XDBrowser 2.0: Semi-Automatic Generation of Cross-Device Interfaces

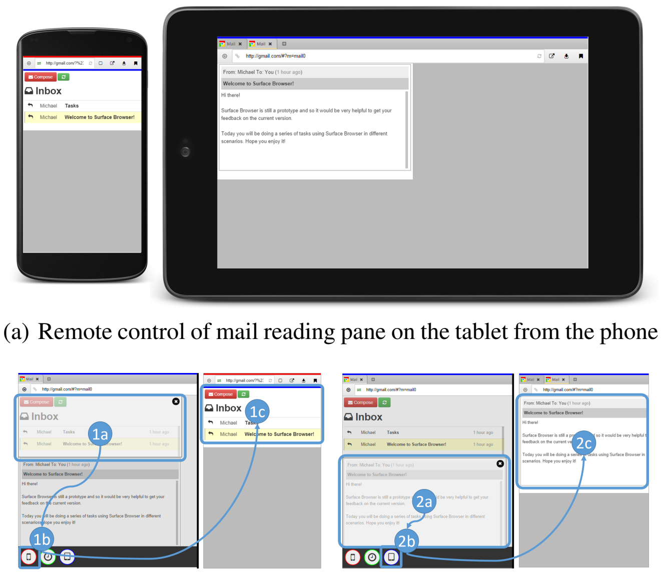
Michael Nebeling
University of Michigan School of Information
nebeling@umich.edu

ABSTRACT
Several recent studies have highlighted the need to support parallel usage of multiple devices for cross-device use. Yet, most interfaces today are still designed for single-device use and require re-authoring to enable cross-device interaction. This paper presents two studies to inform the design of a new web browser with support for semi-automatic generation of cross-device interfaces. Based on the results of a recent study in which users manually customized web pages for cross-device use, our first study elicits from users how they might want to trigger popular cross-device patterns to transform single-device designs with relatively little effort. Our second study then examines how the emerging design patterns could be applied to the Alexa top 50 sites from 10 different genres. Based on these studies, we design semi-automatic techniques for page segmentation and distribution between multiple devices that can work on many existing web sites and require only minimal user input to switch between different cross-device designs. Finally, we discuss possible extensions to the Chrome web browser to make the techniques available for a wide range of desktop, mobile, and wearable devices, and successfully test them on popular web sites.

Author Keywords
distributed user interfaces; cross-device interaction; semi-automatic page segmentation.

ACM Classification Keywords
H.5.2. Information interfaces and presentation: User Interfaces. – Screen design.

INTRODUCTION
State-of-the-art web browsers have added support for keeping the browser history, bookmarks and settings in sync so that users can use multiple personal devices for browsing the web. This partly addresses the need for more seamless multi-device interaction identified in earlier studies on information workers and cross-device web use [11, 20, 22, 29]. More recent studies [26, 32] find an increased need to better support parallel device usage so that users can flexibly distribute tasks between devices, taking into account both device capabilities and user preferences. However, there is no native browser support for using multiple devices in parallel. Rather, special cross-device development toolkits and major modifications to existing web interface code are required [12, 17, 27, 40].



(a) Remote control of mail reading pane on the tablet from the phone

(b) Cut and paste inbox on phone (c) Cut and paste mail on tablet

Figure 1. Re-authoring required in XDBrowser [26] to distribute parts of a mail interface between phone and tablet. This paper presents two studies around semi-automatic generation of such cross-device versions.

The XDBrowser project investigates how web browsers can be improved to better support parallel multi-device usage. A first version of XDBrowser [26] implemented end-user customization tools allowing users to re-author existing web pages so that they can be distributed and synchronized between multiple devices (Figure 1). XDBrowser was then used in an end-user customization study that produced a total of 144 multi-device designs for five popular web applications, leading to seven mobile multi-device patterns. While that version of XDBrowser proved particularly useful for that study, still significant manual re-authoring was required. Given the patterns, we ask how cross-device design can be automated.

Providing automated support for customization could be beneficial to users for several reasons. First, it would avoid the need for a separate customization mode and for mode switching to be able to customize pages for cross-device use. Second, users could browse new pages they have not visited before, without having to first customize them for cross-device use if they want to benefit from XDBrowser. Third, users could just choose from cross-device designs and switch between them depending on the type of page and browsing task.

However, there are two main obstacles. First, while we know *what* cross-device patterns users want and that they need to able to switch patterns for different tasks, we do not know *how* users would want to do that. Second, while we know that pages need to be split into the elements relevant for each pattern before they can be distributed, this problem has been sidestepped until now, either by having users manually select page elements [26, 14, 27] or by having developers manually annotate the respective HTML DOM nodes [12, 17, 40].

Taking the work by Nebeling and Dey [26] further, this paper explores a mixed-initiative approach to cross-device interface design based on existing cross-device patterns and new semi-automatic support. We believe that it is possible to design semi-automatic support that does not require cross-device re-authoring but still leaves end-users in control, and that this opens up a rich research space that the community can explore together. To make a start, this paper presents two studies that investigate such an approach for existing web interfaces:

1. a first study (N=16) on user-defined interactions in three mobile settings that make it possible for end-users to easily activate and switch between cross-device patterns,
2. a second study (N=3) on user-defined segmentations of popular web pages based on existing cross-device patterns to test the extent to which these patterns can be automated.

Informed by both studies, we also present a new XDBrowser prototype that can semi-automatically generate cross-device versions of existing pages by extracting and distributing page elements relevant for each pattern, and a proof-of-concept implementation for existing web browsers on desktop, mobile and wearable devices. However, we see the resulting system as a proof of concept—the studies are the main contribution.

We start in the next section with an overview of the existing XDBrowser and the seven multi-device patterns from [26], before detailing our new studies' designs and results.

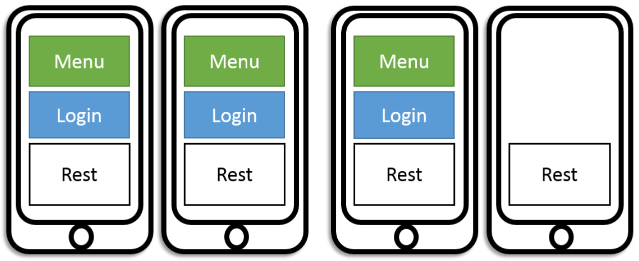
XDBROWSER + PATTERNS OF CROSS-DEVICE WEB USE

XDBrowser builds on the early notion of *multibrowsing* developed by Johanson et al. [19], allowing web pages to be sent to, and opened from, multiple devices. The first version of XDBrowser had the goal of enabling end-users to re-author web pages for cross-device use as they view them directly in the browser. Inspired by WinCuts [36], it provided a rectangular selection tool for users to frame arbitrary regions in the web page that they could then copy or move between devices, without the need for HTML DOM-level coding (Figure 1).

