Sinter: Low-Bandwidth Remote Access for the Visually-Impaired

Syed Masum Billah Donald E. Porter I.V. Ramakrishnan

Stony Brook University

{sbillah, porter, ram}@cs.stonybrook.edu

Abstract

Computer users commonly use applications designed for different operating systems (OSes). For instance, a Mac user may access a cloud-based Windows remote desktop to run an application required for her job. Current remote access protocols do not work well with screen readers, creating a disproportionate burden for users with visual impairments. These users' productivity depends on features of a specific screen reader, and readers are locked-in to a specific OS. The only current option is to run a different screen reader on each platform, which harms productivity.

This paper describes a framework, called Sinter, that efficiently and seamlessly supports remote, cross-platform screen reading, without modifying the application or the screen reader. Sinter addresses these problems with a platform-independent intermediate representation (IR) of a remote application's user interface (UI). The Sinter IR encapsulates platform-specific accessibility code on the remote system, facilitates development of additional accessibility features, and is simple enough to be reconstructed and read on any client platform. In the example above, Sinter allows a Mac-only reader to read remote Windows applications.

Sinter supports low-bandwidth, remote access to a wide range of applications, including Microsoft Word and Apple Mail, with both Windows and OS X clients and servers, as well as a web browser client. Sinter's IR-level programming model facilitates development of accessibility features and other enhancements, transparently to the remote application and reader. Sinter's latency is low enough for practical use, even over a relatively slow network connection.

1. Introduction

Computer users with visual impairments typically rely on special-purpose *assistive technologies (ATs)* to adapt an interactive graphical user interface (GUI), designed for a sighted user, to other senses—primarily hearing. For these users, the assistive technology of choice is a *screen reader*, such as NVDA [43], JAWS [26], SuperNova [24], Window-Eyes [31], or VoiceOver [16]. Screen readers serially narrate the textual content of the screen using a text-to-speech engine. Screen readers also include reader-specific *productivity enhancements* that improve the efficiency of navigation, such as touch and keyboard shortcuts, or reorganizing a two-dimensional content layout for more efficient audio navigation. People with low vision may also use magnifiers, such as MAGic [27] and ZoomText [10], which enlarge a small portion of the screen in addition to reading GUI contents.

For a sense of the size of this population, the World Health Organization reports there are 285 million people with vision impairments worldwide—39 million blind and 246 million with low vision [53]. In the U.S. alone, there are over 21 million Americans suffering from vision loss, and approximately 1.3 million Americans are considered legally blind [11]. These numbers are expected to grow as baby boomers develop vision impairments associated with age-related diseases, such as diabetes and macular degeneration.

This population relies on screen readers and other assistive technologies to use computers, but these technologies are locked into a single operating system (OS) platform—creating undue obstacles for user with visual impairments to leverage applications on multiple OSes. Simply put, current screen readers are neither interchangeable nor portable. A user's productivity hinges on features provided by a specific screen reader, and that screen reader only works on one OS. This lack of interoperability in screen readers stems from how accessibility interfaces are designed in modern OSes. The OS interfaces (APIs) by which screen readers get information seem easy for the OS developer to add to a working system, but, from the screen reader's perspective, are cumbersome, idiosyncratic, and, in our experience, even buggy. Screen readers are locked-in to specific platforms both because of platform-specific hacks to work around API problems, as well as differences in the underlying APIs. Examples of these Accessibility APIS include Microsoft's *MSAA & UI Automation* [37, 39], Apple's *NSAccessibility* [15], and GNOME's *ATK & AT-SPI* [29]. In total, an attempt to port a screen reader from one OS (say, Windows) to another (say,

EuroSys '16, April 18 - 21, 2016, London, United Kingdom
Copyright © 2016 held by owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4240-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2901318.2901335

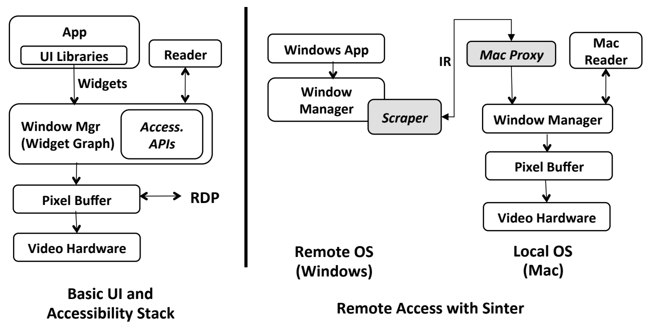


Figure 1. Default GUI and Accessibility stack (left panel), and illustration of Sinter (right panel). Remote Desktop Protocol (RDP) and other remote access technologies relay the hardware pixel buffer, whereas Sinter adds a layer of indirection at the semantic layer between the remote window manager and creates a proxy application which can be read by a local screen reader. We illustrate a Mac client reading a remote Windows system as an example, but other combinations are supported.

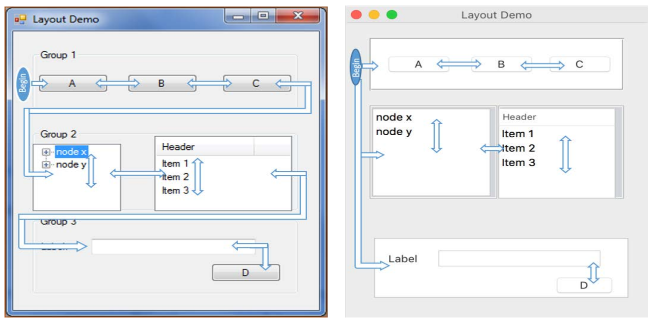


Figure 2. Screen reader navigation models in a typical Windows reader (left), and an OS X reader (right), indicated by arrows. Windows readers, like JAWS, use a "flat" navigation, which cycles through elements in a circularly-linked list. In contrast, VoiceOver on OS X navigates "hierarchically," effectively traversing a logical tree structure of GUI widgets.

Mac OS X) would involve rewriting nearly all of the tens of thousands of lines of code that interact with these OS interfaces. OS lock-in was acceptable 20 years ago, when a typical user had a single, preferred desktop OS, but modern users increasingly interact with multiple OSes, such as a Mac user developing a project for a Linux server inside of a virtual machine, or a physician using an iPad to access a sensitive medical records application running on a Windows remote desktop server.

Moreover, remote desktop protocols and virtualization techniques will not solve this problem, despite serving sighted users well. A typical approach to capturing the display of a virtual or remote machine is to emulate a graphics card frame buffer, which we call hardware virtualization and illustrate in Figure 1. The pixel values are collected from the frame buffer of the remote or virtual system, and redrawn as a simple bitmap on the local screen. Other GUI elements, such as the mouse, keyboard, and audio, are similarly captured and relayed at the hardware abstraction level. When a desktop screen reader encounters a virtual machine window or remote desktop client, the window is treated as a literal black box—the user must install a different screen reader inside the VM and relay audio output.

Requiring multiple screen readers creates several problems for disabled users. First, different OSes also present different logical navigation models of the user interface, illustrated on OS X and Windows in Figure 2. Users typically explore the screen to create a two-dimensional mental map of the elements, and changing this model is highly disorienting. Second, visually impaired users rely on AT-specific features, such as shortcuts, to quickly navigate a two-dimensional space designed for sighted users. Thus, switching screen readers is disruptive to users—comparable to the loss of muscle memory when a user switches from a QWERTY to a Dvorak keyboard layout. Third, high-quality screen readers are often expensive (JAWS retails for around $1,000 [26]), and limited to single-seat licenses; each remote or virtual system requires another license. Finally, blind "power users" can listen to the screen contents at 5$\times$ speed or higher [25]; relaying audio from a remote application can introduce unacceptable latency. These cost and usability issues on remote and virtual desktops create a significant obstacle for blind users.

This paper observes that, for screen readers, the point with the most similarity across platforms is the *semantics* of application GUIs—applications on every OS consist of similar *widgets* such as buttons, drop-down menus, and text fields.

This paper describes **Sinter**, illustrated in Figure 1, a framework for an unmodified screen reader on a local or host system to transparently read unmodified applications running on a remote or virtual system. Sinter **scrapes** snapshots of an application's GUI from a specific OS. Rather than using screen pixels, which lose semantic information essential for screen reading, Sinter collects a generic, least-common-denominator intermediate representation (IR) of high-level widgets, such as text boxes and buttons, potentially transforms this IR, and ultimately renders the snapshot of the application GUI using native widgets in a **proxy client** on a different platform. The screen reader on the client system can read this proxy as if the remote application were running locally. The engineering complexity of this approach is minimized, as one needs to write a proxy application for the Sinter IR only once per platform. Although we focus on cross-platform reading as a challenge application, Sinter can also be used for reading remote applications on the same OS (e.g., Windows-to-Windows reading).

This paper also describes an IR-level programming model, which allows users to write accessibility enhancements for applications, without modifying the application or screen reader. For instance, we demonstrate a most-frequently-used shortcut bar for navigating Microsoft Office ribbons, as well as retooling the Mac Finder interface to present a more familiar navigation experience for a blind Windows user. We believe that IR programming can also be applied for non-accessibility purposes, such as streamlining workflows across multiple GUI-based applications, but we leave this for future work.

This paper makes the following contributions:

- The design and implementation of the Sinter framework and IR, which makes screen readers interoperable with applications running on other OSes.
- *Transformations*—a simple, powerful abstraction for implementing accessibility enhancements transparently to the application and screen reader.
- Proof-of-concept client and remote implementations for Windows, OS X, and a web browser client. The browser client works with an in-browser reader [46], allowing non-visual, remote access from any system.
- A thorough evaluation of the Sinter prototype using a wide range of rich, desktop applications, including Microsoft Word and Apple Mail. We compare Sinter to relaying audio over remote desktop protocol, and find that the user latency is significantly lower and the bandwidth requirements are an order of magnitude lower. Sinter offers comparable latency and bandwidth to NVDARemote [6], but offers wider functionality, including reading across different OSes.

