Enabling Mobile Application Mashups With Merlion

Iqbal Mohomed
Microsoft Research Silicon Valley
iqbal@microsoft.com

ABSTRACT

We present Merlion, a system that enables end-users to build custom mobile applications by creating mashups from existing desktop applications. The original application executes on a machine running remote desktop software (such as VNC server) without any modifications. Users can utilize the Merlion Designer to select relevant visual regions of the original application and create an alternate layout that is more suitable to their circumstances (*e.g.* taking the screen real-estate of their mobile device into account). Once the custom application has been designed, the user can utilize the Merlion Runtime (running on the user's mobile device) to interact with their custom application. Merlion can improve user productivity by simplifying user interfaces, automate repetitive actions, make applications available across different mobile form factors, and can allow applications that work on different OS platforms to operate in concert. In this paper, we describe the design of the Merlion system, details of our initial prototype, and discussion of the benefits and challenges of our approach.

1. INTRODUCTION

The ecosystem of mobile devices is vastly different today than it was even a few years ago. For a variety of reasons - technical, business and social - smartphones have been enthusiastically embraced by a significant number of consumers. Online stores for mobile applications are thriving, with thousands of applications available for each of the major mobile phone platforms. Though they have come very far indeed, mobile phones are still not a replacement for desktop computers. Some of the reasons for this include cumbersome user interfaces, limited screen real-estate, openness of platform and differences in application availability across platforms. At present, each of the major smartphone platforms have significantly different SDKs for application development; on the iPhone, developers must use Cocoa and Objective C, for Windows Mobile there is the .Net Compact Framework, etc. In this paper, we propose a new way of creating mobile applications - visual mashups from exisitng desktop applications - simple enough for end-users to create themselves. While such a notion may have sounded absurd only a few years ago, we believe that this approach is applicable today in limited scenarios, and that its applicability will grow significantly in the next five to ten years. In our approach, graphical data is retrieved from a server's framebuffer (*i.e.* the server transmits pixel data), and we use this as the basis for building mashup applications. Additionally, user input (such as text input or mouse clicks) are relayed to the server. This functionality can be thought of as a lowest-common-demoninator feature across all graphical platforms, and allows greater flexibility in creating mashups that span platforms and applications.

To illustrate the idea, we have implemented a prototype system called Merlion[1]. In our design, unmodified applications run on a server that is always available on the network. Additionally, the server runs remote desktop software (such as VNC server or Remote Desktop Services) to allow client devices to connect, and make use of application functionality. An end-user can utilize the *Merlion Designer* to interactively select desired visual regions of the original application's graphical user interface (GUI) and alter the layout of these regions to better suit the constraints of his or her mobile device. Once the user has designed their mashup, the *Merlion Runtime* is used to run the mashup application. Both the Merlion designer and runtime make use of the remote frame buffer protocol (or some variant) to access the server running unaltered applications.

Merlion can improve user productivity by automating repetitive user interactions, such as logging into to an application or performing some action that normally requires multiple interactions - cumbersome tasks given the limited user interfaces of mobile devices. Merlion can also be used to make applications available on platforms for which they were not specifcally developed. This alleviates challenges that emerge from closed application platforms. Finally, users can utilize Merlion to customize an application's user interface to their specific tastes and tasks, and to the constraints of their mobile devices.

The rest of the paper is organized as follows. Section 2 describes the high-level design of the system. Sections 3 and 4 provide details on the current implementation of the

[1]A Merlion is a mythical creature with the head of a lion on top of the body of a fish. It is a mashup, just like the applications enabled by our system.

HotMobile 2010, February 22-23, 2010, Annapolis, MD, USA.

Merlion designer and runtime, respectively. In section 5, we describe new scenarios enabled by Merlion. Section 6 discusses challenges of our approach, and section 7 describes related works. We present conclusions in section 8.

2. SYSTEM OVERVIEW

In this section, we provide a high-level overview of the Merlion system. Figure 1 illustrates our design. The original applications that are the basis for the user's mashups are run on a machine with strong network connectivity (right-side of the figure). We call this machine the *server*, though it could be a desktop computer running inside a user's home or office, or a virtual machine instance that is hosted by a cloud provider. While the application does not have to be modified in any way, the server must run remote desktop software (such as VNC or Remote Desktop Services), and interacts with the Merlion designer and the Merlion runtime (left-side of the figure).

