Jeffrey P. Bigham. (2013). Visual Challenges in the Everyday Lives of Blind People. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2013)*, 2117–2126. http://doi.org/10.1145/2470654.2481291
12. Jim A. Carter and David W. Fourne. (2007). Techniques to Assist in Developing Accessibility Engineers. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2007)*, 123–130. http://dx.doi.org/10.1145/1296843.1296865
13. Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. (2011). Associating The Visual Representation of User Interfaces with their Internal Structures and Metadata. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2011)*, 245–256. http://doi.acm.org/10.1145/2047196.2047228
14. Alan Cooper. (1995). *About Face: The Essentials of User Interface Design*. John Wiley & Sons, Inc., New York, NY.
15. Morgan Dixon and James Fogarty. (2010). Prefab: Implementing Advanced Behaviors Using Pixel-Based Reverse Engineering of Interface Structure. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2010)*, 1525–1534. http://doi.acm.org/10.1145/1753326.1753554
16. Morgan Dixon, James Fogarty, and Jacob O. Wobbrock. (2012). A General-Purpose Target-Aware Pointing Enhancement using Pixel-Level Analysis of Graphical Interfaces. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2012)*, 3167–3176. http://doi.acm.org/10.1145/2207676.2208734
17. Morgan Dixon, Gierad Laput, and James Fogarty. (2014). Pixel-Based Methods for Widget State and Style in a Runtime Implementation of Sliding Widgets. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2014)*, 2231–2240. http://doi.acm.org/10.1145/2556288.2556979
18. Morgan Dixon, Daniel Leventhal, and James Fogarty. (2011). Content and Hierarchy in Pixel-Based Methods for Reverse Engineering Interface Structure. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2011)*, 969–978. http://doi.acm.org/10.1145/1978942.1979086

19. Morgan Dixon, A. Conrad Nied, and James Fogarty. (2014). Prefab Layers and Prefab Annotations: Extensible Pixel-Based Interpretation of Graphical Interfaces. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2014)*, 221–230. http://doi.acm.org/10.1145/2642918.2647412

20. James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. (2011). Cracking the Cocoa Nut: User Interface Programming at Runtime. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2011)*, 225–234. http://doi.org/10.1145/2047196.2047226

21. W. Keith Edwards, Scott E. Hudson, Joshua Marinacci, Roy Rodenstein, Thomas Rodriguez, and Ian E. Smith. (1997). Systematic Output Modification in a 2D User Interface Toolkit. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 1997)*, 151–158. http://doi.org/10.1145/263407.263537

22. Leah Findlater, Alex Jansen, Kristen Shinohara, Morgan Dixon, Peter Kamb, Joshua Rakita, and Jacob O. Wobbrock. (2010). Enhanced Area Cursors: Reducing Fine-Pointing Demands for People with Motor Impairments. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2010)*, 153–162. http://doi.org/10.1145/1866029.1866055

23. Krzysztof Z. Gajos, Daniel S. Weld, and Jacob O. Wobbrock. (2010). Automatically Generating Personalized User Interfaces with SUPPLE. *Artificial Intelligence*, *174*(12–13), 910–950. http://doi.org/10.1016/j.artint.2010.05.005

24. Google Inc. Accessibility Scanner. https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor

25. Vicki L. Hanson and John T. Richards. (2013). Progress on Website Accessibility? *ACM Transactions on the Web*, *7*(1), 2:1-2:30. http://dx.doi.org/10.1145/2435215.2435217

26. Kyle J. Harms, Jordana H. Kerr, and Caitlin L. Kelleher. (2011). Improving Learning Transfer from Stencils-Based Tutorials. *Proceedings of the International Conference on Interaction Design and Children (IDC 2011)*, 157–160. http://dx.doi.org/10.1145/1999030.1999050

27. Yun Huang, Brian Dobreski, Bijay Bhaskar Deo, Jiahang Xin, Natã Miccael Barbosa, Yang Wang, and Jeffrey P. Bigham. (2015). CAN: Composable Accessibility Infrastructure via Data-Driven Crowdsourcing. *Proceedings of the Web for All Conference (W4A 2015)*, 1–10. http://dx.doi.org/10.1145/2745555.2746651

28. Amy Hurst, Scott E. Hudson, and Jennifer Mankoff. (2010). Automatically Identifying Targets Users Interact With During Real World Tasks. *Proceedings of the International Conference on Intelligent User Interfaces (IUI 2010)*, 11–20. http://dx.doi.org/10.1145/1719970.1719973