XDBrowser supports synchronization of both the view and the input on multiple devices. This is based on a record-and-replay mechanism capturing four types of events: *(1)* click events in the document via the mouse or finger, *(2)* scroll events manipulating the viewport, *(3)* form input events and *(4)* updates of the URL. This level of support is sufficient for users to control the browser tabs, page viewports, view state, and user input on all connected devices. More advanced device migration techniques able to preserve page-internal state and server-side sessions [14, 28] were out of scope.

An end-user customization study with 15 participants probed desirable multi-device page distributions when using a phone and tablet in parallel [26]. Five applications representing popular web browsing activities were examined and the study produced a total of 144 multi-device designs. These fall into three groups of cross-device patterns (Figures 2 to 4).

1. Copy and Sync Patterns: Duplicate/Extend

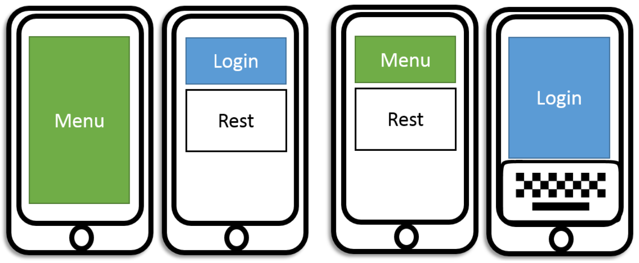


(a) Duplicate pattern (b) Extend pattern

Figure 2. Copy and sync patterns

The first group consists of patterns in which the page is copied and fully synchronized between devices. The *duplicate* pattern assigns equal roles to devices, duplicates the entire page, and keeps the view fully synchronized between devices. The *extend* pattern (Figure 2(b)) replicates page elements on the other device so that parts of the page extend across devices.

2. Move and Sync Patterns: Remote-Control/View+Input



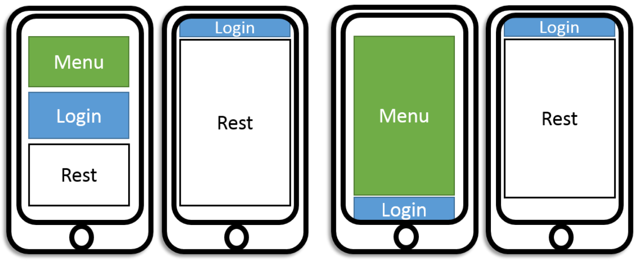
(a) Remote-control pattern (b) View+input pattern

Figure 3. Move and sync patterns

The second group consists of patterns that move parts of the page to other devices and sync all parts between devices as though they formed a single page. First, the *remote-control* pattern (Figure 3(a)) keeps controls such as navigation links and buttons on one device and moves the other elements over. Second, the *view+input* pattern (Figure 3(b)) assigns an input-only role to one device, keeps form elements such as input fields and selection lists on it, and moves the others.

3. Async Patterns: Overview+Detail/Split/Single

The last group consists of patterns of asynchronous device usage. The *overview+detail* pattern (Figure 4(a)) keeps the overview on one device but zooms the view on the other for details. The *split* pattern (Figure 4(b)) uses the same page on both devices but shows different views between devices.



(a) Overview+detail pattern (b) Split pattern

Figure 4. Async patterns

Finally, the *single* pattern loads the page on only one device and does not use the second device. Technically, this is not a cross-device pattern but was included for completeness.

While knowing the cross-device patterns was a first important result for designing more automatic cross-device interface support, there were still several open questions.

First, just for the five web pages used in the study, users spent roughly one hour designing cross-device versions. Re-authoring pages required switching between browsing and customization modes, executing a series of page manipulations, and switching between devices. We wanted to reduce this effort by introducing more automatic support.

However, participants' preferences for patterns varied not only between the five tested applications, but also depending on the tasks carried out with each application. Therefore, users wanted to activate patterns with little effort and also choose different patterns for different distributions of the same interface. A fully automatic approach was not feasible.

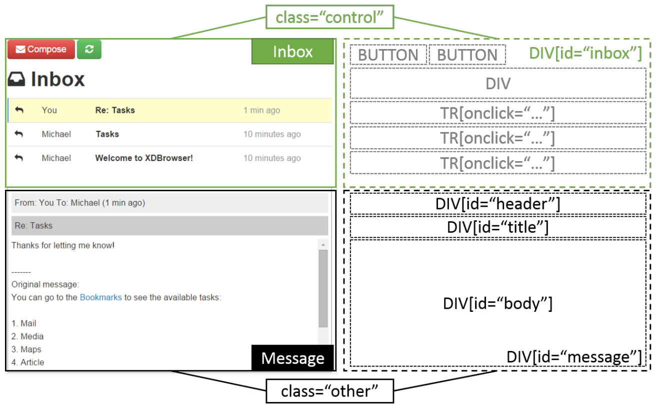
Our new goal then became to develop a mixed-initiative approach by developing semi-automatic support to split and distribute web pages between multiple devices. However, the first XDBrowser study [26] had different goals and provided limited insight so that additional studies were required to inform the design of a system following this new approach.

First, the earlier study focused on finding cross-device design patterns, not on finding a "good" interaction set for users to activate such patterns. Second, the patterns were derived in a study of only five sample web pages, and limited to settings with a phone and tablet. Thus, we were interested to see how the patterns might generalize to a larger set of popular web sites and settings with different and more than two devices.

INTUITION BEHIND SEMI-AUTOMATIC APPROACH

Consider the mail interface in Figure 1 where the inbox and message reading pane are distributed between phone and tablet. To distribute the elements in this way, the first version of XDBrowser required users to perform a series of manual selections and manipulations of page elements. The goal of our new techniques is to avoid the need for re-authoring.

Our main inspiration comes from the way many mobile browsers allow users to quickly zoom in and out to view different portions of the page in more detail. The idea is to allow users to invoke an interaction to zoom page elements on one


Compose

Inbox

You	Re: Tasks	1 min ago
Michael	Tasks	10 minutes ago
Michael	Welcome to XDBrowser!	10 minutes ago

From: You To: Michael (1 min ago)

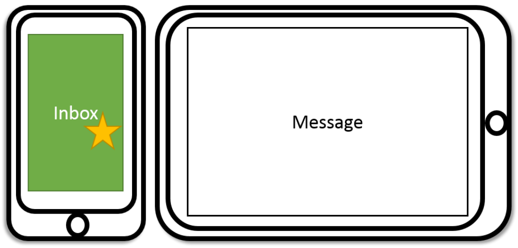
Re: Tasks

Thanks for letting me know!

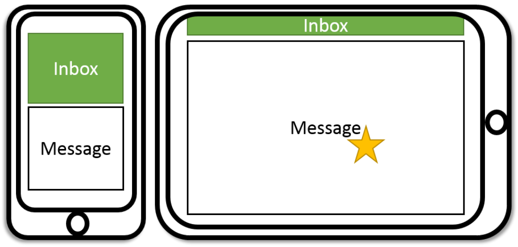
Original message:
You can go to the Bookmarks to see the available tasks:

1. Mail
2. Media
3. Maps
4. Article

Figure 5. Classification of HTML DOM nodes for mail interface



(a) Remote-control pattern



(b) Overview+detail pattern

Figure 6. Segmentation and triggered patterns for mail interface (star marks device and element invoked by a user-defined interaction)

device and, as the view is zoomed, activate a suitable cross-device pattern and make use of connected devices by showing relevant parts of the page that were pushed out of the view.