2. Background

This section reviews current OS accessibility and UI automation tools, and outlines the opportunities to encapsulate idiosyncrasies behind a common model, decoupling screen readers from the OS they were initially written for.

Modern OS windowing systems expose a set of accessibility APIs designed to facilitate assistive technologies (ATs) as well as GUI application testing and automation. These APIs expose a high-level model of the components of an application—similar in many respects to an HTML document object model (DOM) tree [51]. The objects in this tree represent all UI elements, such as text boxes and buttons. A *client* of an accessibility API generally does the following tasks:

- Walks the tree, extracting text or other information, and then rendering this for user, say as audio or in larger print.
- Monitors UI objects for updates.
- Simulates user interaction.

Object access. A client connects to the window manager or an application to obtain *accessible objects*, which expose methods to retrieve information about the UI elements in another program, via an IPC system such as Windows COM [41] or Linux D-Bus [45]. Most OSes define a generic abstract class for accessible objects; specific object types such as buttons, text, or menus can expose additional accessor methods or attributes. The degree of uniformity in accessor APIs across UI object types varies by platform.

When an application's user interface is composed only of standard UI elements provided by the OS, such as only using standard `user32` elements on Windows, the application does not need to do anything to be accessible. If an application includes customized UI elements, it needs to implement certain hooks in order to be accessible or automatable. The same is true for language runtimes with UI components, such as Java Swing. The language runtime must implement a *bridge* that maps language-specific abstractions onto an accessible UI object type, such as a button or drop-down menu.

Monitoring updates and filtering. Most OSes export a native notification mechanism, such as Windows Events, to update the client about changes to UI elements, such as the contents of a text box changing after the user presses a button. As we discovered, this mechanism can generate considerable overheads, and, in practice, this mechanism breaks encapsulation of internal housekeeping, such as garbage collecting occluded elements (§6). Sinter develops techniques to mitigate the shortcomings of native accessibility APIs.

Simulating user interaction. OSes also allow ATs to simulate actions such as a mouse click or keystroke. This is primarily intended to facilitate navigation by converting a shortcut key sequence to a series of cumbersome, visual navigation actions, such as walking a multi-layered drop-down menu. The underlying mechanism is roughly equivalent to injecting a hardware event.

Similarities across platforms. This work observes that, at a sufficiently high level, the models exported by each accessibility API are similar enough that it is tractable to translate one model into another. A significant contributor to this has been IBM's efforts to simply standardize accessibility APIs across platforms; the current standard is called iAccessible2 [35]. This effort demonstrates laudable leadership, but concerns such as backward-compatibility have caused each platform to retain significantly differing APIs. To our knowledge, no prior work has created a cross-OS accessibility bridge. The iAccessible2 effort has, however, brought the various systems close enough to make Sinter tractable.

3. Sinter Overview

Sinter consists of three major components, illustrated in Figure 1. On the remote or virtual system, a **scraper** mines the UI model from the remote or virtual system, and converts it into a common **Intermediate Representation (IR)** of the application. The initial IR is shipped to a client (local or host system), where a **proxy** converts the IR back to a native representation of these elements, which is suitable for use by a local screen reader.

The IR projects all possible UI objects from a given system onto a common, complete subset of UI elements. The

Platform	Scraper kLoC	Proxy kLoC
Windows	1.3	1.7
OS X	1.2	3.1
Web Browser	N/A	0.7

Table 1. Sinter's major components and lines of code.

IR is sufficiently expressive to capture any reasonable UI object, but minimal enough to be implemented easily using ubiquitous native widgets on any platform. The engineering work to reconstruct a model from, say Windows, on OS X using native GUI libraries is quite simple—thousands of lines of new code in total. The current IR design is mature enough to support rich desktop applications as complex as Microsoft Word. Moreover, adding interoperability with any new platform only requires writing a new IR scraper (for remote access) and native proxy application (for client access).

By operating on an abstract model of a user interface, a Sinter proxy can manipulate the model to improve accessibility, such as rearranging a cumbersome hierarchical navigation into a new navigation shortcut. This model of IR programming can also be used for non-accessibility purposes, such as changing the sizes of buttons for access from a device with a different form-factor (e.g., accessing a desktop application from a tablet), or streamlining a task that involves navigating a complex graphical hierarchy. Using IR programming beyond accessibility purposes is beyond the scope of this paper, and left for future work. Transformations are described in more detail in §4.2.

The Sinter protocol relays UI inputs from the proxy back to the scraper, and relays incremental changes in the UI from the scraper back to the proxy. As our evaluation shows, the Sinter protocol requires an order of magnitude lower network traffic than a protocol that transfers screen bitmaps or audio from a remote reader (§7).

Previous GUI frameworks, such as Java AWT, have designed least-common denominator APIs for implementing portable application graphics. Our goal is to describe the GUI of any running application in a format that any screen reader can understand—even if the application and reader are running on different OSes.

Table 1 summarizes the lines of code required to implement Sinter. Our current prototype includes a semantic scraper and proxy for both Windows and OS X, as well as an AJAX web browser proxy client. In general, these scrapers and proxies are only a few thousand lines of code, which indicates that such a wrapper would likely be equally simple to implement and deploy on additional platforms. As a point of comparison, the open source rdesktop [47] RDP client implementation is roughly 28 kLoC, counted by sloccount [52].

Project goals. The Sinter project has the following goals:

- Provide a seamless screen reading experience for users of remote applications with a visual impairment. Perfect visual fidelity of applications is a non-goal, although we do aim for usability by users without impairments, as a secondary goal.
- Compatibility with legacy applications and screen readers. In our experience, the effort to learn and customize a given reader is very high for a visually impaired user, and forcing a user to switch readers harms their productivity.
- Transparently improve accessibility of applications. Without any application or remote OS changes, Sinter can tailor the UI of applications to the user's preferences or for simpler audio navigation. For instance, we demonstrate an overlay of a most-frequently-used button "mega ribbon" for Microsoft Word (§7.4).

Although Sinter does not modify the OS, this is not a requirement. Section 6 explains several problems that would be better fixed in the OS.

Category	Types
OS	Application, Window, Menu, MenuItem, SplitPane, Generic
Basic	Graphic, Cell, Button, RadioButton, CheckBox, MenuButton, ComboBox, Range, Toolbar, Clock, Calendar, HelpTip
Arrangement	Table, Column, Row, ListView, Grouping, TabbedView, GridView
Navigation	TreeView, Browser, WebControl
Text	EditableText, RichEdit, StaticText

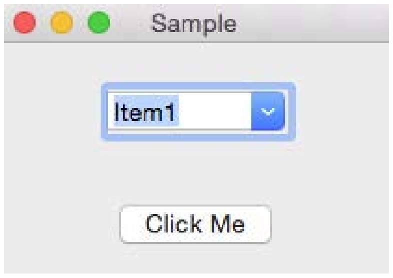
Table 2. Sinter's 33 IR object types, grouped by category.

4. Sinter Intermediate Representation

The Sinter intermediate representation (IR) encodes an application's UI tree in a generic, XML format. The UI includes 33 object types; we selected these as a least common substrate upon which more sophisticated UI elements can be composed or approximated. The IR standardizes several features that can vary by platform, such as placing coordinate `(0,0)` in the top left of the screen. The IR requires that each parent node's area must surround all children.

Figure 3 is a screen shot of a simple application with a Button and a ComboBox, as well as the matching IR. The IR also includes the three buttons in the upper left corner. The ComboBox includes a child button (the downward pointing triangle); when clicked, the IR would change to include the new selections visible in the drop-down window. The ID fields are used to efficiently communicate changes to the IR tree, between the scraper and the proxy.

The IR is designed for easy implementation on any platform. In our experience, a variant of each of the 33 types of objects (listed in Table 2) are available as native objects on all of our target platforms. Each object type can have a number of attributes. There are nine standard attributes, including an ID, the coordinates on the screen, a state (e.g., invisible, selected, clickable), and children. Some types also



```
<Application ID="9994" name="HelloWorld">
  <Window ID="1" value="Sample" height="150" width="212" left="727" top="80">
    <Button ID="2" name="close" height="16" width="14" left="734" top="83"/>
    <Button ID="3" name="minimize" height="16" width="14" left="754" top="83"/>
    <Button ID="4" name="maximize" height="16" width="14" left="774" top="83"/>
    <Button ID="5" name="Click Me" height="32" width="94" left="786" top="185"/>
    <ComboBox ID="6" value="Item1" height="26" width="99" left="785" top="126">
      <Button ID="7" name="" height="26" width="17" left="864" top="126"/>
    </ComboBox>
  </Window>
</Application>
```

Figure 3. A screenshot of a simple application for OS X (left), and simplified IR generated for this application (right).

have type-specific attributes; the Text types (EditableText, RichEdit, and StaticText) include fonts, bold, subscripts, and other decorations. There are 17 type-specific attributes.

We also strove for reasonable minimality: the current elements cannot be removed without losing functionality or are so ubiquitous that there is no advantage in removing them. These attributes and types represent an intersection of necessary functionality across platforms, not a union.

Our IR covers a significant fraction of the standard UI element types on Windows and OS X. Windows has 143 types of UIs (roles) as reported by NVDA [5]. Among them, 115 are mapped to Sinter's roles either directly, or in combination with one or more role-specific properties. In OS X, there are 54 types [4], and Sinter can map 45 of them either directly, or in combination with one or more properties. We have not encountered any of the remaining elements, although these could result in augmenting the IR. When Sinter encounters an element that is not explicitly mapped, including one of these unknown standard UI types or a custom element, it is mapped onto a `Generic` type in Table 2. As long as the native element supports a text accessor method, Sinter can at render its text, although some functionality may be lost. If the element does not export accessibility information, no screen reader can read it—with or without Sinter. Based on our experience rendering fairly complex applications, such as Microsoft Word, we expect only modest additions to the IR model will be needed to cover the remaining standard UI elements.

