Highlight: End User Mobilization of Existing Web Sites

Jeffrey Nichols, Zhigang Hua, Tessa Lau, John Barton
IBM Almaden Research Center

Abstract

Today's web pages provide many useful features, but unfortunately nearly all are designed first and foremost for the desktop form factor. At the same time, the number of mobile devices with different form factors and unique input and output facilities is growing substantially. The Highlight re-authoring environment addresses these problems by allowing users to start with existing sites they already use and create mobile versions that are customized to their tasks and mobile devices. This "re-authoring" is performed through a combination of demonstrating desired interactions with an existing web site and directly specifying content to be included on mobile pages. The re-authored web sites can be deployed to mobile devices through a new server-side architecture that makes use of a remote control metaphor in which the mobile device controls a fully functional browser that is embedded within a proxy server. The system has been tested successfully with a variety of existing sites. A study showed that novice users were able to use the system to create useful mobile applications for sites of their own choosing.

Introduction

Use of the web from mobile devices is becoming increasingly popular [11], however only about one-third of all mobile web users are satisfied with their experience [13]. While more web sites now have mobile versions, these sites are often designed for use on the least functional mobile device, and provide only a subset of the functionality of the original site. Moreover, the cost of producing mobile versions is such that only the most widely-used applications have them. Less popular applications, such as most enterprise software, are rarely supported on mobile devices. Employees who need to perform tasks on the road are left with an extremely limited ability to conduct work processes on their intranets.

Our Highlight system enables end users to *re-author* mobile web applications from existing web sites simply by demonstrating how to complete their task in a desktop browser. Highlight uses the *trace* of a user's interaction with an application as the basis for creating a task-specific mobile version of that application. By interacting with only the controls needed to accomplish a task, a user defines the set of controls that should be surfaced in the mobile web application. Unfortunately, traces are not always sufficient to capture the richness of interaction needed. Our Highlight Designer tool lets users interactively "clip" portions of the original website for display in the mobile version, and generalize flow by specifying additional paths through the application.

A key aspect of Highlight is that it leverages users' existing knowledge of web sites. We believe that many of the tasks users perform on the web are repetitive, particularly those performed in enterprise web applications. A user that is an expert at using a particular web site is uniquely positioned to know how the site is used and what features of that site could be useful in a mobile version. By making the authoring interfaces as simple as demonstrating how to perform a task using a desktop browser, we are lowering the barrier to creating mobile web applications. This enables end users to create their own

customized mobile web experience, optimized for the tasks they need to perform and the mobile device they use to access the content.

In summary, this chapter describes the following contributions:

- Algorithms for converting the trace of an interaction into a mobile web application;
- Highlight Designer, an implemented interactive tool for creating and modifying mobile web applications; and
- An empirical evaluation, showing that end users were able to create useful applications with Highlight, and that our approach saves significant bandwith over existing web applications.

We begin by putting Highlight in context with the related work in this area. Then we describe Highlight's user interface through a walkthrough of a user constructing a mobile version of the amazon.com interface. The next section describes how we implemented some of the key features of the authoring environment, followed by a discussion of Highlight's architecture. In the next section we evaluate Highlight through an informal study of novice users creating applications using the system, in terms of the breadth of existing sites that it can support, and the benefits to users through use of a Highlight mobile application as compared to existing web pages. We conclude with a discussion of the current system and directions for future work.

Related Work

Early work on creating mobile interfaces focused primarily on two approaches:

- Automatically modifying existing interfaces based on heuristic rules or machine learning algorithms
- Creating tools that allow web site builders to model their site and use those models to create new versions of their site for multiple mobile devices.

Many of the first systems to create mobile interfaces attempted to use the automatic approach. Digestor [3] used sets of heuristic rules to modify pages, such as "replace each text block with its first sentence." While Highlight does not use the same automatic approach, it may be useful to support some of the operations suggested by Digestor, such as the sentence replacement rule suggested above. Other automatic approaches have analyzed users' browsing history to improve mobile interfaces (e.g. [1]), such as by increasing the prominence of links that users often follow. Highlight relies on users' recollections of their browsing history to pick the most useful elements of the web site, and thus the authoring environment might be augmented by including some indication of previous history in its interface. These automatic schemes were also limited to making small changes to the interface, whereas Highlight can make radical changes because the user is directly involved in the process.