The Merlion designer is utilized by end-users to visually create their custom mobile application. It can run on a desktop computer and displays the user interface of the original application running on the server (using a remote desktop protocol such as the remote framebuffer protocol (*RFB*) or the remote desktop protocol (*RDP*)). A user can select regions that they deem relevant to the mashup application they wish to create, and use a WYSIWYG layout tool to design how these visual regions will appear on their mobile device (*e.g.* conforming to a specific screen size). The output of the Merlion designer is a *rules* file, which encodes the *(x,y)* coordinates, width and height for each visual region on both the server-side and on the client at run-time.

Once the user has designed the mashup application to his/her satisfaction, the rules file is uploaded to the mobile client. At this point, the user can utilize the Merlion runtime on their mobile device to render the mashup application and forward any input on the client device to the remote server (again, using some remote desktop protocol).

For simplicity, Figure 1 does not illustrate helper software that runs on the server called the *Merlion Coordinator*. The coordinator is responsible for invoking applications, positioning and sizing them on the desktop surface, and terminating applications. In our current implementation, applications are laid out one after the other on the server's display. Moreover, the coordinator utilizes the size and position provided by the Merlion runtime. Another limitation is that we cannot have more applications on a single server than the number that can be laid out on the display. These limitations are not fundamental and can be overcome by additional engineering.

Also in Figure 1, we show the Merlion designer and runtime interacting with a single server running an application. However, this is not a limitation to Merlion. In fact, it is possible to create a mashup from multiple applications (running on the same or different machine) and across platforms (such as different operating systems). The only requirement is that the machine running the application must have remote desktop software.

In the next two sections, we provide details on our implementation of the Merlion designer and runtime.

3. MERLION DESIGNER

The Merlion designer enables users to visually create mashup applications. Figure 2 shows a screenshot of the designer in action. It consists of a toolbar with various buttons at the top, and a client area that displays the screen contents of the server. In this case, the calculator application on Windows is running on the server, and the user has selected four visual regions (depicted by rectangles). Once the visual regions are selected, the user can press the "Layout" button to modify how the visual regions show up on the display of the mobile device. The user is then shown a window with the same surface area as the screen size of their mobile device, and they can arrange the previously selected visual regions however they want (depicted in Figure 3). In addition to allowing modification of the *x,y* coordinates of visual regions, our prototype also supports scaling their height and width.

The example described above and illustrated in the screenshots is relatively simple because there is a single screen of the application being acted upon. To deal with more complex applications, we need some additional machinery (conceptual and software). We define the term *stencil* to correspond to a specific screen state of the application from which the user selects a set of visual regions. We also define four *actions* that can take place at various points in time: (i) sleep for s seconds, (ii) transition to another stencil, (iii) enter text at some *x,y* coordinate, and (iv) peform a mouse click at some *x,y* coordinate. Users can create a sequence of actions and associate them with a particular visual region on a stencil, and that sequence of actions is initiated when explicitly directed by the user. Sequences of actions can also be associated with the overall stencil, but in this case, the sequence is initiated as soon as we transition into the stencil. While we do not claim that these semantics are expressive enough to create any mashup from arbitrary applications, we believe that they are sufficiently powerful for us to build the mashups we desire without being excessively complex in terms of the design model.

Once the user has designed their mashup application, they select the "Generate" button to create a text file (we call this the *rules* file) describing their mashup. This file is moved to the user's mobile device and used by the Merlion runtime, which we describe in the next section.

4. MERLION RUNTIME

The Merlion runtime makes use of the rules file to provide the user with the mashup application. It connects to the server running the unaltered application and a VNC server, and only requests parts of the display relevant to the user's mashup (via the RFB protocol). Figure 4 shows a screenshot of the runtime executing on a Windows Mobile 6 Professional device and continues the example from Figure 3. The Merlion runtime allows the user to press buttons on the touchscreen and relays these as mouse-click events to the server running the original application. Additionally, the runtime can also transmit text input.