The examples in this paper focus on reconstructing textual and control elements, which are widely supported by platform accessibility frameworks. We believe the IR framework could also be easily extended to represent graphical decorations for buttons and regions, such as fill colors and images. However graphical applications, such as games, often have larger barriers to accessibility that are beyond the scope of this work. We note that NVDA does hook graphical libraries such as `gdi32.dll` [44]—an approach we could in future work to support some graphical applications.

4.1 Complex Objects

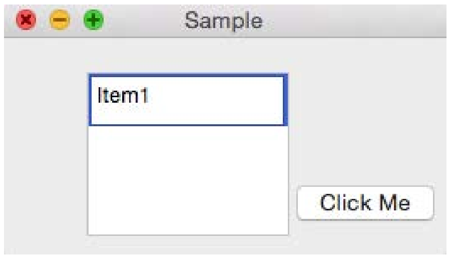
One challenge of having a simple IR is that a scraper must project complex UI objects onto one or more IR objects. In general, we encode complex objects in the IR by allowing multiple objects to occupy the same geometry, as children under a parent object. At any point, only one object is visible.

One simple example of a complex object is a ComboBox. A ComboBox is a combination of a text entry box, that, when active, also drops down a list of text suggestions or options, such as recently entered values. When a UI is initially mined from the application, a ComboBox has no children. Once the ComboBox is clicked on, the IR is potentially populated with a list of children. If a menu item is selected, or text is typed in the box, the proxy client is responsible to relay the event back to the remote scraper as if the entry occurred in the original ComboBox (by the parent's object identifier). Thus, shared geometry and object identifiers encode complex objects.

A more complex example is a **multi-personality** object, such as a Windows **Breadcrumb**. An example of a Breadcrumb is the navigation bar in Windows Explorer. The default view of a Breadcrumb shows the current working directory (e.g., `C:\foo\bar`). When the Breadcrumb is clicked, it behaves as a ComboBox—allowing text entry and selection of recent history. Finally, individual path components can also be selected as drop-down menus, when moused over. Within the UI tree, a BreadCrumb appears as a Grouping, and its child or children is the active personality (e.g., a ProgressBar (`Range`) or a `MenuButton`). Each personality change adds some children to the tree and removes others. When the Windows scraper detects a BreadCrumb, it replaces the ProgressBar with a `Grouping`, as other platforms cannot implement a semi-transparent progress bar, and the proxy displays the active child personality. Rather than forcing every platform to implement every widget, Sinter leverages dynamic object creation and destruction to project the active personality onto appropriate primitives.

4.2 IR Transformations

One powerful feature of Sinter is the ability to implement accessibility features at the IR level. Current Assistive Technologies (ATs) include productivity enhancements, such as shortcuts for faster navigation of visually-oriented menus, and decluttering web content for easier reading [8, 9]. Although it is convenient for some of these features to be bundled with a reader, there is no fundamental reason these need not have a modular, composable architecture.



```
nodeCB = find "./Window/ComboBox"
nodeBT = find "./Window/Button", name="Click Me"
chtype nodeCB List
nodeCB.height += 60
rm nodeCB.children
nodeBT.left += 100
```

Figure 4. Example IR transformation code and screenshot for Figure 3, which replaces the `ComboBox` with a `List`, and moves the `Click Me` button right.

Our insight is that many of these enhancements can be modeled as mutations of the IR graph, and that these mutations can be easily applied to the IR XML at the proxy client (or the scraper in a few cases). We call these **IR transformations**. This approach is practically useful because it does not require cooperation of application developers, who often neglect accessibility, nor does it require cooperation of the screen reader developer, who have a limited budget for features. Rather, this model empowers users with some programming skill to solve their own problems. The example transformations below are only tens of lines of code, whereas a similar decluttering effort in a web browser, without transformations, required tens of thousands of lines of code [8].

A transformation operates directly on the IR and is implemented in a simple language that extends XML XPath [19] rules with control flow (`while`, `for`, `if`) and simple commands for XML manipulation. These commands are listed in Table 3. Transformations run in an interpreter in the proxy or scraper, making the code platform-independent. Figure 4 illustrates simple transformation code on the IR in Figure 3, that replaces the `ComboBox` with a `List` and moves the `Click Me` button on the right to make space for the `List`, along with the resulting output image.

Transformations can implement features such as shortcuts in navigating a complex, two-dimensional menu or ribbon, into a simpler, flatter structure more appropriate for non-visual navigation. Multiple transformations can be applied to a given IR instance. Transformations may be applied only to specific applications, or on specific platforms. This subsection includes several examples that illustrate how transformations work, as well as their power in solving a number of UI portability problems.

Redundant Object Elimination. One common transformation prunes out invisible state from the tree. The `Grouping` type logically organizes related objects in the UI tree. Developers and UI frameworks often group related UI elements, leading to a significant amount of otherwise invisible wrappers that only introduce overhead for the proxy. Similarly, we eliminate redundant system-provided buttons (e.g., close and minimize) and scrollbars, when the client system provides similar elements by default.

Command	**Description**
`find xpath, [condition]`	Returns a `node` pointed by `xpath` and `condition`. Node attributes are accessed with “dot” syntax, such as `node.id`.
`chtype node type`	Changes the type of `node` to `type`.
`rm [-r] node`	Removes `node`, and its children with `-r`.
`mv [-c] node pnode`	Moves `node` under `pnode`; `-c` only moves children of `node`.
`cp [-r] node tnode`	Copies `node` to `tnode`. Children are also copied with `-r`.

Table 3. Sinter IR transformation syntax.

Topology Adjustment for Arrow Key Navigation. In most desktop window managers, the arrow keys navigate among elements visually. Web browsers generally break encapsulation by having the arrow keys navigate the topology of the DOM tree instead. For instance, the right arrow key generally moves the cursor to the next sibling node in the DOM tree, which may not necessarily be to the right in the visual coordinate plane. To address this problem, our browser client includes a transformation that adjusts the topology of the UI tree (and thus the DOM tree within the browser) to match the visual hierarchy. The basic approach to this problem is inserting table cells within a row around objects that should be horizontally aligned.

User Preferences. Sinter allows users to customize an application’s UI with transformations; this is particularly useful to correct cumbersome artifacts after automatic button resizing. From the user’s perspective, she manually moves buttons around and saves the preference for the future.

More generally, a transformation can replicate or move specific elements to different regions of the screen. As one proof of concept, we made a “mega-ribbon” for Word, which a user can use to avoid ribbon navigation for frequently-used buttons. The mega-ribbon is inserted on the left edge of the screen, and other items are shifted right. The mega-ribbon is automatically populated based on frequent actions.

Look-and-Feel Emulation. For an infrequent user of an unfamiliar system, say a Windows user infrequently accessing a Mac, a familiar navigation experience can flatten the learning curve. We implemented an example IR transformation that causes OS X’s Finder to have a similar look-and-feel (at least from the perspective of a screen reader) as using Windows Explorer (§7.4). We believe such a feature can improve productivity of infrequent cross-system users, which we will study in future work.

To Scraper	
Message	**Description**
list	Request a list of open processes and associated windows.
IR window	Request a complete IR tree of a window.
input	Send keyboard & mouse input, such as keystroke(s), coordinate of click, number of clicks, and click types.
action	Send action such as bring a window in the foreground, dialog open/close, menu open/close.
To Client Proxy	
Message	**Description**
IR full	Send complete IR.
IR delta	Send IR changes.
notification	Send System and User notifications.

Table 4. Messages in the Sinter's client/scraper protocol.

5. Proxy Client

The proxy client renders the remote application on the user's local machine. At a high level, the proxy takes the IR as input, and dynamically generates an application UI using native APIs. The network protocol between the proxy and remote scraper (§6) is asynchronous and stateful. The proxy relays user input and the scraper relays deltas to the IR. The protocol is summarized in Table 4.

When the client first connects to the scraper, the client queries the scraper for a list of all running applications on a given desktop session. In a production deployment, the connection would first be authenticated on the remote host, which we expect would be straightforward and orthogonal to the research goals of Sinter. The Sinter client can create a proxy application window for each remote application; for the sake of simplicity, we focus discussion on how the proxy represents a single application. Similarly, a user can run multiple proxies for applications on different remote servers.

Once a proxy is started for an application, the proxy requests a complete IR of the application from the scraper. Once the IR is received, the proxy first applies transformations to the tree (§4.2), then recursively walks the tree to render each object into equivalent native UI library primitives. After each object is created, the proxy monitors each object for user input, as appropriate.

The connection to the remote scraper is stateful, in order to incrementally update the client. The client initially requests a full IR of the current UI state. As the scraper observes changes to the UI, the scraper sends deltas to the client. Each object in the IR includes a unique identifier ("ID" in Figure 3); the scraper keeps the mapping of IR identifiers to remote OS abstractions only as long as the connection is open. If the connection is disconnected, this table is garbage collected. After disconnection the proxy cannot assume previous objects or IDs are still valid, and must completely re-read the application IR.

Once the initial rendering is complete, the proxy client executes in a simple event loop. As clicks, keystrokes, or other input is observed, messages are asynchronously relayed to the scraper. The proxy does not block for any responses, to ensure local responsiveness. For our motivating case of screen readers and other assistive technologies, this is an important feature, as the client screen reader can navigate and read elements on the screen concurrently with relaying keystrokes and updates over the network. As IR delta messages arrive from the scraper, the messages are inserted into the proxy's event queue. When a scraper delta is processed, transformations are applied if appropriate, and then the native UI state is updated appropriately.