Other systems have used a model-based approach to creating mobile web interfaces. Vaquita [5] provides a tool and some heuristic rules for reverse engineering a web page into a XIML presentation model that could later be transformed for use on other devices. In contrast, the MDAT system [2] starts with the designer creating a generic interface model for their web page and then provides tools to transform the generic interface for use on a variety of different devices. Unlike Highlight, these systems require significant knowledge of abstract modeling and programming to use.

A few projects have investigated the idea of allowing end users to create their own user interfaces from those found on web sites. Clip, Connect, Clone for the Web (C3W) [8] is a system that allows users to clip elements from existing web pages onto a separate panel and then link the elements together to create useful combined applications. Unlike Highlight, however, interfaces created in C3W exist entirely on one page and were not designed to work on a mobile device.

d.mix [9] allows users to create mash-ups by combining elements found in existing web applications. It supports creation of mobile interfaces, but this appears to require users to create and edit scripts written in the Ruby programming language.

Another relevant system is Adeo [7], which allows mobile phone users to run previously recorded macros in a web browser on a remote machine and receive back the results on their phone. Inputs can be provided to the macros at the start, but otherwise macros run completely independent of the phone and only return when they have completed. Adeo's architecture has an element of the remote control metaphor used by Highlight, although in Adeo the browser is treated more like a function that returns a result instead of as an interactive entity. We believe that Adeo could be built using Highlight, however the reverse would not be possible.

PageTailor [4] is a tool that allows users to remove, resize or move web page elements while browsing on a mobile device. The tool runs directly on the mobile device, and studies have shown that its modification algorithms are robust to website changes over long periods of time. While PageTailor can modify the content of pages, it does not allow users to specify the transitions between pages as Highlight does. PageTailor also requires the mobile device to download most of the content of every web page, because the modification algorithms are run directly on the mobile device. Highlight only requires the mobile device to download the content required for the user-designed application because its modification algorithms are run on a proxy server.

Common to all of these approaches, and the approach of Highlight, is the idea that content will need to be modified for use on the mobile device. Researchers have also been designing new interaction techniques in an attempt to replicate the experience of browsing a web page on a typical PC browser within the constraints of the mobile browsing environment. The Apple iPhone's multi-touch interface is a particularly good example of this work. While these techniques have had some success, we believe there is still a place for content modification approaches such as that of Highlight. Modified interfaces should always be smaller and easier to navigate than regular web pages. If the modified pages contain the correct content and features, then they are likely to be easier to use. We hope that by involving the user in the design process, the modified pages will contain the correct content in an easy-to-use format.

The Highlight System

Re-authoring is performed through the Highlight Designer extension to the Firefox web browser running on a typical PC. A user begins by visiting an existing site and demonstrating an interaction with the site to include in the mobile application. As the user interacts, the Designer automatically adds content to the current "pagelet," a mobile version of the current page, and creates new pagelets as needed. While demonstrating an interaction, the user may choose to explicity add additional content from the existing

web page and rearrange or remove elements already in the current pagelet. A storyboard-style interface gives the user a visual overview of the application and allows the user to return to previous locations in the mobile application. This enables the user to demonstrate alternate actions that might be taken while interacting with the mobile application or to help the Designer generalize its knowledge of interactions that were previously observed. Once the user has finished, a description of the application can be saved to a proxy server that will allow access to the application from mobile devices.

A novel feature of Highlight is that it allows users to author both the content of mobile pages and the sequences of interactions that characterize the transitions between pages. This feature allows users to create mobile applications with a page structure that differs from that of the existing web site. For example, a mobile application can skip over unnecessary web pages, allowing users to perform a minimum of interaction in order to complete their task.

Interface Walkthrough

We illustrate the use of our system by walking through an example scenario in which Amy creates a mobile application using the Highlight Designer to buy a single item from amazon.com.

The Designer opens in a Firefox sidebar to the left of the main browser window (see Figure 1a). This sidebar contains two main parts: a storyboard, in the top pane, which contains a node-and-arrow overview of the application being created, and the preview browser showing the current pagelet in the mobile application. Amy starts by typing “amazon.com” into the browser's location bar and loading the retail site. Highlight records this event as one of the initial events required to set up the application. She then selects “Books” from the category drop-down list, types her search term into the search box and presses the “Go” button to initiate the search. As she performs these actions, the Designer automatically clips the widgets with which she is interacting, as well as a label or other descriptive text that explains their use. Thus, the Designer adds the drop-down list, the search box, labels for both, and the search button to the current pagelet, and those items are displayed in the sidebar’s preview browser. All of these clips are made automatically, just by virtue of Amy's trace of interactions with the web site.