Figure 5 illustrates how the inbox and message elements are composed from nested HTML DOM nodes. Figure 6 illustrates the page segmentation relevant for two of the patterns we want to automate, *remote-control* and *overview+detail*.

Selecting a message in the inbox controls which message is shown in the reading pane. Hence, the *remote-control* pattern should become active when the user invokes the inbox element on one device. The inbox should be zoomed on that device and the message moved to the other device. For such a distribution, we would need to traverse the DOM tree from the node that first received the user input event upwards until we find the node wrapping the inbox element, i.e., the `DIV` with `id` "inbox" (Figure 5). This node is characterized by having an `id` attribute and containing a set of `BUTTON` and `LI` nodes with `onclick` event handlers, indicating that it is a "control" node. To obtain the pattern, we can zoom this node on the phone and remove it on the tablet so that the message gets zoomed there instead (Figure 6(a)).

On the other hand, if the message element which does not contain any "control" nodes is invoked, the *overview+detail* pattern should become active instead. The message reading pane should be zoomed on the current device and the overview should be kept on the other device. The message element is composed of several nested `DIV` nodes with `id` "header", "title", and "body". Since all of them have the same width as the "message" `DIV`, zooming any of them has the same effect as zooming the "message" `DIV` directly. To obtain the pattern, we can zoom either node on the current device and keep the overview on the other device (Figure 6(b)).

In the following, we present two studies to design such cross-device support, the first to identify desirable interactions for activating the different patterns, and the second to develop a segmentation method that can find corresponding nodes.

STUDY 1: USER-DEFINED INTERACTIONS

In [26], participants were asked to create desirable cross-device interfaces using XDBrowser's customization tools. That study focused on *what* kinds of cross-device interfaces users would want, and identified seven multi-device design patterns. The present study instead asks *how* users would want to obtain such interfaces. Specifically, we seek simple user-defined interactions *(1)* for selecting a cross-device pattern to transform a single-device design, and *(2)* for switching between different cross-device patterns. Interaction proposals must be coherent, implementation in XDBrowser feasible, and the number of previously required editing steps reduced.

Study Design

As motivated by Morris et al. [25], we used *partners*, *production*, and *priming* for elicitation. Our lab-based study was conducted in pairs and each pair were asked to produce at least three different interaction proposals per pattern to make sure that they considered a wide range of possibilities. Participants were primed with a mobile multi-device "on-the-go" scenario to set the context and had full control over devices, using them freely to demonstrate interactions, but an in-the-wild study may still lead to different proposals. Participants were also primed with a video showing the envisioned semi-automatic support for cross-device interfaces, however, without showing any user interactions that might trigger a pattern.

The typical protocol for elicitation studies shows the *effect* of an interaction to participants, e.g., using a video, and prompts them for the *cause* for a number of referents [39]. We adapted this study design as follows. Based on an example single-device design, we showed the effect in the form of a popular cross-device version of the design after applying a given pattern from [26], and then asked participants to demonstrate interactions that could trigger that effect and explain what should happen—independent of any technical considerations.

To create and switch between single-device and cross-device designs, the study facilitator loaded each interface into XDBrowser. In a Wizard of Oz-style similar to [24], it was also the facilitator who used XDBrowser's tools to re-author designs for a given pattern. This is in contrast to [26] where participants were given access to all features of XDBrowser and asked to customize the interfaces themselves.

A post-study questionnaire collected participants' ratings of the three studied settings using XDBrowser, comments on the benefits and limitations of using XDBrowser, and demographics. The study took slightly more than one hour per study pair, and participants were compensated $20 USD each.

Elicitation Tasks

We aimed to cover a broad range of common web browsing tasks: *(a)* information browsing tasks (with little to no text input), e.g., when playing videos or browsing maps, *(b)* information input tasks, e.g., when composing a new mail, and *(c)* information gathering tasks as combinations of browsing and input tasks, e.g., when responding to messages in a mail interface and checking availability or setting up appointments in a calendar interface. Our elicitation study was composed of 18 tasks: six referents $\times$ three conditions (Figure 7).

As referents, we chose the most popular designs for the *overview+detail*, *view+input*, *remote-control*, *split*, and *extend* cross-device patterns from [26]. We added a new referent (R3) prompting participants to switch patterns between the *overview+detail* and *view+input* patterns. The need to switch between different interface distributions depending on the task was frequently raised by participants but only possible using multiple tabs. The referent for the *split* pattern (R5) was included to probe how participants would want to control synchronization, a question of particular concern to the earlier study. The last referent (R6) is a version of the *extend* pattern that replicates a tab viewing the calendar on the second device. This is a cross-apps task frequently mentioned by participants but not part of the earlier study [26].

We considered mobile settings as ones in which users might especially benefit from combining multiple devices for more screen real estate and larger input surfaces. We prepared three settings involving different types of mobile devices—*phone+tablet* as in [26], *watch+phone*, and *watch+phone+tablet*. The two new settings compared to [26] allowed us to study how users might want to adapt and refine the patterns when using different and more than two devices.

Raedle et al. [30] presented an elicitation study that included move, copy, expand, and duplicate (both fully and partially) referents for phone and tablet settings. Their study therefore provided a good basis for comparison. However, we required additional referents, namely to switch between cross-device versions of the same interface (R3) and switch between different interfaces (R6), and control synchronization (R5).

Participants' Backgrounds and Experience

The study targeted end-users. We recruited 16 HCI bachelor and master students (8 male, 8 female, median age of 24 years) with rich diversity in backgrounds and experience (UX design, architecture, media, project management). Each passed a short selection interview making sure that they have experience with mobile touch devices (median 5 out of 5) and