Finally, the current prototype has the invariant that only one proxy may connect to each application at a time. We do not believe there is anything fundamentally difficult about keeping two proxy replicas in a consistent state with each other and the scraper, which we leave for future work.

5.1 Coordinate and Cursor Projection

Because transformations can perturb window geometry, Sinter includes the ability to project coordinates back to the original application layout. In contrast, most remote desktop systems require the client and server to agree on a geometry.

Most remote systems represent mouse movements and clicks in terms of screen geometry; when these events are relayed back to the scraper, the coordinates must be projected back from the client's possibly-transformed geometry. Otherwise, clicks intended for a specific button or other element may not be delivered to the correct component. Keyboard navigation and cursor placement face a similar problem: if a text box is re-wrapped at a client, simply relaying the "up" arrow can cause the cursor position to diverge between the proxy and remote applications.

To address this problem, each Sinter proxy constructs a **reverse coordinate map** for the screen, which maps client screen coordinates back to remote screen coordinates. If an IR object is repositioned or resized on the remote system or by a transformation during the course of execution, these offsets are adjusted appropriately on the proxy.

Similar challenges are faced with wrapping text on a smaller screen. One can either rewrap text for easier arrow key navigation (avoiding horizontal scroll bars), or preserve WYSIWYG navigation for typesetting. Rewrapping text is optional and configurable at the proxy client, depending on the user's goals for the document—reading versus composition and layout.

Similar to the reverse coordinate map, rewrapped text boxes must catch arrow key navigation events ("Up" or "Down") and manually set the cursor onto a different relative position in the remote text box. Each text element keeps a reverse character position mapping, and relays an equivalent series of arrow-key movements to the remote scraper.

5.2 Web Browser Client

To demonstrate the versatility of the Sinter IR, and for additional usability, we also implemented a JavaScript proxy that can run in a browser. Our in-browser proxy is compatible with in-browser screen readers. In-browser readers are becoming increasingly popular for users of public terminals or OSes with minimal support for native applications, such as Chrome OS. In-browser reading has the obvious limitation that it can only read web content, and cannot reach out to the local (or remote) desktop; the Sinter in-browser proxy expands the reach of in-browser reading.

We wrote a web front-end for a scraper using Ruby on Rails—Ruby version 2.0.0p481 on Rails version 4.2.0, running Rails default server. We verified that this client can be read by the ChromeVox [46] in-browser screen reader.

When a web browser connects to the Ruby web service, the browser loads a page that includes our JavaScript proxy. The JavaScript proxy then communicates with the scraper via the Ruby web service—initially requesting the complete IR, and then relaying events and IR deltas over `http`. Because the IR is implemented using XML, this proxy was a fairly straightforward AJAX-style application.

Because HTTP is stateless, the web server-side application maintains a connection to the scraper and buffers pending updates. The JavaScript client uses a cookie to collect updates since the last connection. If a client arrives for the same application with a different cookie, the session is ejected and a new session is created. A final difference between AJAX and the Sinter protocol is that it is more natural for a JavaScript client to poll the remote system, rather than have the server push updates to the client.

In order to balance responsiveness with bandwidth consumption, we use a bounded exponential back-off heuristic during idle periods. In other words, when the user has just interacted with the application or the server has relayed a change back to the application, the timer is set for 1 second. If the timer fires and there are no updates from either side, the timer interval is doubled. In a production system, the correct behavior at this point might be to gray the window and stop setting the timer until a user manually restores the connection. We leave tuning idle behavior for future work.

Our browser client case study demonstrates the generality of the Sinter IR and model. As more applications are moved to web variants, such as Google Docs and Microsoft Office 365, the Sinter scraper and browser proxy can be viewed as a simple way to make basic, web variants of legacy applications, for users with and without visual impairments.

6. Remote Scraper

The final component of Sinter is the scraper, which runs on the remote system to collect the IR and replay client events to the remote application. Much of the scraper's work is using platform-specific accessibility and automation APIs to collect a logical model (UI tree) of the application. The platform-specific objects must be translated into the IR, and then shipped to the client proxy. As part of the initial IR construction, the scraper also creates an internal model of the application, with references to the *accessible object* (i.e., the wrapper for the UI element used to retrieve accessibility metadata, including alternate text).

Once the initial IR is constructed, the scraper monitors the application for changes to any UI element. Notification APIs, such as SetWinEventHook, StructureChangedEvent, and PropertyChangedEvent on Windows [37, 39], and AXObserverAddNotification [14] on OS X, allow the scraper to register for updates from a given process. As messages arrive, the message includes a reference to the accessible object. The scraper then queries the state of the accessible object to identify the potential changes, and compares these changes against its internal model. The scraper relays any changes to the client. The scraper also maintains a table mapping IR-level, integer IDs onto system-specific identifiers or handles.

The scraper listens on a socket for messages from the proxy client. Most commonly, these messages require the scraper to synthesize keystrokes or mouse events on the remote system, using OS-specific routines such as `user32.mouse_event` or `user32.SendInput` on Windows, or `CGEventPost` on OS X.

The primary challenges we faced in building scrapers for both Windows and OS X were unreliable or idiosyncratic behavior in the accessibility APIs. The following subsections describe two categories of issues, and how the Sinter design robustly encapsulates them. Solutions initially designed for one OS, say Windows, proved useful on the other for similar, but not the same, reasons.

6.1 Reliable Object Identifiers

In order to calculate and relay incremental changes of the UI's logical state, the scraper must maintain a model of the UI objects and their states. Ideally, the OS would expose changes as a simple, batched delta with unique identifiers. Instead, current OS APIs return one notification with a handle to each object that is modified, removed, or added.

To minimize network traffic, each scraper maps these notifications onto an internal model of the UI, and calculates a more precise, batch delta to send to the proxy.

The challenge is that the handle included in this notification may not include a unique identifier (OS X), or the identifier for the same object may change (older versions of Windows). Newer Windows applications, compatible with the UIAutomation standard, do include such a robust identifier [40]. Older, legacy applications, may only be compatible with the older MSAA standard, which can change the OS-provided identifier without warning. In other words, for an MSAA-compatible application, a notifications for a value change event can arrive which refers to a completely new object ID (and accessible object wrapper). Upon further inspection, however, the new object's contents, such as placement, size, and text, are otherwise indistinguishable from a previous object already cached in the system. Moreover, the original object is no longer referenced in future notifications. This unreliable behavior is documented, but the root cause is not clearly explained. New object ID assignment most commonly happens when minimizing and restoring a window, which leads us to suspect this is breaking encapsulation of library-internal garbage collection.

In order to reliably track objects, both the Windows and OS X scrapers organize their model of objects into a hashmap. The hash function uses object fields that are expected to be stable, including the object type, as well as the objects' position in the UI graph. For instance, the OS X scraper's hash function also includes the path from the object to the nearest ancestor with a stable object ID. When an update occurs for an object, the stable fields are hashed and the scraper searches the appropriate bucket for matching IDs and likely matches (i.e., all stable fields match except for the OS-provided ID).

Likely matches are evaluated by walking both objects to a common ancestor object in the UI tree. From the ancestor, each node in the tree is checked that the contents are topologically and visually identical (same structure, same coordinates). If the path from the common ancestor is identical, and the nodes themselves are otherwise similar, the scraper assumes the objects are the same. Thus far, this strategy has proved sufficiently robust that we have not encountered any lost or mis-delivered updates.

Hashing UI elements by content and topology robustly maps OS-level notifications to IR-level identifiers, encapsulating platform-specific object ID behavior. Moreover, this technique dramatically reduced Sinter's bandwidth requirements compared to earlier prototypes.

6.2 Repeated, Verbose, and Lost Notifications

On both OS X and Windows, the update notification mechanism is rather verbose; we found that sending all updates by rote induces significant bandwidth costs. Both OSes also drop notifications if updates are not processed fast enough.

In the case of OS X, notifications for value changes are often raised multiple times for no clear reason. In Windows, notifications are not repeated, but the default mechanism to ask for all changes, called a structure change notification, is too verbose. For instance, one might use a structure change notification to tell when a tree is expanded. In principle, Windows includes a batch notification mechanism, but it is the application programmer's job to tell Windows when to use it [38], as opposed to Windows transparently setting a threshold of pending notifications to switch to batch mode; we have yet to find an application that enables this mechanism. Sinter includes a robust batch-notification wrapper based on several additional strategies.

First, on Windows we use domain-specific knowledge to identify a **minimal set of notification events** for structure changes, such as collapsing a tree or replacing the contents of a panel. Dialing down the verbosity of notifications made a first-order performance difference. For instance, the average time to scrape a tree expansion dropped from 600 ms down to 200 ms.

Second, we **re-batch rapid notifications**. Similar to interrupt handling with a separate "top half" and "bottom half", the scraper's notification handler marks an element as stale, and returns to the OS as quickly as possible. Once a burst of notification activity subsides, the scraper returns to the highest non-stale ancestor in the UI tree and re-queries all children. Reprobing also addresses the case where some notifications are lost after a large subtree is replaced.

Third, we use **periodic background scans**, during idle time, to detect any other stale objects in the model. With a reliable notification mechanism, this would not be necessary, but we have found several cases where the accessibility API simply does not deliver notifications, especially when an object is removed. In fact, the OS X API explicitly says that only certain object creation events can be treated as reliable [36], effectively forcing assistive software writers to eschew efficient caching for less-efficient, brute-force scans.

Finally, the scraper **filters** notifications for updates already reflected in its internal UI model. Notification filtering addresses both repeated value changes, and (effectively) repeated notification of new objects appearing in the UI tree, preventing needless network traffic to the proxy.

7. Evaluation and Case Studies

The evaluation of this work answers the following questions:

- How do Sinter's overheads compare to other remote access protocols? (§7.1)
- Is the system reasonably robust for cross-platform UI rendering? (§7.2)
- Are users with visual impairments able to use Sinter to access remote applications? (§7.3)
- Is the IR transformation model sufficient to implement non-trivial accessibility enhancements? (§7.4)

These questions are addressed with both quantitatively and qualitatively in the following subsections.