The search resulted in a list of hits being displayed in the main browser. Amy wants to clip all of these results for display in the mobile application. She clicks the “Add Content” button in Highlight. Now, as she moves the mouse over the browser, portions of the web page are highlighted in red, indicating the region of the page that would be clipped. Amy moves the mouse such that all of the results are contained within the red box, and clicks to clip that region. These search results appear in the preview browser. In addition, a new pagelet is automatically constructed (“Search results”), and added to the storyboard. The storyboard now contains two pagelets, one containing the search interface, and a second containing the list of search results (see Figure 1a).

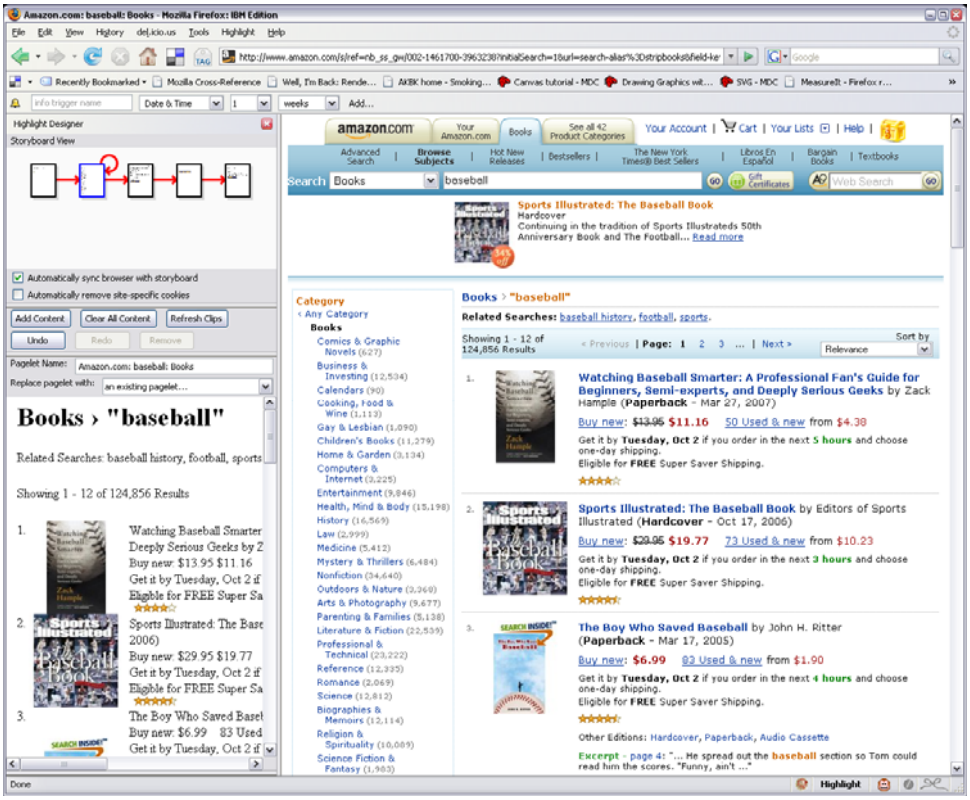
Next, Amy clicks on one of the items for sale. The item page is displayed in the main browser. On this page, she is interested in the item information such as the name and the purchase price. She uses the “Add Content” button again to clip the region containing these details to her application, adding a third pagelet (“Item details”) to the storyboard.

Amy wants to be able to purchase the item with her application, so she clicks the “Add to Shopping Cart” button on the item description page. The next page is an overview of the current shopping cart. Amy’s goal is to create an application for buying a single item, so she decides not to include any content from this page and clicks on the “Proceed to Checkout” button. Highlight chooses not to create a pagelet that includes this button because the page would have only included one button. Highlight does not create such pages by default because they typically add an additional navigation step without any interactive value. The click on the “Proceed to Checkout” button is recorded as part of the transition to the next pagelet however, which ensures that it will automatically happen when the mobile application is executed.

The next page requires a login/password to continue. By typing in her amazon.com username and password, these widgets are automatically clipped and used to populate the fourth pagelet in the application (“Sign in”). On the next page, which is a confirmation of the order, she uses the “Add Content” button to create a new pagelet with information such as the final price of the order and the shipping address. At this point, Amy could click the “Place Your Order” button to place the order, which would also automatically add that button to her pagelet. Because she is just specifying an example however, she does not actually want to buy the item. Instead she uses the “Add Content” button in the sidebar to highlight the “Place Your Order” button and add it to the current pagelet, which completes the basic structure of her application.