Apart from relaying user input events, Merlion must continuously refresh the screen. In our current implementation, we poll the server 25 times a second for smooth transitions

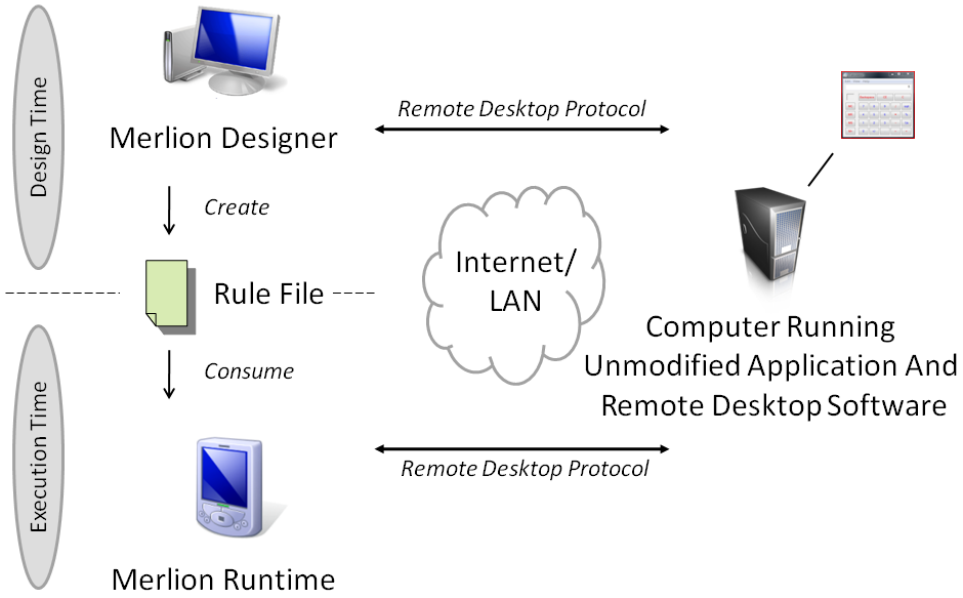

Figure 1: Merlion System Design

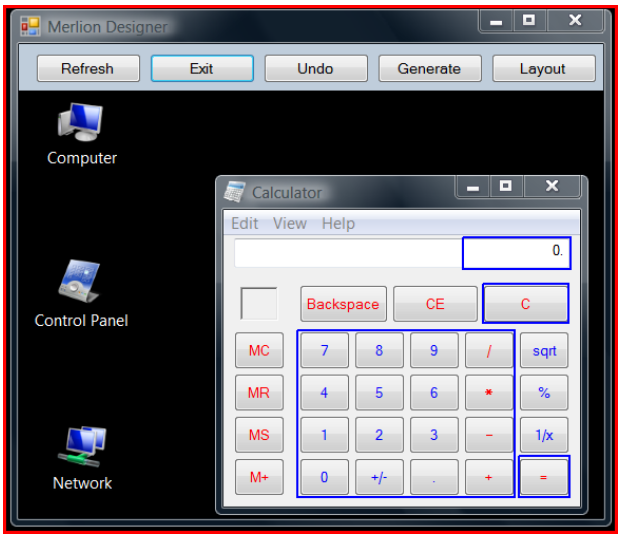

Figure 2: A screenshot of the Merlion Designer

on the user interface. The RFB protocol allows the server to state that parts of the screen have not changed, and when they have changed, to provide compressed representations. This affects the amount of data that must be transmitted from the server, and is revisited in the discussion section.

5. SCENARIOS

In this section, we describe some scenarios where Merlion can be used.

Customizing existing desktop and web applications: Customizing desktop and web applications for mobile devices has been explored in several recent research projects. The key idea is to either limit the functionality of the application and allow the user to quickly access the features that they care about the most. Or alternatively, to allow users to customize the organization of information or the features of an application to better suit their device. Merlion can be used to achieve the same functionality as other works[10, 13] with some key additional benefits. First, Merlion works for desktop applications as well, while prior work focuses on customizing web pages (by changing the DOM underlying the web page). Second, for web applications, Merlion does not break Javascript functionality. This is a weakness suffered by prior approaches as DOM changes can potentially break scripts that run on web pages.

Running applications on novel devices: Merlion can allow mashup applications to be created on platforms that were not originally intended to run an application. For example, it is possible to build a mashup application that allows a user to review their slides on a handheld gaming device such as the Nintendo DS (which supports WiFi and has a vibrant homebrew community[5]) by implementing a Merlion runtime for that device and having the server run the slideshow application.