All Windows measurements use Windows 7 64-bit. The test system is a ThinkPad W550s, with a 4-core 2.6 GHz Core i7 CPU, 16 GB DDR 3 RAM, and a 250 GB SSD HDD. Our OS X measurements were collected on a MacBook Pro with a 8-core 2.8 GHz Core i7 CPU, 8 GB DDR3 RAM, and a 500 GB SSD HDD running OS X version 10.11. For network measurements, the machines were connected by a private Gigabit network.

7.1 Microbenchmarks

This subsection measures bandwidth and interaction latency of several remote access protocols. We use instrumented versions of Sinter, NVDARemote [6], and FreeRDP open-source RDP client [28]. We measure bandwidth using Microsoft RDP client for Mac version 8.0.26, erring on the side of a more mature client where instrumentation is not required. To measure latency, all clients collect the time when a keystroke is pressed on the local machine, as well as the time when the last packet is received following that keystroke. For measurements with a remote screen reader, we collect the time when the last audio packet is received by the local machine. In order to minimize unnecessary artifacts during RDP, we configure FreeRDP client to disable the wallpaper, menu-animation, aero theme, and font smoothing in Windows. Finally, the screen resolution of Windows is set to 1280×720.

NVDARemote works by intercepting text from the remote screen reader just before audio synthesis, and synthesizes audio at the client. We note that NVDARemote is not a fair comparison, as it does not work across screen readers, much less different OSes. NVDARemote also does not support mouse interactions. In contrast, Sinter and RDP are cross-platform.

We measured the response times to a set of tasks, scripted with Keyboard Maestro V.7.0.3. These tasks were all run from the local MacBook Pro, accessing the remote Windows Laptop. Because NVDARemote only works on Windows, we ran the client in a VirtualBox VM on the Macbook.

We measure three types of operations:

1. Rich text editing with Microsoft Word.
2. Using Windows Explorer and regedit to explore, expand, and collapse nodes in a directory tree. Each element in the tree is walked.
3. Updates to a list of items. We measure time to observe changes to the sorted process list in Task Manager, and time to select a different folder in Windows Explorer (and the right panel changes contents). When a view changes, the elements are traversed with the arrow keys.

Bandwidth. Table 5 measures the bandwidth requirements of each protocol. Across all of the traces, the bandwidth requirements, both in terms of data and packets sent, are an order of magnitude lower for Sinter than RDP. This stands to reason, as graphical updates simply require more data than logical updates to the UI model. To be clear, most users consider the costs of RDP acceptable, especially on a local area network. However, on a slower, wide-area network (WAN), or a pay-per-byte cellular link, a tenfold bandwidth reduction is considerable.

Sinter's bandwidth requirements are comparable to NVDARemote, although Sinter consistently requires fewer round-trips. The difference is how Sinter batches updates: when more of the batch is used locally by subsequent operations, as in Calculator, net bandwidth is lower. In contrast, NVDARemote will spend more round-trips and bandwidth exploring unchanged Calculator UI elements on the remote server. On the other extreme, Word has significant churn in its UI elements, and some of these updates are never used by the scripted client on Sinter; by lazily exploring UI elements, NVDARemote reduces bandwidth. We believe this difference can be bridged in future work with an adaptive heuristic that batches fewer updates when most of the batch is not used. In either case, Sinter and NVDARemote have comparably low bandwidth, but Sinter offers more functionality, such as IR transformations and cross-OS reading.

		Alone		With Reader	
App	**Protocol**	**KB**	**Packets**	**KB**	**Packets**
Calc	Sinter	47	171	47	171
	RDP	2,013	2,502	2,663	4,622
	NVDARmt	-	-	384	3,885
Explorer	Sinter	151	374	151	374
	RDP	1,104	2,223	4,457	7,484
	NVDARmt	-	-	121	749
Word	Sinter	223	513	223	513
	RDP	2,486	3,873	2,990	5,975
	NVDARmt	-	-	119	910

Table 5. Network traffic for several application traces for Sinter, RDP, and NVDARemote. The "Alone" columns show remote access; the "With Reader" columns add Apple's VoiceOver reading from the Sinter proxy, and RDP relaying audio from the NVDA Screen Reader. Lower is better.

Interaction Latency. To simulate different network types such as a WAN and 4G, we use Microsoft's Network Emulator for Windows Toolkit (NEWT) v2.1. The WAN is configured as having 30 millisecond round-trip delay, 20 Mbps download speed, and 5 Mbps upload speed, as measured over one of the author's home ISP. The round-trip delay in the 4G network is configured at 70 millisecond, and the download and upload speeds are set to 3.25 Mbps and 0.75 Mbps respectively—a current average as reported by FierceWireless [1].

Figure 5 shows cumulative distribution function (CDF) plots of the response times for each series of operations on simulated WAN and 4G network connections. The CDF plot shows the fraction of time that the system's response to user input was less than a given latency. Based on anecdotal discussion with blind users, we believe 500 milliseconds is the upper bound on acceptable response latency, indicated by the black vertical line in the figures.

For experiments on a WAN using RDP, 91–99% of requests receive responses in under 500ms. Adding audio over RDP drops this range to 7–70% of packets, indicating that most users will notice significant lag in relaying audio from a remote screen reader. In contrast, 99-100% of Sinter's packets are under 500ms, except for Word, at 90% (comparable to RDP's 92%). Word is unique in that there is a significant volume of dynamic control windows that change on the fly,

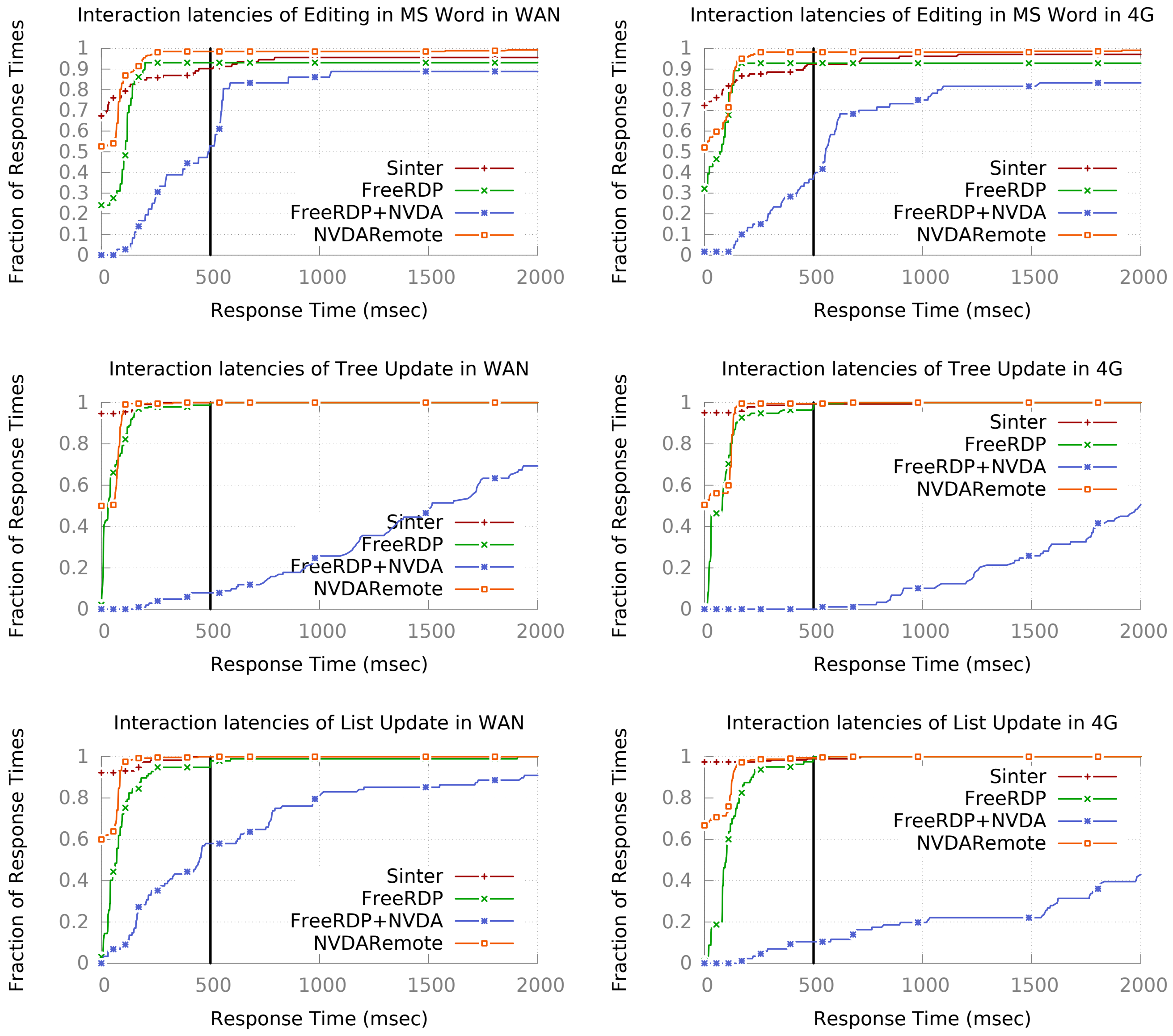


Figure 5. Cumulative distribution function (CDF) plots of interactive response times for three categories of operations over simulated WAN (left) and 4G (right) networks. The top row is text editing in Word. The second row is tree-based navigation in Windows Explorer and Regedit. The third row is updates to a list view in Task Manager and Windows Explorer.

generating additional traffic. These reductions in latency correlate with a reduction in synchronous network round trips.