At any point, Amy can test her application by double-clicking on the pagelets in the storyboard to return to previous pages. Currently, this is the only interaction supported with the storyboard interface. Returning to the “Search results” pagelet, the main browser navigates back to the search result page, and the clipped search results are refreshed in the preview browser. Amy clicks on a different item this time. When she does this, Highlight detects that she has performed an action that looks very similar to an action she performed previously, and asks if she wishes to generalize them. By saying “yes”, Amy is indicating that a click on the title link of *any* of the items in the search result pagelet should lead to the item detail pagelet. The next time she returns to the search result page and clicks on a different item, she will automatically be redirected to the appropriate “item detail” pagelet.

As a final step, Amy can add the functionality of navigating across multiple pages of search results, which is available on the amazon.com site by clicking links at the bottom of the page. Amy can start adding this functionality by returning to the search result pagelet in the storyboard, and then clicking on the “Next” link at the bottom of the search results page. Clicking on the link takes the main browser to the next page of search results. Amy now has two options to create the interface that she desires. She could use the “Add Content” button to add the search results content to her pagelet, as she did previously. The Designer will recognize that this new pagelet is similar to the previous search results pagelet, and it will ask Amy if she would like to use her previous pagelet. Answering “yes” to this question creates a looping edge from the “Search results” pagelet back to itself. Alternately, Amy could have explicitly specified that the “Search results” pagelet should be used by selecting it from the drop-down list in the preview browser. To make the rest of the search page navigation links work, Amy can click one of the other links. The Designer will detect that this link click is similar to the “Next” click, and ask if she wants to generalize. By answering “yes,” Amy will tell Highlight to generalize all of the search navigation links.



a.

b.

Figure 1. a) The Highlight Designer running inside of the Mozilla Firefox web browser. This screenshot was taken from the "Search results" pagelet after construction of the amazon item purchasing example. b) An overview of the final amazon mobile application.

Through a mixture of demonstrating how she interacts with the application, and using clipping regions to select desired content on each page, Amy has constructed an application that allows her to search for and purchase items via a lightweight web interface suitable for use on mobile devices. When this interface is loaded into her mobile device (see Figure 1b), she will be able to search for items by name, navigate through the list of search results, see item details for a particular item, and purchase that item. The interface is optimized for the task that Amy wishes to do with this website, and contains only the subset of the amazon.com application that is relevant to her task.

Implementation

Highlight Designer works by recording the actions a user takes in the browser, and converting these actions into a mobile application description. Mobile applications are represented as a directed graph of "pagelets." Each pagelet represents one page that might be seen on the mobile device. Pagelets are described in two parts:

- *Content operations* that describe how the pagelet's content will be constructed from the content of the page on the existing site.
- *Transition events* that describe the navigation element in the pagelet that causes a transition to the next pagelet. These events also store the sequence of interactions that were demonstrated on the existing web site to reach the page from which the next pagelet's content will be clipped. Each transition is represented by an arrow on the storyboard view.

Content Operations

The most common content operation is extracting some content from the existing web page and adding it to a pagelet. When a user interacts with a form field, such as a textbox or a radio button, this field is clipped and added to the Highlight application. In addition, a descriptive label is generated for some elements whose function is not obvious from their appearance alone; these include text boxes, dropdown list boxes, check boxes, and radio buttons. The label is determined by first looking for labels or captions specified in the HTML; if these are not present, heuristic rules (borrowed from CoScripter [10]) are used to extract textual content close to the target element that might plausibly constitute that element's label.

Content can also be clipped using the "Add content" tool, rather than by directly interacting with the page. This form of clipping is used to add read-only content to the mobile application, such as a flight status or a weather report, or to add multiple related interactive elements simultaneously, such as the search results in the amazon application. Another use is to add content for future use that should not be activated at this time, such as the "Place your order" button in our amazon scenario. When the "Add content" button is selected, moving the cursor around on the web page causes a red box to be drawn around the HTML element currently in focus. By moving the mouse, the user can select the target element to be clipped. Multiple elements can be clipped by invoking "Add content" for each item.