29. Amy Hurst, Jennifer Mankoff, Anind K. Dey, and Scott E. Hudson. (2007). Dirty Desktops: Using a Patina of Magnetic Mouse Dust to Make Common Interactor Targets Easier to Select. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2007)*, 183–186. http://doi.acm.org/10.1145/1294211.1294242

30. Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. (2009). Direct Manipulation Interfaces. *Human–Computer Interaction*, *1*(4), 311–338. http://dx.doi.org/10.1207/s15327051hci0104_2

31. IDI Web Accessibility Checker : Web Accessibility Checker. http://achecker.ca/checker/index.php

32. Yeongjin Jang, Chengyu Song, Simon P. Chung, Tielei Wang, and Wenke Lee. (2014). A11y Attacks: Exploiting Accessibility in Operating Systems. *Proceedings of the ACM Conference on Computer and Communications Security (CCS 2014)*, 103–115. http://doi.org/10.1145/2660267.2660295

33. Alex Jansen, Leah Findlater, and Jacob O. Wobbrock. (2011). From The Lab to The World: Lessons from Extending a Pointing Technique for Real-World Use. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI 2011)*, 1867–1872. http://doi.org/10.1145/1979742.1979888

34. Shaun K. Kane, Jeffrey P. Bigham, and Jacob O. Wobbrock. (2008). Slide Rule: Making Mobile Touch Screens Accessible to Blind People Using Multi-Touch Interaction Techniques. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2008)*, 73–80. http://doi.acm.org/10.1145/1414471.1414487

35. Shaun K. Kane, Chandrika Jayant, Jacob O. Wobbrock, and Richard E. Ladner. (2009). Freedom to Roam: A Study of Mobile Device Adoption and Accessibility for People with Visual and Motor Disabilities. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2009)*, 115–122. http://doi.org/10.1145/1639642.1639663

36. Shaun K. Kane, Meredith Ringel Morris, Annuska Z. Perkins, Daniel J. Wigdor, Richard E. Ladner, and Jacob O. Wobbrock. (2011). Access Overlays: Improving Non-Visual Access to Large Touch Screens for Blind Users. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2011)*, 273–282. http://dx.doi.org/10.1145/2047196.2047232

37. Shinya Kawanaka, Yevgen Borodin, Jeffrey P. Bigham, Darren Lunn, Hironobu Takagi, and Chieko Asakawa. (2008). Accessibility Commons: A Metadata Infrastructure for Web Accessibility. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2008)*, 153–160. http://doi.org/10.1145/1414471.1414500

38. Caitlin Kelleher and Randy Pausch. (2005). Stencils-Based Tutorials: Design and Evaluation. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*, 541–550. http://dx.doi.org/10.1145/1054972.1055047

39. Richard E. Ladner. (2015). Design for User Empowerment. *Interactions*, *22*(2), 24–29. http://dx.doi.org/10.1145/2723869

40. Walter Lasecki, Christopher Miller, Adam Sadilek, Andrew Abumoussa, Donato Borrello, Raja Kushalnagar, and Jeffrey Bigham. (2012). Real-Time Captioning by Groups of Non-Experts. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2012)*, 23–34. http://dx.doi.org/10.1145/2380116.2380122

41. Walter S. Lasecki, Phyo Thiha, Yu Zhong, Erin Brady, and Jeffrey P. Bigham. (2013). Answering Visual Questions with Conversational Crowd Assistants. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2013)*, 18:1–18:8. http://doi.org/10.1145/2513383.2517033

42. Jonathan Lazar, Daniel F. Goldstein, and Anne Taylor. (2015). *Ensuring Digital Accessibility through Process and Policy*. http://www.elsevier.com/books/ensuring-digital-accessibility-through-process-and-policy/lazar/978-0-12-800646-7

43. Lauren R. Milne, Cynthia L. Bennett, and Richard E. Ladner. (2014). The Accessibility of Mobile Health Sensors for Blind Users. *International Technology and Persons with Disabilities Conference Scientific/Research Proceedings (CSUN 2014)*, 166–175. http://doi.org/10211.3/133384

44. J. Morris and J. Mueller. (2014). Blind and Deaf Consumer Preferences for Android and iOS Smartphones. In *Inclusive Designing*. Springer International Publishing, Cham, 69–79. http://doi.org/10.1007/978-3-319-05095-9_7

45. Brad Myers, Scott E. Hudson, and Randy Pausch. (2000). Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction 7*, 3–28. http://doi.org/10.1145/344949.344959

46. Dan R. Olsen, Jr., Scott E. Hudson, Thom Verratti, Jeremy M. Heiner, and Matt Phelps. (1999). Implementing Interface Attachments Based on Surface Representations. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 1999)*, 191–198. http://doi.acm.org/10.1145/302979.303038