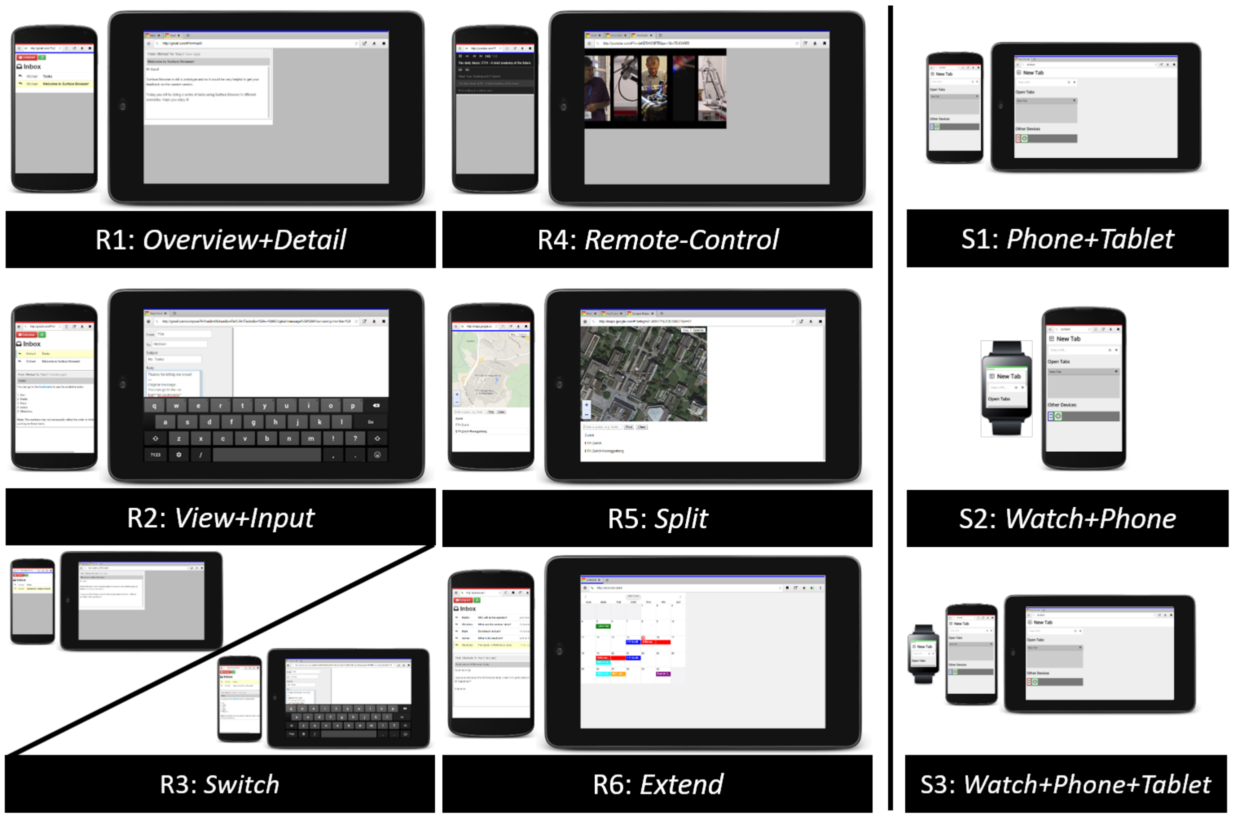


Figure 7. R1-6 show the cross-device interface referents and S1-3 the multi-device settings that formed the 18 elicitation tasks in the study.

interface design (median 3.5 out of 5). All expressed strong familiarity with the sample interfaces, using their mobile devices for mail, media, and maps-related tasks on a daily basis.

Proposed Cross-Device Interactions

Pattern	Most Popular Interactions
Overview+Detail (R1)	Drag detail element
View+Input (R2)	Swipe / double touch form input element
Switch (R3)	Swipe
Remote-Control (R4)	Drag / double touch control element
Split (R5)	Long press to trigger action only locally
Extend (R6)	Long press drag element or page to be copied

Table 1. Cross-device patterns and most popular interaction proposals

Overall, participants proposed 161 interactions after going through all six tasks for each of the three multi-device settings so that all three settings could be supported with their defined interaction sets. Table 1 shows a mapping of patterns to participants' most popular interaction proposals. See `https://github.com/mi2lab/xdbrowser2` for the complete range of interactions and details of our following analysis.

Participants' Design Goals

Participants aimed to define interaction sets that we characterize as simple, consistent, and conflict-free. By simple, we mean that interactions could be executed in a single step to keep required user input minimal. By consistent, we mean that users would not have to learn and memorize a large variety of gestures and that similar gestures had a similar *effect* (as in trigger a closely related cross-device pattern). By conflict-free, we mean that each new interaction had a different effect and that it did not conflict with existing interactions on the three types of devices used in the study.

Popular Types of Interactions

For the following analysis of participants' interaction proposals, we referred to Material Design Gesture Patterns[1] to analyze the touch mechanics (what fingers did on the screen) and touch activities (results of specific gestures), in addition to the research literature on device motion gestures [31] and multi-device gestures [7, 8, 15, 18, 30, 33, 35]. We computed a variant of the max-consensus and consensus-distinct ratio proposed by Morris [24] to identify the overall most popular interaction proposals across the three multi-device settings for a given referent (using a consensus threshold of two).

The predominant interactions for the first four referents were swipe (gross gesture, faster, typically has no on-screen target) and drag (fine gesture, slower, more controlled, typically has an on-screen target). For example, participants preferred swipe (with no particular on-screen target) for *view+input* (R2) and switching between *overview+detail* and *view+input* (R3), and drag (with detail and control element as target) for *overview+detail* (R1) and *remote-control* (R4).

This was not too surprising since all of these tasks involved moving page elements between devices and other studies found similar interaction preferences [30]. However, a second high-ranking proposal was to just touch or double touch

[1]https://material.google.com/patterns/gestures.html

the corresponding node. Similar to our own intuition, participants' common expectation with most proposals was that the browser would somehow determine the desired pattern, just from the type of element involved in the interaction, and then zoom the respective node and activate the pattern.

For the *split* (R5) referent, participants favored long press to trigger an action only on the current device without synchronization on other devices. For *extend* (R6), they frequently proposed long press drag to copy and sync the target element, or the entire page if no target was selected. With *extend*, participants also envisioned that a drag image would be generated from the selected element which they would drag in the direction of the target device and release at the edge of the screen. As an alternative spatially-agnostic interaction, an overlay with icons for all connected devices could be shown on the screen. As the element is dropped on a particular device's icon, only that element would be shown on that device.

Changing Interactions to Fit Device Combinations
Initially, participants proposed a total of 140 interactions for the *phone+tablet* setting. We recorded 21 changes to the interaction sets as the device settings changed—10 for *watch+phone* and 11 for *watch+phone+tablet*. We classified three types of changes. First, participants needed to discern the target device of an interaction in the three-device setting, e.g., when moving a page element to one of the other devices. Second, a new setting prompted them to switch the preferred interaction. Third, in two cases, participants proposed a new interaction. The reason for making such changes was conflicting swipe interactions to move elements between devices with bringing up another screen on the watch.

Leaning towards Spatially-Agnostic Interactions
Initially, we could also observe the tendency to prefer spatially-aware interactions similar to Raedle et al. [30]. However, as the watch was introduced in the second setting, and in particular when all three devices were used in the third, participants increasingly felt that using devices freely and positioning them for spatial interactions actually made it harder to select a target device, especially when target devices were aligned in the same line. In fact, they frequently tried several spatial interactions before they resorted to spatially-agnostic menus, buttons, and dropdowns as their preferred techniques.