The Designer also supports the "move" and "remove" content operations, which allow users to modify content already added to the pagelet. These operations are supported by interactions in the preview browser. Clicking on an item will select it, and then that item can either be dragged to move it to a new location or removed by clicking the "Remove" button.

Transition events

The Designer also records the series of interactive steps that the user demonstrated in order to transition from one pagelet to the next and stores this in a transition event. For example, in the amazon.com walkthrough the transition event from the "Item details" pagelet to the "Sign in" pagelet would contain the steps "click on the Add to Shopping Cart button" and "click on the Proceed to Checkout button." The following steps would have been recorded from the "Sign in" pagelet to the final "Confirm Order" pagelet: "enter <username> into the Login: textbox," "enter <password> into the

password textbox," and "click the Login button." Note that all of the user's operations are stored in the transition event, even those that may have caused some content to be clipped into the pagelet. All of these steps are included in the transition event so that the Designer can keep the browser and application in sync when the user double-clicks in the storyboard interface. Each transition event contains steps that were recorded from when the user created or navigated to the current pagelet and end when the user creates or navigates to a new pagelet.

Identifying Elements On A Web Page

Both content operations and transition events must be able to identify web page elements in a repeatable manner. This allows the same content to clipped for a pagelet every time it is shown and allows the steps specified by an event to be replicated properly on the web page.

The Highlight Designer uses a combination of two approaches to identify web page elements: XPath expresssions [6] and a heuristic representation ("slop") pioneered by the CoScripter system [10]. XPath has the capability of precisely describing any element on a web page, but its expressions are often prone to failure if the page structure changes even slightly. CoScripter's slop uses textual descriptions and heuristics to identify elements on the page (e.g., "the Advanced link" refers to an <a> element containing the text "Advanced"). CoScripter slop is much more robust to page changes, however it was designed to identify small functional elements, such as textboxes and buttons, so it is not capable of describing non-interactive content regions on the page. Because slop interpretation is heuristic, it is possible that in some cases the interpreter will produce an incorrect match, creating an application that does not work even though it might appear that it should.

Slop is a good match for transition events because it was designed to represent traces of interactions with web pages. Currently we record both the slop representation and an XPath expression for each event. If the XPath expression fails to find the correct element, then we can recover by trying to interpret the slop instead.

However, slop is less useful for representing content operations such as clipping a region of the page. Because CoScripter's focus has been on capturing user interactions with a web page, it does not contain instructions for selecting arbitrary content on the page. Although we have enhanced the slop interpreter in Highlight to be able to understand human-written instructions, such as "clip the table containing Flight Status", we have yet to devise intelligent algorithms for recording slop based on user demonstrations. Thus, Highlight relies on XPath expressions to specify elements that are the targets of content operations. It is an area of future work to incorporate more robust methods for describing regions to be clipped.

Generalizing Transition Events and Pagelets

We observed early on in the development of the Designer that many web applications have repetitive page structures, such as the search pages we saw in the amazon.com application example. There are two types of repetition that we wanted to support with the Highlight Designer:

- Some sites have multiple paths to get to the same type of page. For example, the "Item details" page on amazon.com can be reached both from searching and browsing through the site's product

hierarchy.

- Some pages, such as pages of search results, contain repetitive blocks of content. Often there will be similar interaction elements in each of these blocks, such as a link on a heading, that lead to a similar page.

To support creating a mobile application for web sites with these characteristics, we wanted to add a set of lightweight interactions that would allow users to specify that an existing pagelet should be re-used and that a set of links all lead to the same pagelet.

We created two techniques for allowing users to specify that a pagelet should be re-used. These methods are both needed once the user has navigated to a new page and is about to clip content to create a new pagelet. The first method is explicit. If the user immediately recognizes that they wish to re-use a pagelet, then they can select that pagelet from a drop-down list in the sidebar and the existing pagelet will immediately be applied. The second method is implicit. If the user does not recognize that an existing pagelet might be re-used, then they will begin clipping content from the new page into the pagelet. The Designer will analyze the XPath locations of the content as it is clipped, and if it appears to match a previous pagelet then the system will offer to replace the new pagelet with the old one. To reduce annoyance, the system will only ask this question once for each new pagelet.

In order to specify that a set of links all should lead to the same pagelet, the user must first specify a trace using one of the links in the set. In the amazon.com example above, remember that Amy clicked on the first result and created a pagelet for the result item before returning to the search results pagelet. When the user returns to the results pagelet and clicks on another link in the set, Highlight will analyze the new event and compare it to previous events. Specifically, it will look for similarities in the event interactions that caused the mobile pagelet to advance to a next page.