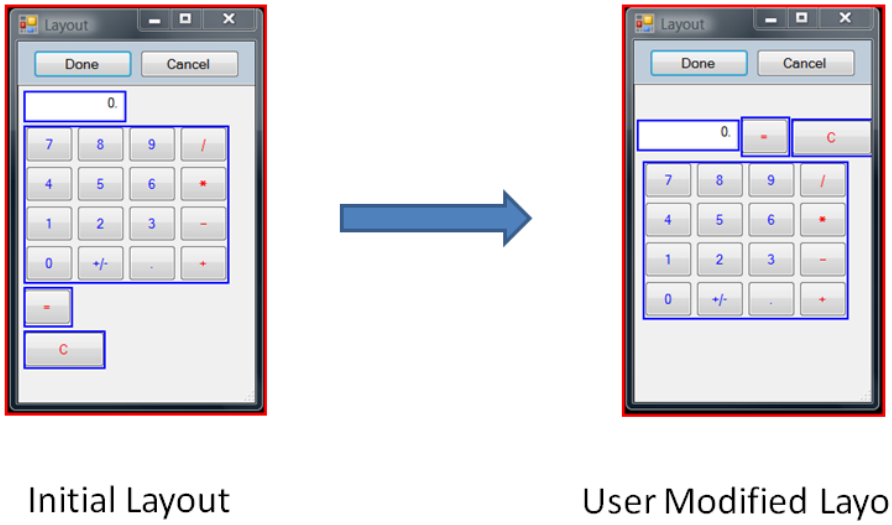

Figure 3: Screenshot showing how an initial layout of visual regions (arranged in a single column) and subsequent to user modification

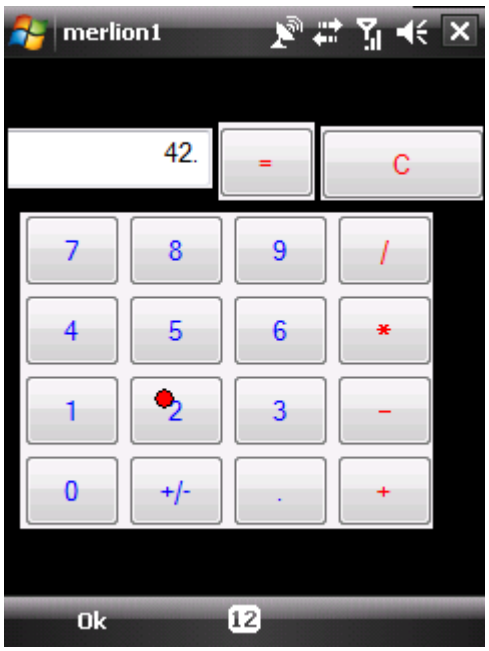

Figure 4: A screenshot from the Merlion Runtime

Controlling actions on a desktop via a feature phone: We are currently developing an application that integrates with an Interactive Voice Response (IVR) service provider as well as Merlion in order to allow a feature phone to control a desktop computer. Rather than allowing access to all desktop functionality, we are using the Merlion designer to enable a user to define a sequence of actions that are performed on software running on the desktop, and then integrate these action sequences with IVR functionality (such as presses on a phone's keypad). This would allow a feature phone to be used as a desktop replacement.

Automating repetitive tasks: Another use case for Merlion is to automate repetitive tasks on a desktop computer when accessing it via a mobile device. Some smartphones today provide VNC clients that can be used to access the entire screen surface of a desktop. Due to the limited user interface capabilities of mobile devices (screen real-estate, text entry, lack of mouse input, *etc.*), routine actions are cumbersome to perform. Our initial motivation in creating Merlion was to simplify the user input required in order to log-in to applications, check whether we have new messages on Facebook (prior to the custom Facebook application appearing on our device) and making toll-free long-distance phone calls (via a website that makes a local call to both caller and callee). As an alternative to the Merlion runtime, we created a desktop application for the Windows Vista sidebar that would simply display an image of the status of an application running on a server. We used this to instantly see if we have new messages (on Facebook and MS Outlook), without running the applications on our desktop.

6. DISCUSSION

In this section, we discuss key challenges and issues that must be overcome by the research community in order to create mashup applications that are based on accessing a remote framebuffer.

Network: To work well, Merlion requires a low latency, high bandwidth and unmetered network connection. At present, these requirements are met by home wireless access points that are not under heavy load. As such, Merlion can be used at home today to access desktop functionality "from the couch". This role is currently served by laptop computers but we believe that Merlion's ability to automate repetitive actions can result in an improved user experience. On the otherhand, when we consider current 3G networks, we see a challenging environment for Merlion. Latencies on these networks are higher (average RTT is 395ms on 3G versus 90ms over WiFi, as observed in our previous work[8]), and users of "so-called" unlimited dataplans face maximum caps for data transfer. Future wide-area wireless networks may not suffer from these limitations, and could provide an excellent environment for Merlion to function.