Adapting Patterns to Different Devices
When switching to the *watch+phone* setting, participants argued that the patterns quite naturally translated to the smaller set of devices. Participants just shifted the role of the phone to the watch, and the role of the tablet to the phone. However, for 5 tasks in this setting, participants said that even though the proposed interactions would still hold, they would probably not use the watch and disable the cross-device interface, i.e., switch to the *single* pattern on the phone.

Expanding Patterns to More Than Two Devices
When re-introducing the tablet in the third setting, participants typically reassigned the tablet its original role. There was then the overall preference to duplicate the phone's interface on the watch. The exception was one pair of participants who argued that they would not extend the mail/ calendar interface (R6) to the watch since having it distributed between the phone and tablet was already sufficient.

Participants' Feedback
In the post-study questionnaire, participants rated enjoyment, ease of use, and effectiveness when operating the browser in all three settings (using a 7-point Likert scale). The first setting, *phone+tablet*, fared by far the best with averages around 5.5 for all three ratings. When switching to *watch+phone*, the ratings were consistently lower around 3.5. Finally, there was a consistent average rating of 3.9 for *watch+phone+tablet*.

This is an interesting finding since it is often considered a limitation to just have two devices in cross-device studies [26]. The common assumption among cross-device researchers seems to be that using more than two devices might be beneficial [15]. We observed the inverse effect, but the results might look again different with other types of devices.

STUDY 2: USER-DEFINED PAGE SEGMENTATIONS
Web page segmentation is the process of splitting page content into smaller blocks of elements. These elements can then be extracted as a group. Common techniques are based on analyses of HTML DOM nodes' content, structure, and hierarchy [6, 9, 16, 34]. We needed such a technique for our new browser to select and distribute page elements relevant to each pattern, but from the XDBrowser studies so far, we did not have the required information since participants selected elements visually in the rendered page, without looking at the underlying nodes [26]. Another limitation of the earlier study is that it was based on only five pages inspired from GMail, YouTube, Google Maps, Wikipedia, and Flickr. We wanted to see how the patterns could be applied to a larger corpus.

Study Design
Our second study was divided into two parts. The goal of the first part was to develop heuristics for automatic page segmentation. We produced a dataset of 41 web pages manually segmented and labeled for the same five cross-device patterns as in the previous study. The goal of the second part was to develop a segmentation method based on content analyses of the labeled pages. We then performed tests to check whether our new method is able to match the manually created labels.

We started with compiling a list of 50 Alexa Top 500 Global Sites. By selecting the top five sites from ten different Alexa categories (Arts, Business, Games, Health, Home, News, Science, Shopping, Society and Sports), we wanted to cover a broad range of web site genres. We then hired three students for the labeling tasks. The students first worked together to select and download one representative content page for each site—e.g., an article page on news sites such as CNN. Nine pages that required an account to access content (e.g., on banking sites) were skipped, giving us a sample of 41 pages.

In the next steps, each student individually labeled the elements corresponding to each pattern. To do this, they used the Chrome developer tools to inspect each page, visually select the element in the rendered page, and copied the CSS selector of the respective HTML DOM nodes into a spreadsheet.

In the last step, the students met and used a shared display to review their node selections on a pattern-by-pattern basis for each page. They determined which nodes were the same or similar visually, which selections were different CSS selectors but still targeted the same visual elements, and which were actually different, and marked those using color coding.

Each student took on average 10 hours from start to finish for the labeling tasks, and about 2.5 hours to compare and analyze the differences. The pay rate was $15 USD per hour.

Dataset

Applying the five cross-device patterns to all 41 pages in the sample, the students produced a joint total of 981 labels. See **`https://github.com/mi2lab/xdbrowser2`** for the complete dataset and the spreadsheet with labels and our analysis.

Agreement Scores

We computed initial agreement scores between the three students by building the proportion of similarly labeled nodes to all nodes they individually selected for each pattern.

- The highest agreement was achieved for the *view+input* pattern (85%). The students stated that the input controls were easy to visually identify and select in each page.
- Agreement scores were still relatively high for *extend* (73%) and *split* (68%). While the students typically identified the same elements, they selected different child nodes.
- Agreement scores were lower for *remote-control* (49%). The students found competing candidates for the control (e.g., top menus and sidebars) and had different preferences for the remote (whole page vs. just target elements).
- Finally, there was very little agreement for *overview+detail* (4%) due to a much higher number of candidates given that many pages contained multiple alternative detail nodes (e.g., video description in text vs. actual video).

Content Analyses

In the next steps, we performed content analyses to identify the reasons for varying agreement between the students. We found that the existence of multiple page elements with similar functions and hence scoping was the main reason for the mixed agreement scores. Even though students selected different nodes when considering the whole page, they in fact identified different "local" elements with similar "global" functions, meaning that the criteria they applied matched.

- The closest matching DOM selections were indeed found for input elements, which either wrapped or were input elements themselves.
- The control nodes were commonly characterized by wrapping links, buttons, or drop-down lists.
- The remaining selections only had in common that they were container nodes for multiple block-level elements.
- Approximately 89% of selected nodes had an `id` attribute.

Preliminary Page Segmentation Method

Based on these insights, we developed a preliminary segmentation method. When the user invokes an element in the page, we get the DOM path to the node that caught the event. Starting from this node and, by traversing the DOM hierarchy upwards, subsequently visiting all parent nodes until we reach `BODY`, we find the closest node that can be classified as:

- *control*, if the node has an `id` and wraps `A` nodes, or nodes with `onclick` handler, `BUTTON` or `INPUT[type="button"]`, or `SELECT` nodes;
- *input*, if the node has an `id` and wraps `INPUT` or `TEXTAREA` nodes, or is one itself;
- *other*, if the node is neither *control* nor *input*, has an `id`, and contains `DIV`, `P`, `UL`, or `TABLE` nodes.

We acknowledge that this is a first simple segmentation method that was derived from our analyses of 41 pages. While we believe it to be generic, it is hard to find a "good" training set that represents most existing web pages. Nevertheless, our initial tests showed that these three classes and respective node types can achieve good results on many pages.

XDBROWSER 2.0 PROTOTYPE

Based on the results of our two studies, our new prototype (Figure 8) adapted XDBrowser [26] as described below.