Our current algorithm for detecting similarities in events is as follows. First, we test to see if the events were of the same type. Clicking a link cannot be the same as pressing a submit button, for example. Second, we examine the XPath expressions of the elements involved in the two events. For example, in the amazon.com example the XPaths for the two search result links were:

First link: `/HTML[1]/…/TBODY[1]/TR[1]/TD[1]/A[1]`

Second link: `/HTML[1]/…/TBODY[1]/TR[2]/TD[1]/A[1]`

The Designer considers events to be generalizable if the XPath expressions differ only in the indices, such as for the `TR` element in the amazon example. This means that the elements are located in much the same place in the web page but are offset by some repetition of structure. Often this will occur when items are located in the same relative location within different cells of a table. The particular indices at which the elements differ are saved.

If the events are generalizable, then the system will identify the pagelet to which the previous event leads and ask the user if that pagelet should be the target for all similar links. If the user says “yes,” then the two events are combined into a single event. The new event remembers the XPath indices that differed between the two events that created it and any future interaction with an element that has an

XPath that differs only in those indices will cause the mobile application to follow that event. In our experience, this mechanism has worked well for a variety of search result pages, and has also been shown to be useful in other contexts, such as category browsing pages with many links. This heuristic does have limitations however, particularly in situations where repeating chunks of content are not completely identical. For example, some forms of eBay searches will return a list of results, but the format of each particular item in the list will vary depending on the auction type for that item.

Architecture

In order to make mobile applications available outside of the Designer, we have implemented a proxy server component that serves mobile applications based on existing sites. When the user wishes to access a mobile application, they navigate their mobile browser to the proxy server's main page, select their application from a list of available applications, and then proceed to use the application.

The proxy server is implemented as a typical web server that contains a fully functional Firefox web browser as a component [12]. Selecting an application establishes a session with the server and causes the proxy server's Firefox browser to automatically navigate to the page of the existing web site that corresponds with the first pagelet in the mobile application, execute the content operations of that pagelet, and return the HTML of this content to the mobile device. Subsequent requests from the mobile device are matched to a transition event for the current pagelet, any form data from the mobile device is filled in appropriately, and then the proxy browser advances to the next page based on the interactions specified by the event. Thus, the interface between the proxy server and the mobile device is similar to a remote control in that each request by the mobile browser specifies a series of user interface operations for the proxy browser to perform in order to get the contents of the next mobile page. For example, the proxy server might fill in some form fields, press the submit button, and navigate through several subsequent pages before the constructing the next mobile page.

The use of a proxy server provides several advantages. First, only the clipped content is sent to the mobile device, resulting in fast load times despite slow network connections. Second, because the browser running on the proxy is a full-fledged desktop browser, any "client-side" JavaScript from the existing web site can be executed in place, rather than relying on the mobile device's (often poor) JavaScript support. This feature enables the proxy server to serve mobile versions of Ajax applications, although the Highlight Designer does not yet support authoring mobile applications that make use of Ajax.

Integration with CoScripter

Since Highlight makes use of traces of user interaction with web applications to construct interfaces, we extended it to make use of a large repository of such traces as collected in the CoScripter project (formerly known as Koala [10]).

CoScripter is a programming by demonstration system for Firefox that records user actions performed in the browser and saves them as pseudo-natural-language scripts. CoScripter's representation for scripts is a plaintext language with steps consisting of commands such as "go to http://google.com", "type coscripter into the search box", and "click the Search button". These steps are both human- and

machine-understandable, resulting in scripts that are easy for people to understand yet interpretable by a machine.

Scripts are automatically saved to a central wiki for sharing with other users. The CoScripter community has created thousands of scripts for web-based tasks such as checking stock prices, creating trouble tickets, and managing queues in a call center [10].

This repository of scripts provides a wealth of information about the tasks people do with web applications. It also provides an excellent starting point for the creation of Highlight applications, particularly if we could enable CoScripter users to import their scripts into Highlight and, with little or no effort, be able to complete their tasks from a mobile device.

Thus, we added the capability to Highlight to create a new application from a CoScript. When a script is loaded, Highlight uses CoScripter's interpretation component to programmatically run through the script, clicking on buttons and entering text as if the user had done these actions directly. Meanwhile, Highlight records the actions and uses them to construct an initial application.