Virtualization: In the system model described in the paper, Merlion requires strong network connectivity to the server. Specifically, the server running the unmodified application must always be accessible by the designer and runtime. An alternate model, utilizing virtual machines, could alleviate this limitation. A mobile device of the future could conceivably run the virtual machine image of a desktop operating system. The Merlion runtime, also executing on the phone, can interact with the virtual machine to provide users with a UI that is more appropriate for mobile form factors. Such an embodiment would not impose any additional requirements for network connectivity beyond that which is needed for the application itself. However, this would require a hypervisor capable of running desktop VMs to operate on mobile phones.

Accessibility and look-and-feel: As Merlion operates at the framebuffer level and deals with pixel data, it loses higher level information embedded within user interface controls. For instance, in most Windowing systems, a button can be queried for its text. This information can be used by software such as screen readers to provide additional functionality for users with special needs. In addition, user interface components (such as buttons) retain the look-and-feel of the Windowing system of the server. If a mashup is created from multiple applications running on different platforms, this disparity in visual decorations could distract users. A more complex implementation can alleviate both the challenges to accessibility and improve look-and-feel. Such an implementation could run a component called the *Merlion Introspector* on the server, which queries the user interface components of applications running on it, and relay this information to clients. The functionality of the Merlion runtime would also have to be expanded to support changing look-and-feel of UI components, and to expose a programmatic interface that could be consumed by assistive software running on the mobile device.

Scrolling: In the current implementation of Merlion, a user must select regions around scroll buttons. Given how pervasive scrolling is in applications, we believe that additional functionality could be provided by Merlion to simplify scrolling.

Updates to the application's user interface: A key assumption in Merlion is that the user interface of applications remains static. While this is true for most desktop applications, the assumption is violated for two classes of applications. First, some desktop applications such as Microsoft Word, attempt to improve the user's experience by showing truncated menus that only consist of commonly used menu items. Of course, this list changes based on use, and breaks the functionality of our system (it should be noted that in the case of MS Word, users can choose to always show full menus and this alleviates the problem). Second, web applications are notorious for having user interfaces that change[10, 2]. This is a problem faced by all customization systems for web pages. However, Merlion does impose a new set of challenges in that since we only transmit framebuffer contents from the server to the runtime, we lose information such as the document object model (DOM) tree underlying a web page. This can be addressed by either having the server make the additional information available and the client smart enough to consume it, or alternatively, to save the image of visual regions selected by the user at design time and use heuristics to determine how the selected regions have moved.

Stateful application user interfaces: The UIs of complex applications (e.g. the ribbon UI in Office 2007) may not expose all possible operations at the same time. For example, in Word 2007, the command to insert a table is part of the "Insert" group while that for changing text spacing in a paragraph is part of the "Page Layout" group. If a mashup application wants to expose both commands at the same time, we need some additional functionality. Instead of just allowing the user to select and arrange visual regions in the Merlion designer, we need to let her add placeholder GUI elements (e.g. buttons) not part of the original application. At runtime, interacting with these placeholder elements would invoke a sequence of operations on the original application to achieve the desired operation.

Frame rate and screen resolution: In our current implementation of Merlion, we do not compress the transmission of pixel data from the server to the client. Despite this, we observed acceptable frame-rates (around 25 per second) because our mobile device had a screen-size of only 240×320 pixels. It is not clear if screen resolutions for mobile devices will increase significantly in the future. Some trends, such as the large form-factor of the Kindle and upcoming devices that support HD video, suggest this might be the case. These trends would certainly pose challenges for Merlion but it is likely that these can be overcome by applying low-latency compression algorithms.

7. RELATED WORK

PageTailor[10] and Highlight[11, 13] are two systems that enable customized page layouts of web pages for improved usability on mobile devices. Both systems assume that the end-user utilizes a visual tool to easily customize the original web page (as rendered by a desktop browser) into a form that is more appropriate for a mobile device. Merlion shares this approach of having an initial design phase. While end-users are required to pay the cost of customizing up front, several factors alleviate this burden. First, the customization is done visually and is arguably feasible for non-programmers[16]. Second, customizations can be reused across users. For instance, there is a thriving community of

users[7] who share scripts for the GreaseMonkey[3] plugin for Firefox. It should be noted that PageTailor and Highlight only target web applications, while Merlion is more general and works for any application with a graphical interface.