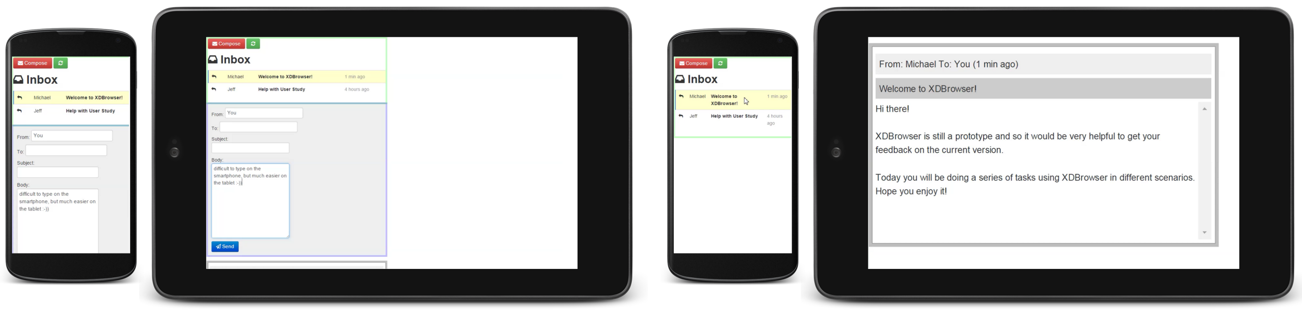
Semi-Automatic Segmentation and Pattern Activation

First, we need to emphasize that our technique is robust in that it does not break web pages–we only zoom target elements on one device and hide them on the other. For edge cases that activate a different pattern than the one desired by the user, the cost is not higher than zooming the wrong element in common mobile browsers. The user can easily restore the unzoomed view. As in current browsers, they could first use the browser's default zoom for more precise node selections, and then our method can be used to trigger the pattern.

Our new prototype uses the *duplicate* pattern as the default (Figure 8(a)). When connecting a new device, it copies open pages and synchronizes their view state and input. From this pattern, all other patterns are triggered as described below.

When swipe, drag, or double touch is invoked on an element, we perform the classification above and proceed as follows:

- If a node of class *control* is found, the *remote-control* pattern will become active. The *control* node will be zoomed on the current device. On connected devices, the *control* will be hidden and remaining nodes zoomed (Figure 8(b)).
- If a node of class *input* is found, *view+input* will become active with only the *input* node visible and zoomed on the current device, while being hidden on connected devices.
- If a node of class *other* is found, *overview+detail* will become active and the node zoomed on the current device.
- If no such node is found before `BODY`, the *duplicate* pattern will become active and the unzoomed view restored.



difficult to type on the smartphone, but much easier on the tablet :-))

From: Michael To: You (1 min ago)

Welcome to XDBrowser!

Hi there!

XDBrowser is still a prototype and so it would be very helpful to get your feedback on the current version.

Today you will be doing a series of tasks using XDBrowser in different scenarios. Hope you enjoy it!

(a) Duplicate as the default (b) Remote-control pattern after invoking the inbox element

Figure 8. XDBrowser 2.0 prototype with default *duplicate* pattern and after activating the *remote-control* pattern on the mail interface from Figure 1

When long press is invoked on an element, we activate *split* and perform the action without synchronization with other devices. When long press is followed by drag, we find the closest *control*, *input*, or *other* node. If there is such a node before we reach BODY, the *extend* pattern will become active and, on the target device, all nodes will be hidden except for that node. If no such node can be found, instead the *duplicate* pattern becomes active and all nodes will be shown. When overriding the active pattern with *extend* or *duplicate*, sync will be enabled on that device. This allows users to reintegrate a second device without having to switch devices.

Architecture and Implementation

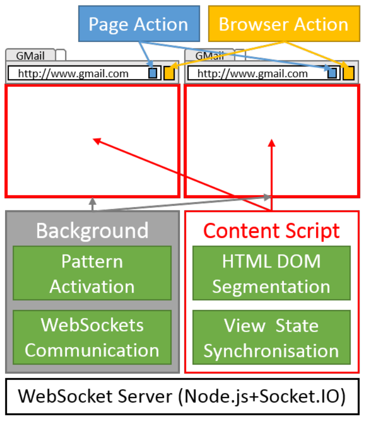


Figure 9. Main components of our prototype's Node.js-based client-server architecture with WebSockets multi-device communication

Our new prototype is a hybrid of a standalone web application and a Chrome extension, making it possible to run on both desktop and Android mobile devices, including tablets, phones and watches, where Chrome extensions are currently not supported. On tablets and smartphones, any browser can function as the host browser for our prototype application. On smartwatches, the Wear Internet Browser app is required.

The architecture is shown in Figure 9. The prototype connects and jointly operates multiple browser windows split over connected devices. The implementation is divided into client-side background and content scripts and a Node.js[2] server using Socket.IO[3] for WebSockets communication.

[2]http://nodejs.org/
[3]http://socket.io

The background script is executed once per browser window and runs in the background. It is used to activate patterns and maintain a WebSocket connection between multiple browser windows through the server. It is implemented in JavaScript using Chrome's APIs to define the page and browser actions and listen to messages from content scripts.

The content script is executed for every page loaded into a tab. It is used to inject our DOM segmentation and view state synchronization methods. It is implemented using jQuery for DOM manipulation, hammer.js[4] for handling touch events, and zoom.js[5] for zooming page elements.

Technical Evaluation

We conducted a technical evaluation of our new prototype in two parts: *(1)* testing the page segmentation method and *(2)* testing the prototype on multiple devices and popular web sites. An end-to-end user study is planned for the future.

Testing the Semi-Automatic Approach

We tested the new prototype on pages in our sample, attempting to match nodes selections with semi-automatic segmentation that students had manually labeled and agreed on for the five cross-device patterns. For scoping, we started the method by invoking the rendered element that students had visually selected, which does not guarantee a match. There are many variables such as how and where the user invoked the element, DOM hierarchy, and CSS positioning. Conversely, even non-exact matches can have similar visual results because of how nodes are often nested to compose elements.

In our tests, we were able to classify and match input elements with high accuracy, and trigger the *view+input* pattern as intended. We achieved similarly high accuracy for *remote-control*, but, because of the way the remaining elements were constructed on some pages, it did not always achieve the intended zoomed view on the remote device. For *split*, *extend*, and *overview+detail*, we were able to match the exact nodes about 50% of the time, as the method often selected a child node "closer" to the node that was actually invoked. However, on the majority of pages, we were still able to achieve the intended zoomed view because of similar visual attributes.

[4]http://hammerjs.github.io
[5]http://github.com/hakimel/zoom.js

Testing on Multiple Devices and Popular Sites

The second part of our evaluation is a report on the author's experience using the prototype to browse popular web sites in a number of multi-device settings. Although we successfully deployed the prototype on many Android tablets, phones, and smartwatches, we experienced a couple of limitations of the current implementation. The following should give an indication to readers, while more systematic tests are planned.

First, we noticed performance issues with zooming that sometimes impacted UI responsiveness. Generally, double touch to zoom worked best in the larger device settings and when used to offload content from, or redirect input to, a smaller device.

The *view+input* pattern is a good example of this. The larger device can be transformed into a keyboard for entering data on the smaller device. This was very useful for longer text entry such as when writing mails in the *phone+tablet* setting. However, it was also very helpful for searching on many sites. In the *watch+phone* setting, this was the most feasible way of entering search terms on the watch. Using the phone rather than the watch for input also seemed like a good option when speech input is not suitable in the current situation. It was even useful in a larger device setting, where using one device for input only helped to quickly refine searches while viewing results on the other device.