One piece missing from the CoScripter language was the ability to specify portions of the content of the web page to be clipped, akin to using Highlight's “Add content” tool. Thus, we extended CoScripter's language with an additional type of instruction to describe regions to be clipped. These instructions take the form “clip the table containing Flight Status”, and are parsed by Highlight and turned into an XPath expression that select the smallest table element in the document that contains both the words “Flight” and “Status”.

With this addition to the CoScripter language, users are able to import scripts directly from the CoScripter wiki into Highlight and have a fully-functional mobile application with no additional authoring work. Once the script has been imported into Highlight, any of Highlight Designer's interactive features can be used to modify the application in order to customize it further for one's mobile device.

This integration reduces the cost of authoring mobile applications even further, because Highlight users are able to take advantage of already-created scripts for completing common tasks on the web.

Evaluation

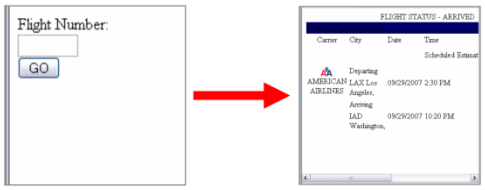
We have evaluated the Highlight Designer by informally testing it with several users and by performing an empirical comparison of Highlight interfaces with the corresponding unmodified interfaces.

Informal User Study

We conducted an informal user study to help us understand whether users would understand interacting with the system and be able to create applications of their own.

Three subjects from our research lab participated in the study. Two of the subjects were male, one was female, and all three came from different age groups. The subjects were not regular users of the mobile web, either because they did not own devices that were capable of it or they did not believe its benefits were worth paying for the service. They all were able to recall instances in which they would have liked

a.



b.

Figure 2. Overviews of the pagelets and structure of the (a) google image search and (b) AA flight status mobile apps.

to access the web in a mobile setting however, for example to look up nearby restaurants, get directions, get movie times for a local theater, view current traffic, or access web-based e-mail.

We gave the subjects a very brief verbal introduction to the system and then asked them to create two mobile applications that would allow them to explore the system's capabilities. For the first application, we asked subjects to create a two-pagelet mobile application from mapquest.com. The first pagelet was to contain the fields for entering an address and the second pagelet was to contain some information about that location. Note that we would have asked our subjects to clip the resulting map, but the maps on mapquest.com are generated from client-side JavaScript and are not easily replicated by simply copying their HTML. At the time, we did not have Designer support for clipping portions of a web page as an image.

The second application was a three-pagelet simplified version of google image search. The first pagelet was to contain the search textbox and button, the second pagelet was to contain the grid of search results, and the final pagelet was to contain the full-size version of an image selected from the results (see Figure 2 for a finished example of this application). Subjects were required to create a mobile application that skipped over a page of the existing site (an extra page exists in google image search to show the image in the context of the page in which it was found) and to use the generalization features of the Designer in order to complete their application.

All of our subjects were able to build the first two applications, though there was a clear learning curve to using the tool. In particular, users were initially unclear on when to expect the system to add content for them as compared to when they should add content explicitly. After creating the applications, users seemed to form a clearer model of what the system was capable of doing automatically. We also found that the Designer's capabilities for allowing users to make and then fix mistakes were lacking. Users would often explore a site to understand the possible interactions before choosing one for their

application. While the Designer has some capabilities for changing and rearranging the content within a pagelet, it needs more facilities for changing the storyboard structure after the initial recording.

After creating the two applications, we asked users to think of an interesting mobile application that they would like and then try to create it using the Designer. With just this instruction, our subjects were able to successfully create applications for the San Francisco Chronicle's Bargain Bites web site and the weather.com 10-day forecast. The subject who created the Bargain Bites web site was particularly happy, because the subject had been manually creating a mobile version of this site and syncing it with a PDA.

Our third subject attempted to create a traffic application from the www.beatthetraffic.com web site, but unfortunately this site used a great deal of client-side JavaScript code that the Designer was not able to correctly interact with in order to playback the application. A different subject also attempted to create a mobile application from gmail, but gmail's Ajax features prevented the Designer from working correctly.

Although the subjects occasionally ran into difficulties while learning to use the system, they encountered few problems with the Designer's UI while creating their third application. At the conclusion of the study, all of the subjects reported that they were excited about the technology and wanted to use it in their everyday lives.