In the case of the simulated 4G network, Sinter and RDP without audio are fairly comparable: all experiments range from 92–99% of requests being serviced in 500ms or less. Relaying audio over a 4G connection exacerbates the overheads of the approach, dropping the percent of requests serviced in under 500ms to 0–43%.

In most cases, Sinter latency is comparable to NVDARemote. Sinter services a higher fraction of interactions immediately from local state on Sinter for all workloads, and the worst-case latency is comparable in most cases. The exception is Word, yielding a trade-off with update batching, explained above, which we expect can be tuned with an adaptive batching heuristic.

In general, the performance of relaying audio is much worse for complex updates, like collapsing a tree or replacing a list view (the bottom four graphs). Most of these interactions have latencies above one second, except for list update on a WAN, for which more than 40% of requests take over 400 ms. The issue with remote audio synthesis is that each item explored introduces a synchronous round trip. In contrast, Sinter can read each item in the list from the local representation.

These results indicate that, except on a very fast network, relaying audio will not be usable. For good performance, the client must generate audio locally from a a more compact, logical representation. The generality of the Sinter model comes at little-to-no cost relative to simply relaying text for local speech synthesis, and can optimize a number of common cases and improve functionality.

7.2 Cross-Platform Remote Access

To demonstrate the relative maturity of the Sinter IR design, we show a range of application interfaces virtualized and displayed on platforms other than the one they were written for. The Windows applications accessed from OS X are illustrated in Figure 6, and include Microsoft Word, Windows Calculator, Windows Explorer, the Windows registry editor, and the DOS command line (`command.exe`). The Mac applications accessed from Windows are shown in Figure 7 and include Apple Mail, HandBrake (a media ripping and transcoding utility), Messages, Calculator, and Contacts. Figure 8 shows a proxy running in Google Chrome and accessing both Windows Explorer and command line.

This wide array of applications demonstrates that the Sinter framework is sufficiently robust for cross-platform UI rendering. As we have added support for each new application, the incremental effort has decreased, and the churn has mostly been within the scraper to translate increasingly arcane objects to the Sinter IR. That said, there is a long tail of additional applications to test, most notably highly-visual applications such as PowerPoint.

We tested the browser proxy for interoperability with the ChromeVox [46] in-browser screen reader. We were able to navigate all test applications by audio, indicating that these techniques can be composed.

We note that several of the screenshots need additional work to be usable by user without a visual impairment, such as the small text on buttons in our Word screenshot. We expect this can be addressed in future work by either optionally including images for the buttons, or using a transformation to adjust the layout to enforce minimal button and font sizes.

7.3 User Focus Group

We conducted a preliminary, IRB-approved, user focus group at the Lighthouse Guild in New York [2], with 21 subjects who are either blind (18/21) or have extremely low vision (3/21). We had them do several simple tasks with Sinter. In general, all of the 21 users responded positively to the user experience and would be interested in using Sinter again. We leave a larger-scale user study for future work, but these results indicate that our prototype has made a significant improvement to usable remote access.

7.4 IR Transformations

This section provides examples of the power of the IR programming model to implement additional accessibility support, transparently to applications. Here we list two substantial examples of IR transformations we implemented in a under one hundred lines of code each.

Mega-Ribbon. Figure 6 shows one form of automatic adjustment for simpler navigation of ribbons. Anecdotally, blind users have found ribbon navigation cumbersome. This screen shot shows Microsoft Word with an extra “mega ribbon” on the left. The mega ribbon shows up to the 10 most frequently used buttons for faster access. This transformation also shifts the text box to the right, and is implemented entirely transparently to Word. We expect the ability to streamline navigation with transformations will be of value to users with vision impairments.

Mac Finder with Windows Explorer Look-and-Feel. A significant fraction of blind desktop users use Windows [7]. If a blind user were asked to occasionally use a Mac for work, we expect that this transition would be simplified by masking navigation differences. Thus, we wrote a transformation that makes the navigation in Finder similar to navigating Windows Explorer, at least from the perspective of a screen reader, shown in Figure 9.

8. Related Work

This section reviews essential related work, grouped into remote access techniques, and assistive technology portability.

8.1 Remote Access

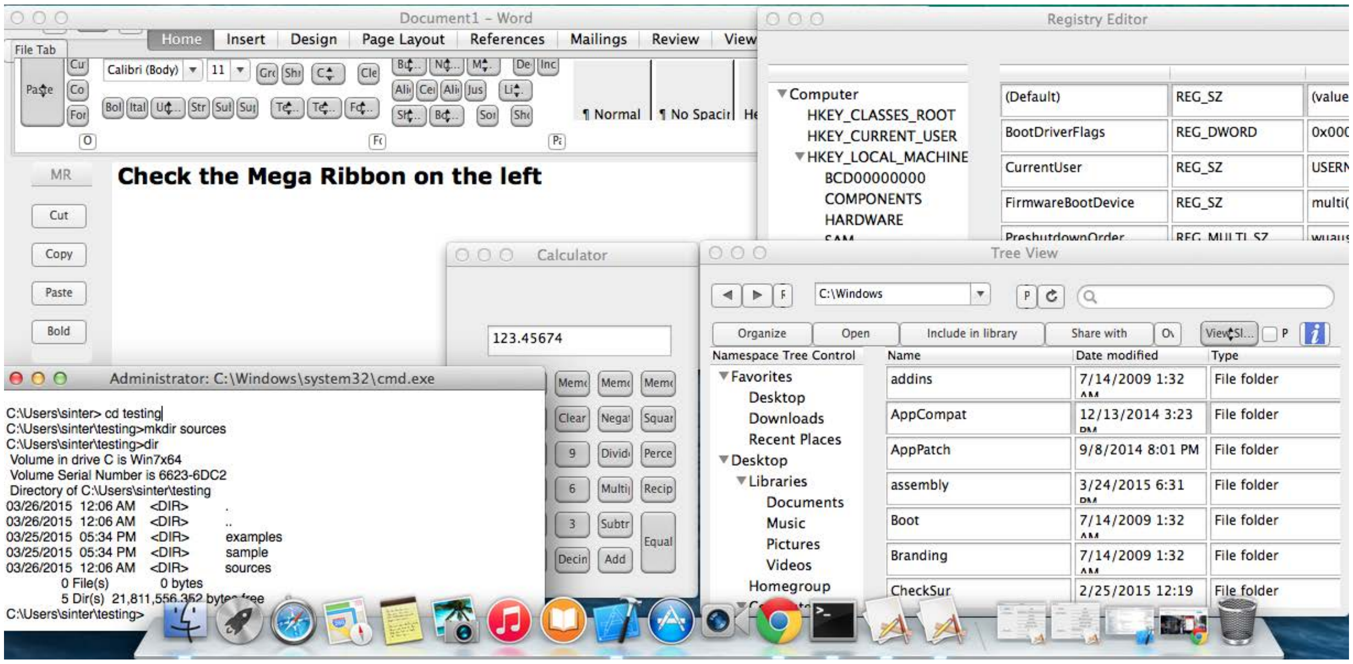
Remote Desktop Protocol (RDP) [42], ICA [32], VNC [48], PCoIP [50], and other protocols for remote access and virtual machines [13, 18, 34, 49], are based on “screen scraping”, where the bitmap in a virtual graphics frame buffer is collected and shipped over the network. RDP has extensible channels, which can be used for audio or even to export some advanced graphics, such as for Aero effects. These systems do not support screen reading from the client—only by exporting audio from the server.

The JAWS and NVDA screen readers have recently introduced remote access features, which essentially work by replaying navigation events remotely and then relaying a stream of text, which is then synthesized on the local computer [6, 21]. These only work with the same reader and OS on both ends. This paper demonstrates that an semantics-preserving, IR-based approach can generalize this approach across platforms with low bandwidth requirements.

THINC [18] retains some semantic information about bitmaps and shapes in the display to optimize thin client performance and minimize network traffic. pTHINC [34] can resize or zoom a display for a mobile device, but this work is pushed to the server (as in other remote access systems), not done on an IR. To our knowledge, no THINC variant addresses accessibility.

8.2 Portability of Assistive Technologies

In-browser readers, such as WebAnywhere [20] and ChromeVox [46] are the state of the art in fully-portable assistive