The *overview+detail* pattern proved problematic in the *watch+phone* setting when the watch was used as the detail view, in particular, for graphic-heavy content such as images and videos. Here, our own double touch to zoom technique showed a much poorer performance than default browser behavior using pinch to zoom. The problem had mostly to do with execution performance of JavaScript and CSS transformations which were very demanding on the watch. However, even standard zoom was often "lagging" when panning and rescaling the view. On the other hand, the pattern can be used for its own benefit by zooming into different areas of the page on the watch and viewing details on the other devices.

Finally, the *remote-control* pattern was an exception to our observations above. Here, the phone lent itself well to controlling the tablet and the watch for controlling the phone. In the *phone+tablet* setting, this seemed most useful for reading mails as in previous examples used throughout the paper and for remote playback of videos from YouTube and for controlling Flickr slideshows. In the *watch+phone* setting on Flickr, either device could be used to view image slideshows as they were automatically scaled to the browser viewport, but controlling them from the watch would avoid screen occlusions on the phone. While *remote-control* was generally also a possibility for Google Maps, here it seemed more natural not to redirect input to another device, but to control it directly to pan and zoom the map. There was the minor issue that the map zoom level changed when double touch to zoom was invoked on the map to activate the *remote-control* pattern.

RELATED WORK AND DISCUSSION

Below we discuss related work on *adaptive web browsing systems* and *cross-device interfaces and interactions*. We also discuss significance and generalizability of our new studies.

Adaptive Web Browsing Systems

A first line of research investigated novel web browsing systems providing interactive tools to address emerging device characteristics. For example, PowerBrowser [6] summarized page text and produced section outlines, while reducing white space and image quality. WEST [5] supported a combination of text reduction and focus+context visualization to provide an overview of pages that are too large to be viewed in full. WebThumb [38] allowed users to perform various operations on selected page elements, for example, picking up, zooming, and panning. All of them share the goal of making single-device interfaces on small-screen devices more accessible for web browsing. While Johanson et al. [19] created a first multi-device web browser, it was only with XDBrowser [26] that users were enabled to customize existing web pages for multi-device use using visual web page authoring tools.

Similar to XDBrowser [26], Highlight [28] and PageTailor [4] enabled end-users to customize pages for mobile devices. As an alternative, Collapse-to-Zoom [2], ThumbSpace and Shift [21, 37] developed new interaction methods for users to interactively zoom page content on small screens. Finally, opaccess [3] iteratively magnifies web pages as long as it does not lead to horizontal scrolling and overlapping text. Our new prototype adds to this line of research, and introduces automation support for distributing web pages on demand.

Rather than providing user-driven tools, a second line of research developed more automatic page transformation techniques. Again with the goal of fitting content into smaller screens, existing techniques rely on page summarization [5, 6], page splitting [9, 16], or fish-eye and thumbnail views [1, 23]. Advanced layout generation approaches required to cater to a much wider range of display contexts were investigated by Gajos et al. [13] and Schrier et al. [34]. As well as being limited to single-device use, both were originally designed for desktop user interfaces and are not easily ported to the web. Our segmentation method based on cross-device design patterns makes it possible to automate page distribution between multiple devices, and forms a contribution in this space.

Cross-Device Interfaces and Interactions

Our work also adds to a growing body of knowledge on providing better support for cross-device interactions. Here, we can identify at least two relevant streams of research.

The first stream of research has investigated new cross-device design tools [12, 14, 15, 18, 27, 40]. The vast majority of these tools, however, are targeted at developers, allowing them to distribute interfaces between devices programmatically or using special authoring tools. While a number of possible cross-device interfaces were showcased to illustrate technical support, whether or not these address actual user needs and are desirable by end-users was only recently explored with XDBrowser [26]. Our new prototype adds to the growing list of tools for cross-device design. The semi-automatic approach we have developed based on a first set of cross-device patterns is a promising new direction that could potentially help transform the large variety of existing single-device interfaces without the need for major modifications.

The second stream is comprised of end-user elicitation. To probe possible designs without technological constraints [39], elicitation studies have proven to be useful to generate user-defined interaction sets, e.g., for multi-display environments [35], for connecting phones and large displays [33], and for interacting with multiple phones and tablets [30]. In particular, the study by Raedle et al. [30] on spatially-aware vs. spatially-agnostic interactions is closely related. While we could observe similar trends, it was interesting to see how participants changed their interaction proposals from spatially-aware to increasingly agnostic techniques, as we changed both the types and the number of devices during elicitation.

Significance and Generalizability

In sum, cross-device research has independently produced elicitation studies and systems. Raedle et al. [30], Nebeling and Dey [26], and our paper generate research value by incorporating both aspects. The two studies we present fill the gap between user-driven cross-device elicitation as in [30] and developer-driven automatic solutions such as Panelrama [40].

There were two significant aspects to our studies that go beyond what was found by Hamilton and Wigdor [15], Raedle et al. [30], and Nebeling and Dey [26].

First, our design involved three devices and probed how interactions need to be adapted for different device combinations to make them work across all three device settings.

Second, our work helped identify the need for, and issues associated with, semi-automatic generation of cross-device interfaces. We developed first support based on the patterns from [26], paired with interaction proposals from Study 1. From Study 2, we developed an understanding of which patterns can be better automated than others, and why. Our analysis at a per-pattern level helps direct future research.

CONCLUSION

Existing cross-device design solutions impose significant effort on users and/or developers and require interfaces to be manually redesigned—either through visual authoring or in code [10, 12, 14, 17, 18, 27, 40]. Building on user-defined cross-device patterns from [26], this paper contributes two studies that inform the design of more automatic support for obtaining cross-device designs of existing interfaces. Specifically, we introduced a simple set of user-defined interactions to trigger cross-device designs and a pattern-based page segmentation method for semi-automatically distributing page content between multiple devices. Our new techniques strive to minimize the need for user *and* developer intervention. We discussed insights from cross-device studies that incorporate settings with more than two devices, including smartwatches and how they fit in. Implicitly, the paper also showed how to use our results, here to derive interactions and semi-automatic distribution to build a new version of XDBrowser [26].

Acknowledgments

Thanks to the Swiss National Science Foundation for partially supporting this research under mobility grants, P300P2_154571 and P300P2_164646. Special thanks to Andie Dumas, Annabel Weiner, and Licheng Zhu for assisting with this research, anonymous students for participating in the studies, and Anind Dey for comments on earlier drafts.