47. Elaine Pearson, Chrstopher Bailey, and Steve Green. (2011). A Tool to Support the Web Accessibility Evaluation Process for Novices. *Proceedings of the Conference on Innovation and Technology in Computer Science Education (ITiCSE 2011)*, 28–32. http://dx.doi.org/10.1145/1999747.1999758

48. André Rodrigues. (2015). Breaking Barriers with Assistive Macros. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2015)*, 351–352. http://dx.doi.org/10.1145/2700648.2811322

49. André Rodrigues and Tiago Guerreiro. (2014). SWAT: Mobile System-Wide Assistive Technologies. *Proceedings of the International BCS Human Computer Interaction Conference (British HCI 2016)*, 341–346. https://dl.acm.org/citation.cfm?id=2742991

50. Franziska Roesner, James Fogarty, and Tadayoshi Kohno. (2012). User Interface Toolkit Mechanisms for Securing Interface Elements. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2012)*, 239–250. http://doi.acm.org/10.1145/2380116.2380147

51. Daisuke Sato, Hironobu Takagi, Masatomo Kobayashi, Shinya Kawanaka, Chieko Asakawa, and Asakawa Chieko. (2010). Exploratory Analysis of Collaborative Web Accessibility Improvement. *ACM Transactions on Accessible Computing (TACCESS)*, *3*(2), 5:1-5:30. http://doi.acm.org/10.1145/1857920.1857922

52. Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. (2006). User Interface Façades: Towards Fully Adaptable User Interfaces. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2006)*, 309–318. http://doi.acm.org/10.1145/1166253.1166301

53. Hironobu Takagi and Chieko Asakawa. (2000). Transcoding Proxy for Nonvisual Web Access. *Proceedings of the ACM Conference on Assistive Technologies (ASSETS 2000)*, 164–171. http://doi.org/10.1145/354324.354371

54. Hironobu Takagi, Shinya Kawanaka, Masatomo Kobayashi, Takashi Itoh, and Chieko Asakawa. (2008). Social Accessibility: Achieving Accessibility Through Collaborative Metadata Authoring. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2008)*, 193–200. http://doi.acm.org/10.1145/1414471.1414507

55. Hironobu Takagi, Shinya Kawanaka, Masatomo Kobayashi, Daisuke Sato, and Chieko Asakawa. (2009). Collaborative Web Accessibility Improvement: Challenges and Possibilities. *Proceedings of the ACM Conference on Computers and Accessibility (ASSETS 2009)*, 195–202. http://doi.acm.org/10.1145/1639642.163967

56. Desney S. Tan, Brian Meyers, and Mary Czerwinski. (2004). WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*, 1525–1528. http://doi.acm.org/10.1145/985921.986106

57. Jacob O. Wobbrock, James Fogarty, Shih-Yen (Sean) Liu, Shunichi Kimuro, and Susumu Harada. (2009). The Angle Mouse: Target-Agnostic Dynamic Gain Adjustment Based on Angular Deviation. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2009)*, 1401–1410. http://doi.acm.org/10.1145/1518701.1518912

58. Jacob O. Wobbrock, Shaun K. Kane, Krzysztof Z. Gajos, Susumu Harada, and Jon E. Froehlich. (2011). Ability-Based Design: Concept, Principles and Examples. *ACM Transactions on Accessible Computing (TACCESS)*, *3*(3), 1–27. http://doi.org/10.1145/1952383.1952384

59. Aileen Worden, Nef Walker, Krishna Bharat, and Scott E. Hudson. (1997). Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 1997)*, 266–271. http://doi.acm.org/10.1145/258549.258724

60. World Health Organization. (2011). World Report on Disability. http://www.who.int/disabilities/world_report/2011/report/en/

61. Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. (2009). Sikuli: Using GUI Screenshots for Search and Automation. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2009)*, 183–192. http://doi.acm.org/10.1145/1622176.1622213

62. Jason Chen Zhao, Richard C. Davis, Pin Sym Foong, and Shengdong Zhao. (2015). CoFaçade: A Customizable Assistive Approach for Elders and Their Helpers. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2015)*, 1583–1592. http://doi.org/10.1145/2702123.2702588

63. Yu Zhong, Astrid Weber, Casey Burkhardt, Phil Weaver, and Jeffrey P. Bigham. (2015). Enhancing Android Accessibility for Users with Hand Tremor by Reducing Fine Pointing and Steady Tapping. *Proceedings of the Web for All Conference on (W4A 2015)*, 29:1-29:10. http://dx.doi.org/10.1145/2745555.2747277