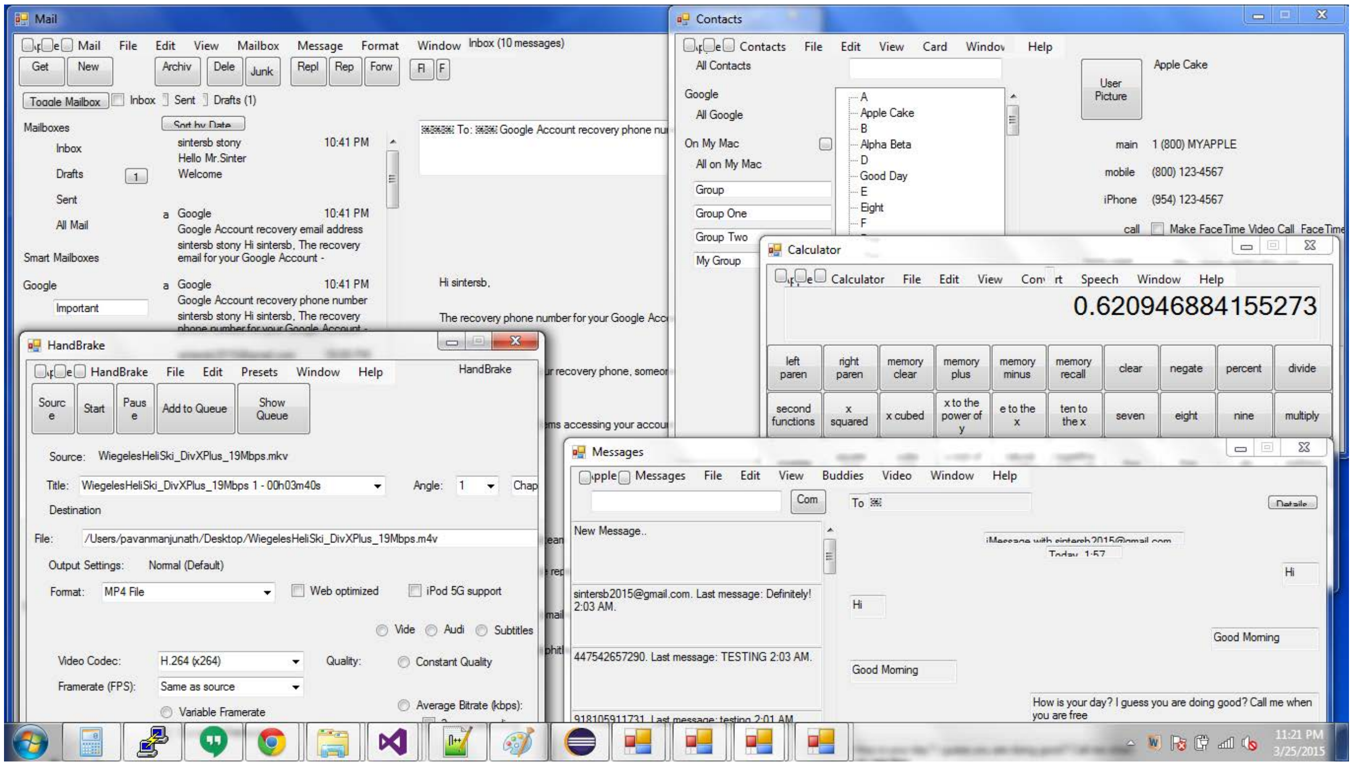


(Default)	REG_SZ	(value
BootDriverFlags	REG_DWORD	0x000
CurrentUser	REG_SZ	USERN
FirmwareBootDevice	REG_SZ	multi(
PreshutdownOrder	REG_MULTI_SZ	wuaus

Name	Date modified	Type
addins	7/14/2009 1:32 AM	File folder
AppCompat	12/13/2014 3:23 PM	File folder
AppPatch	9/8/2014 8:01 PM	File folder
assembly	3/24/2015 6:31 PM	File folder
Boot	7/14/2009 1:32 AM	File folder
Branding	7/14/2009 1:32 AM	File folder
CheckSur	2/25/2015 12:19	File folder

```
C:\Users\sinter> cd testing
C:\Users\sinter\testing>mkdir sources
C:\Users\sinter\testing>dir
 Volume in drive C is Win7x64
 Volume Serial Number is 6623-6DC2
 Directory of C:\Users\sinter\testing
03/26/2015  12:06 AM    <DIR>          .
03/26/2015  12:06 AM    <DIR>          ..
03/25/2015  05:34 PM    <DIR>          examples
03/25/2015  05:34 PM    <DIR>          sample
03/26/2015  12:06 AM    <DIR>          sources
               0 File(s)              0 bytes
               5 Dir(s)  21,811,556,352 bytes free
C:\Users\sinter\testing>
```

Figure 6. Windows applications, accessed remotely on an OS X client, using Sinter. Clockwise From the top left: Microsoft Word, the Windows registry editor (`regedit`) Windows Explorer, Windows Calculator, and the Windows command line (`cmd.exe`). Word also displays the mega ribbon on the left hand side, automatically saving the most frequently used buttons.



sintersb stony 10:41 PM
Hello Mr.Sinter
Welcome

Google 10:41 PM
Google Account recovery email address
sintersb stony Hi sintersb, The recovery email for your Google Account -

Google 10:41 PM
Google Account recovery phone number
sintersb stony Hi sintersb, The recovery phone number for your Google Account -

To: Google Account recovery phone nu

Hi sintersb,

The recovery phone number for your Google Acc

Apple Cake

main	1 (800) MYAPPLE
mobile	(800) 123-4567
iPhone	(954) 123-4567
call	Make FaceTime Video Call FaceTime

A
Apple Cake
B
Alpha Beta
D
Good Day
E
Eight
F

0.620946884155273

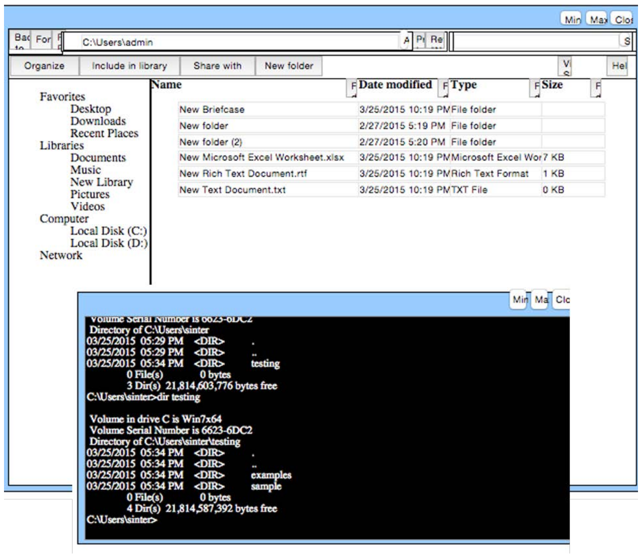
Source: WiegelesHeliSki_DivXPlus_19Mbps.mkv
Title: WiegelesHeliSki_DivXPlus_19Mbps 1 - 00h03m40s
Angle: 1
Destination
File: /Users/pavanmanjunath/Desktop/WiegelesHeliSki_DivXPlus_19Mbps.m4v
Output Settings: Normal (Default)
Format: MP4 File
Web optimized
iPod 5G support
Vide Audi Subtitles
Video Codec: H.264 (x264)
Quality: Constant Quality
Framerate (FPS): Same as source
Variable Framerate
Average Bitrate (kbps):

New Message..
sintersb2015@gmail.com. Last message: Definitely! 2:03 AM.
447542657290. Last message: TESTING 2:03 AM.
918105911731. Last message: testing 2:01 AM.

iMessage with sintersb2015@gmail.com
Today 1:57
Hi
Hi
Good Morning
Good Morning
How is your day? I guess you are doing good? Call me when you are free

11:21 PM
3/25/2015

Figure 7. Mac applications, accessed remotely on a Windows client, using Sinter. Clockwise from the top left: Apple Mail, Apple Contacts, Apple Calculator, Apple Messages, and HandBrake (media ripping and transcoding utility).



Name	Date modified	Type	Size
New Briefcase	3/25/2015 10:19 PM	File folder	
New folder	2/27/2015 5:19 PM	File folder	
New folder (2)	2/27/2015 5:20 PM	File folder	
New Microsoft Excel Worksheet.xlsx	3/25/2015 10:19 PM	Microsoft Excel Wor	7 KB
New Rich Text Document.rtf	3/25/2015 10:19 PM	Rich Text Format	1 KB
New Text Document.txt	3/25/2015 10:19 PM	TXT File	0 KB

```
 Volume Serial Number is 6623-6DC2
 Directory of C:\Users\sinter
03/25/2015  05:29 PM    <DIR>          .
03/25/2015  05:29 PM    <DIR>          ..
03/25/2015  05:34 PM    <DIR>          testing
               0 File(s)              0 bytes
               3 Dir(s)  21,814,603,776 bytes free
C:\Users\sinter>dir testing

 Volume in drive C is Win7x64
 Volume Serial Number is 6623-6DC2
 Directory of C:\Users\sinter\testing
03/25/2015  05:34 PM    <DIR>          .
03/25/2015  05:34 PM    <DIR>          ..
03/25/2015  05:34 PM    <DIR>          examples
03/25/2015  05:34 PM    <DIR>          sample
               0 File(s)              0 bytes
               4 Dir(s)  21,814,587,392 bytes free
C:\Users\sinter>
```

Figure 8. Windows Explorer and Command Line, accessed remotely by a Chrome browser client.

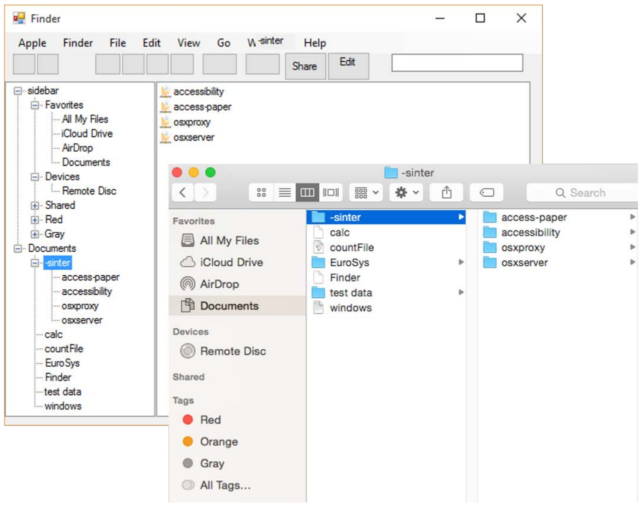


Figure 9. Mac Finder converted to the look-and-feel of Windows Explorer using an IR transformation. Original in foreground, Windows look-and-feel in background.

technologies (AT)—a screen reader that runs entirely in the web browser. These in-browser readers directly access the browser's DOM tree for content. Naturally, their utility is limited to reading web content, and is not designed to read local or remote desktop applications; our in-browser proxy client extends these solutions to desktop applications.

Several projects, including Adobe Reader [30] and Mozilla Firefox [3] include platform-specific wrappers for the *application* to program against a unified accessibility API. On a given platform, this wrapper maps Mozilla's API onto the native accessibility API. Sinter addresses a complementary problem: making unmodified screen readers inter-operable with different platforms. Gonzalez and Reid proposed a similar wrapper architecture for *implementing* portable screen readers [30], but this proposal only generalizes an old version of Windows screen reading APIs, and, to our knowledge, was never implemented.