The Virtual Network Computing (VNC) system was developed in the late 90s to allow a user to remotely connect to the graphical interface of a desktop computer[14]. In VNC, a client requests the latest pixel data for any part of the graphical display from the server, and renders it locally. The client also captures any local user input events (mouse, keyboard, etc.) and forwards these to the server. The server, in turn, applies these input events to the desktop. The protocol used by VNC is called the remote framebuffer (RFB) protocol, and is relatively simple to implement[6]. Systems such as THINC[9] provide high-performance, and can support playing video and audio in LAN and WAN environments. Another relevant work, the FlashProxy[12] system, allows multimedia applications requiring specialized plugins to work in any web browser. It renders frames of the application on a remote server, which does have a web browser with the required plugin, and relays frames from the application to the client device.

Merlion is related to the WinCuts[15] project. One key difference is that we enable application mashups. The Merlion designer allows visual regions to be brought together from several applications and across platforms. Moreover, the designer allows the end-user to modify the layout of visual regions to better suit the constraints of their mobile device. Another difference is our method of retrieving graphical data from a server's framebuffer, and using this as the basis for building mashup applications. This functionality can be thought of as a lowest-common-demoninator feature across all graphical platforms, and allows greater flexibility in creating mashups that span platforms and applications.

In recent years, desktop operating systems have provided a feature (web slices on Windows and web clips on MacOS)[4, 1] that allows a user to select part of a web page, and quickly access the state of that visual region. This feature is similar in spirit to Merlion, and in fact, we used Merlion to implement an add-on for the Windows Vista sidebar. Using Merlion provides interactive capabilities, use in applications other than web pages, mashups of several applications and customized layouts for mobile form factors.

8. CONCLUSION

We presented Merlion, a system that enables end-users to create custom mobile applications from existing desktop applications. Users can select desired visual regions from existing applications, alter their layout to better suit a mobile display, and set up transitions. We described a prototype implementation of both the Merlion designer and runtime, presented scenarios enabled by Merlion, and discussed challenges that can be addressed by the research community to enable the vision of the paper.

9. REFERENCES

[1] Apple: Creating Web Clip Widgets. `http://www.apple.com/pro/tips/webclip.html`.

[2] CNN Technology Article: changes coming in response to user complaints. `http://www.cnn.com/2009/TECH/03/25/facebook.changes/index.html`.

[3] Greesemonkey. `http://wiki.greasespot.net/Greasemonkey`.

[4] Internet Explorer 8 Features: Web Slices. `http://www.microsoft.com/windows/internet-explorer/features/easier.aspx`.

[5] Nintendo DS homebrew. `http://en.wikipedia.org/wiki/Nintendo_DS_homebrew`.

[6] The RFB Protocol (Version 3.8). `http://www.realvnc.com/docs/rfbproto.pdf`.

[7] Userscripts.org: Power-ups for your browser. `http://userscripts.org/`.

[8] Mahesh Balakrishnan, Iqbal Mohomed, and Venugopalan Ramasubramanian. Where's that phone?: geolocating ip addresses on 3g networks. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 294–300, New York, NY, USA, 2009. ACM.

[9] Ricardo A. Baratto, Leonard N. Kim, and Jason Nieh. Thinc: a virtual display architecture for thin-client computing. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 277–290, New York, NY, USA, 2005. ACM.

[10] Nilton Bila, Troy Ronda, Iqbal Mohomed, Khai N. Truong, and Eyal de Lara. Pagetailor: reusable end-user customization for the mobile web. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 16–29, New York, NY, USA, 2007. ACM.

[11] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1719–1728, New York, NY, USA, 2008. ACM.

[12] Alex Moshchuk, Steven D. Gribble, and Henry M. Levy. Flashproxy: transparently enabling rich web content via remote execution. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 81–93, New York, NY, USA, 2008. ACM.

[13] Jeffrey Nichols and Tessa Lau. Mobilization by demonstration: using traces to re-author existing web sites. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 149–158, New York, NY, USA, 2008. ACM.

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

[15] Desney S. Tan, Brian Meyers, and Mary Czerwinski. Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1525–1528, New York, NY, USA, 2004. ACM.

[16] Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444, New York, NY, USA, 2007. ACM.