REFERENCES

1. Patrick Baudisch, Bongshin Lee, and Libby Hanna. 2004a. Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *Proc. AVI*.
2. Patrick Baudisch, Xing Xie, Chong Wang, and Wei-Ying Ma. 2004b. Collapse-to-Zoom: Viewing Web Pages on Small Screen Devices by Interactively Removing Irrelevant Content. In *Proc. UIST*.
3. Jeffrey P. Bigham. 2014. Making the Web Easier to See with Opportunistic Accessibility Improvement. In *Proc. UIST*.
4. Nilton Bila, Troy Ronda, Iqbal Mohomed, Khai N. Truong, and Eyal de Lara. 2007. PageTailor: Reusable End-User Customization for the Mobile Web. In *Proc. MobiSys*.
5. Staffan Björk, Lars Erik Holmquist, Johan Redström, Ivan Bretan, Rolf Danielsson, Jussi Karlgren, and Kristofer Franzén. 1999. WEST: A Web Browser for Small Terminals. In *Proc. UIST*.
6. Orkut Buyukkokten, Hector Garcia-Molina, Andreas Paepcke, and Terry Winograd. 2000. Power Browser: Efficient Web Browsing for PDAs. In *Proc. CHI*.
7. Nicholas Chen, François Guimbretière, Morgan Dixon, Cassandra Lewis, and Maneesh Agrawala. 2008. Navigation Techniques for Dual-Display E-Book Readers. In *Proc. CHI*.
8. Xiang 'Anthony' Chen, Tovi Grossman, Daniel J. Wigdor, and George W. Fitzmaurice. 2014. Duet: Exploring Joint Interactions on a Smart Phone and a Smart Watch. In *Proc. CHI*.
9. Y. Chen, W.Y. Ma, and H.J. Zhang. 2003. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. In *Proc. WWW*.
10. Pei-Yu (Peggy) Chi and Yang Li. 2015. Weave: Scripting Cross-Device Wearable Interaction. In *Proc. CHI*.
11. David Dearman and Jeffrey S. Pierce. 2008. "Its on my other Computer!": Computing with Multiple Devices. In *Proc. CHI*.
12. Luca Frosini and Fabio Paternò. 2014. User Interface Distribution in Multi-Device and Multi-User Environments with Dynamically Migrating Engines. In *Proc. EICS*.
13. Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. 2007. Automatically Generating User Interfaces Adapted to Users' Motor And Vision Capabilities. In *Proc. UIST*.
14. Giuseppe Ghiani, Fabio Paternò, and Carmen Santoro. 2012. Push and Pull of Web User Interfaces in Multi-Device Environments. In *Proc. AVI*.

15. Peter Hamilton and Daniel J. Wigdor. 2014. Conductor: Enabling and Understanding Cross-Device Interaction. In *Proc. CHI*.

16. G. Hattori, K. Hoashi, K. Matsumoto, and F. Sugaya. 2007. Robust Web Page Segmentation for Mobile Terminal Using Content-Distances and Page Layout Information. In *Proc. WWW*.

17. Tommi Heikkinen, Jorge Goncalves, Vassilis Kostakos, Ivan Elhart, and Timo Ojala. 2014. Tandem Browsing Toolkit: Distributed Multi-Display Interfaces with Web Technologies. In *Proc. PerDis*.

18. Steven Houben and Nicolai Marquardt. 2015. WatchConnect: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications. In *Proc. CHI*.

19. Brad Johanson, Shankar Ponnekanti, Caesar Sengupta, and Armando Fox. 2001. Multibrowsing: Moving Web Content across Multiple Displays. In *Proc. Ubicomp*.

20. Shaun K. Kane, Amy K. Karlson, Brian Meyers, Paul Johns, Andy Jacobs, and Greg Smith. 2009. Exploring Cross-Device Web Use on PCs and Mobile Devices. In *Proc. INTERACT*.

21. Amy K. Karlson and Benjamin B. Bederson. 2008. One-handed touchscreen input for legacy applications. In *Proc. CHI*.

22. Amy K. Karlson, Shamsi T. Iqbal, Brian Meyers, Gonzalo Ramos, Kathy Lee, and John C. Tang. 2010. Mobile Taskflow in Context: A Screenshot Study of Smartphone Usage. In *Proc. CHI*.

23. Heidi Lam and Patrick Baudisch. 2005. Summary Thumbnails: Readable Overviews for Small Screen Web Browsers. In *Proc. CHI*.

24. Meredith Ringel Morris. 2012. Web on the Wall: Insights from a Multimodal Interaction Elicitation Study. In *Proc. ITS*.

25. Meredith Ringel Morris, Andreea Danielescu, Steven M. Drucker, Danyel Fisher, Bongshin Lee, m. c. schraefel, and Jacob O. Wobbrock. 2014. Reducing Legacy Bias in Gesture Elicitation Studies. *Interactions* 21, 3 (2014).

26. Michael Nebeling and Anind K. Dey. 2016. XDBrowser: User-Defined Cross-Device Web Page Designs. In *Proc. CHI*.

27. Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira C. Norrie. 2014. Interactive Development of Cross-Device User Interfaces. In *Proc. CHI*.

28. Jeffrey Nichols, Zhigang Hua, and John Barton. 2008. Highlight: A System for Creating and Deploying Mobile Web Applications. In *Proc. UIST*.

29. Antti Oulasvirta and Lauri Sumari. 2007. Mobile Kits and Laptop Trays: Managing Multiple Devices in Mobile Information Work. In *Proc. CHI*.

30. Roman Rädle, Hans-Christian Jetter, Mario Schreiner, Zhihao Lu, Harald Reiterer, and Yvonne Rogers. 2015. Spatially-aware or Spatially-agnostic?: Elicitation and Evaluation of User-Defined Cross-Device Interactions. In *Proc. CHI*.

31. Jaime Ruiz, Yang Li, and Edward Lank. 2011. User-Defined Motion Gestures for Mobile Interaction. In *Proc. CHI*.

32. Stephanie Santosa and Daniel Wigdor. 2013. A Field Study of Multi-Device Workflows in Distributed Workspaces. In *Proc. UbiComp*.

33. Dominik Schmidt, Julian Seifert, Enrico Rukzio, and Hans Gellersen. 2012. A Cross-Device Interaction Style for Mobiles and Surfaces. In *Proc. DIS*.

34. Evan Schrier, Mira Dontcheva, Charles E. Jacobs, Geraldine Wade, and David Salesin. 2008. Adaptive Layout for Dynamically Aggregated Documents. In *Proc. IUI*.

35. Teddy Seyed, Chris Burns, Mario Costa Sousa, Frank Maurer, and Anthony Tang. 2012. Eliciting Usable Gestures for Multi-Display Environments. In *Proc. ITS*.

36. Desney S. Tan, Brian Meyers, and Mary Czerwinski. 2004. WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. In *Proc. CHI EA*.

37. Daniel Vogel and Patrick Baudisch. 2007. Shift: A Technique for Operating Pen-Based Interfaces Using Touch. In *Proc. CHI*.

38. Jacob O. Wobbrock, Jodi Forlizzi, Scott E. Hudson, and Brad A. Myers. 2002. WebThumb: Interaction Techniques for Small-Screen Browsers. In *Proc. UIST*.

39. Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. 2009. User-Defined Gestures for Surface Computing. In *Proc. CHI*.

40. Jishuo Yang and Daniel Wigdor. 2014. Panelrama: Enabling Easy Specification of Cross-Device Web Applications. In *Proc. CHI*.