Prefab [22, 23] applies computer vision to reconstruct semantic knowledge, such as text and tables, from a screen image. Similar techniques have been used to work around deficiencies in accessibility APIs [33], but this is more expensive and error-prone than retaining the semantic information.

Cider [12] allows iOS and Android apps to run on the same platform, by introducing wrappers between different system services. Apportable [17] automatically cross-complies iOS apps written in Objective-C to native Android apps. In both cases, it is unclear whether the platform correctly supports screen readers, a source code is not available for either project. Sinter's IR strategy could simplify this engineering effort considerably.

9. Conclusion

Sinter demonstrates that semantic virtualization of user interfaces is feasible to implement, and can more efficiently address the needs of users with visual impairments than relaying hardware-level events, such as video or audio. Sinter is sufficiently robust to replicate and read a wide array of application user interfaces on several platforms, including Windows, OS X, and in a browser. We believe the IR transformation model has potential to spur innovation in productivity enhancements for users in general, by transparently and easily modifying user interfaces.

This paper also identifies a number of cases in which current accessibility APIs create needless difficulties for ATs. Specifically, most OSes do not provide reliable or batched notifications of changes and frustrate attempts to cache expensive information. Sinter encapsulates these problems, but could be more efficient if OSes addressed these shortcomings or adopted the Sinter model directly.

10. Acknowledgments

We thank the anonymous reviewers and our shepherd, Franziska Roesner, for their insightful comments on earlier drafts of the work. We thank Glenn Dausch for invaluable advice and feedback on the Sinter prototype. Gautham Reddy Kunta, Tabish Ahmad, Pavan Manjunath, and Siva Prasad Reddy contributed to the Sinter implementation.

This research was supported in part by NSF grants IIS-1218570, IIS-1447549, CNS-1149229, CNS-1161541, CNS-1228839, CNS-1405641, CNS-1408695, CNS-1526707, and VMware.

References

[1] Fiercewireless. `http://www.fiercewireless.com/tech/special-reports/3g4g-wireless-`

network-latency-how-did-verizon-att-sprint-and-t-mobile-compa-0.

[2] Lighthouse guild. http://www.lighthouseguild.org/.

[3] Mozilla accessibility architecture. https://developer.mozilla.org/en-US/docs/Mozilla/Accessibility/Accessibility_architecture.

[4] Nsaccessibility protocol reference. https://developer.apple.com/library/mac/documentation/AppKit/Reference/NSAccessibility_Protocol_Reference/index.html#//apple_ref/doc/constant_group/Roles.

[5] Nvda control types. https://github.com/nvaccess/nvda/blob/master/source/controlTypes.py.

[6] Nvda remote brings free remote access to the blind. http://nvdaremote.com/.

[7] Screen reader user survey #6 results. http://webaim.org/projects/screenreadersurvey6/.

[8] F. Ahmed, Y. Borodin, Y. Puzis, and I. V. Ramakrishnan. Why read if you can skim: towards enabling faster screen reading. In *Web for All Conference (W4A'12)*, 2012.

[9] F. Ahmed, Y. Borodin, A. Soviak, M. A. Islam, I. V. Ramakrishnan, and T. Hedgpeth. Accessible skimming: faster screen reading of web pages. In *ACM Symposium on User Inetrface Software and Technology (UIST'12)*, pages 367–378, 2012.

[10] Ai Squared. Zoomtext magnifier. http://www.aisquared.com/zoomtext/more/zoomtext_magnifier.

[11] American Foundation for the Blind. Facts and figures on american adults with vision loss. http://www.afb.org/Section.asp?SectionID=15&TopicID=413&DocumentID=4900, 2013.

[12] J. Andrus, A. Van't Hof, N. AlDuaij, C. Dall, N. Viennot, and J. Nieh. Cider: Native execution of ios apps on android. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 367–382, 2014.

[13] Apple. Airplay—play content from ios devices on appletv. https://www.apple.com/airplay/.

[14] Apple. Notification api. https://developer.apple.com/library/mac/documentation/ApplicationServices/Reference/AXUIElement_header_reference/index.html#//apple_ref/c/func/AXObserverAddNotification. [Online; accessed 22-March-2015].

[15] Apple. Nsaccessibility. https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ApplicationKit/Protocols/NSAccessibility_Protocol/index.html#//apple_ref/doc/constant_group/Focus_change_notifications. [Online; accessed 11-Feb-2015].

[16] Apple Inc. Voiceover. https://www.apple.com/accessibility/osx/voiceover/.

[17] Apportable. http://www.apportable.com/.

[18] R. A. Baratto, L. N. Kim, and J. Nieh. Thinc: A virtual display architecture for thin-client computing. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 277–290, 2005.

[19] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. Xml path language (xpath). *World Wide Web Consortium (W3C)*, 2003.

[20] J. P. Bigham, C. M. Prince, and R. E. Ladner. Webanywhere: a screen reader on-the-go. In *W4A '08: Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)*, pages 73–82, New York, NY, USA, 2008. ACM.

[21] Citrix. Using common accessibility & adaptability software with citrix xenapp. https://www.citrix.com/content/dam/citrix/en_us/documents/support/using-common-accessibility-and-adaptability-software-with-citrix-xenapp.pdf.

[22] M. Dixon and J. Fogarty. Prefab: Implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1525–1534, New York, NY, USA, 2010. ACM.

[23] M. Dixon, A. Nied, and J. Fogarty. Prefab layers and prefab annotations: Extensible pixel-based interpretation of graphical interfaces. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 221–230, New York, NY, USA, 2014. ACM.

[24] Dolphin. Supernova screen reader. http://www.yourdolphin.com/productdetail.asp?id=1.

[25] R. D. Fields. Why can some blind people process speech far faster than sighted persons? http://www.scientificamerican.com/article/why-can-some-blind-people-process/. [Online; accessed 11-Feb-2015].

[26] Freedom Scientific. Jaws screen reader. http://sales.freedomscientific.com/Category/11_1/JAWS%C2%AE_Screen_Reader.aspx. [Online; accessed 11-Feb-2015].

[27] Freedom Scientific. Magic screen magnification software. http://www.freedomscientific.com/products/lv/magic-bl-product-page.asp.

[28] FreeRDP. Freerdp. https://github.com/FreeRDP/FreeRDP.git. [Online; accessed 5-March-2016].

[29] GNOME Accessibility Architecture (ATK and AT-SPI). https://accessibility.kde.org/developer/atk.php. [Online; accessed 11-Feb-2015].

[30] A. Gonzalez and L. G. Reid. Platform-independent accessibility api: Accessible document object model. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, W4A '05, pages 63–71, New York, NY, USA, 2005. ACM.

[31] GW Micro. Window-eyes. http://http://www.gwmicro.com/Window-Eyes/.

[32] J. Harder and J. Maynard. Technical deep dive: Ica protocol and acceleration. http://s3.amazonaws.com/legacy.icmp/additional/ica_acceleration_0709a.pdf.

[33] A. Hurst, S. E. Hudson, and J. Mankoff. Automatically identifying targets users interact with during real world tasks. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, IUI '10, pages 11–20, New York, NY, USA, 2010. ACM.

[34] J. Kim, R. A. Baratto, and J. Nieh. pTHINC: A Thin-client Architecture for Mobile Wireless Web. In *Proceedings of the 15th International Conference on World Wide Web (WWW)*, pages 143–152, 2006.

[35] Linux Foundation. IAccessible2 API Version 1.3. http://accessibility.linuxfoundation.org/a11yspecs/ia2/docs/html/index.html.

[36] Mailing Lists: Apple Mailing Lists. Philosophy behind ui element destroyed notifications. http://lists.apple.com/archives/accessibility-dev/2004/Jun/msg00007.html.

[37] Microsoft. Microsoft Active Accessibility: Architecture. https://msdn.microsoft.com/en-us/library/windows/desktop/dd373592(v=vs.85).aspx. [Online; accessed 15-March-2015].

[38] Microsoft. Structurechangetype enumeration. https://msdn.microsoft.com/en-us/library/windows/desktop/ee671618(v=vs.85).aspx.

[39] Microsoft. UI Automation Overview. http://msdn.microsoft.com/en-us/library/ms747327.aspx.

[40] Microsoft. Use the AutomationID Property. https://msdn.microsoft.com/en-us/library/aa349646(v=vs.110).aspx. [Online; accessed 20-March-2015].

[41] Microsoft. What is COM? https://www.microsoft.com/com/default.mspx.

[42] Microsoft. Remote desktop protocol: Basic connectivity and graphics remoting specification. http://msdn.microsoft.com/en-us/library/cc240445%28v=prot.10%29.aspx, 2010.

[43] NVAccess. Nv access: Home of the free nvda screen reader. http://www.nvaccess.org/.

[44] Gdi hooking for nvda coming soon, i think. http://klango.net/en/forum/thread/tid/203831.

[45] J. Palmieri. Get on d-bus. *Red Hat Magazine*, January 2005.

[46] T. Raman, C. Chen, D. Mazzoni, R. Shearer, C. Gharpure, J. DeBoer, and D. Tseng. Chromevox: A screen reader built using web technology. *Google Inc*, 2012.

[47] rdesktop: A remote desktop protocol client. http://www.rdesktop.org/.

[48] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual network computing. *Internet Computing, IEEE*, 2(1):33–38, Jan 1998.

[49] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. The interactive performance of slim: A stateless, thin-client architecture. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 32–47, 1999.

[50] Teradici. Pc-over-ip technology explained. http://www.teradici.com/pcoip-technology.

[51] w3 Consortium. Document object model (dom). http://www.w3.org/DOM/.

[52] D. Wheeler. Sloccount, 2009.

[53] World Health Organization. Visual impairment and blindness. http://www.who.int/mediacentre/factsheets/fs282/en/, 2012.