Breadth and Benefits

Using the Highlight Designer, we have created working mobile applications from both popular and niche web sites. These sites include aa.com, amazon.com, ebay.com, google.com, mapquest.com, sfgate.com, weather.com, and many others. In our experience, the Designer works best with pages that use a minimal amount of client-side JavaScript. The Designer seems to work with pages that use scripting for small UI features, such as highlighting an item when the mouse moves over it, but will likely break if the page's DOM is manipulated or new content is loaded via an XmlHttpRequest. We are exploring how to extend the Designer to work with pages that contain these features.

In order to understand the benefits of using a Highlight mobile application, we compared performing a task using a Highlight mobile application to performing the same task using the original interface on a mobile web browser with desktop browser features, such as Minimo or Opera Mobile. Some of the problems with mobile web browsing include small screens that are only able to display a few UI elements at a time and slow networks and processors that result in delayed page rendering.

The small screens found on today's mobile devices make it difficult to navigate interfaces that have many clickable elements. For example, the front page of American Airlines' web site has 298 distinct elements that can be clicked on or interacted with. One benefit to using Highlight is that task-driven mobile interfaces can reduce screen clutter to only those controls that are necessary for the task at hand. To measure the value of this claim, we calculated the total number of interactive elements (form fields, links, and buttons) that are displayed in the original application versus the Highlight application, throughout the course of performing each task (see Table 1). While not a perfect measure, this number

Table 1. A comparison of interactive elements and the amount of data downloaded for applications created using the Highlight Designer and the desktop web sites used for the same task.

	Interactive Elements		Size (kB)		Percent
Description	**Original**	**Highlight**	**Original**	**Highlight**	**Size**
Check status of AA flight	736	3	711	3.6	0.5%
Update Facebook status	217	5	296	0.5	0.2%
Find nearby Wi-fi hotspot	74	18	1072	2.8	0.3%
Get weather in my area	486	6	1079	7	0.6%
Sprint cellphone usage	175	6	739	4.6	0.6%
Log today's exercise	128	4	393	0.9	0.2%
Update Fitday food diary	169	38	145	12.7	8.8%
Get calories for food	88	16	63	11.5	18.3%
Real estate in my area	274	35	1036	194.1	18.7%
Show trip itineraries	77	17	726	42.7	5.9%
Find Amazon book price	823	4	844	4.1	0.5%

approximates the complexity of the interface in terms of the number of options the user must sift through in order to complete her task.

The results show that the number of interactive elements in the Highlight application is drastically reduced compared to the original application. The reduction is greatest for large, multi-purpose web sites where Highlight's task focus makes it possible for the user to concentrate on the elements that are required for the task at hand, and ignore all the elements used to access irrelevant portions of the application.

Another problem with mobile web browsing is slow networks and costly page rendering. To show how Highlight addresses this problem, we have measured the amount of bandwidth required to download the set of pages (and associated content, such as images and scripts) required to complete a task using Highlight vs. the original website (see Table 1). All Highlight applications were significantly smaller than their unmodified counterparts. The applications that exhibited the least reduction in size were all ones that forwarded lists of results to the mobile device (e.g., choices of food items or listings of real estate)—all of which represented data that would have had to be transmitted to the client device in any case.

Discussion and Future Work

In this chapter, we have described a method of designing mobile web interfaces based on user demonstrations of an interaction trace through an existing web site. While the Designer allows users to then expand upon and improve their mobile application, we have observed that quite often a single trace is sufficient for creating a useful mobile application. This is especially apparent through the integration of Highlight with CoScripter. CoScripts are linear traces by nature, and we were able to find a number of existing scripts that could be turned into useful applications (many of which are listed in Table 1).

Of course, without the Designer it would be impossible to create applications with non-linear structures or with the ability to perform a search and navigate through the results. An important capability of the Designer is allowing the user to demonstrate two different interactions and have them generalized across a larger set of possible interactions. Our current generalization scheme is a heuristic based on the format of XPath expressions, which has been successful but could be improved. In particular, we would like to design an algorithm that detects repetitive chunks of content in the pagelet and then generalizes based on the detected repetition.

A current limitation of the Designer is its inability to support automating sites that use a lot of client-side JavaScript, particularly Ajax sites. We believe that it may be possible to extend the Designer to support these sites by extending it to also record the changes that occur in the website as the user interacts. Currently, the Designer only records the user's input and makes assumptions about the changes that can result in the interface based on the operation. An example assumption is that a new pagelet is only needed after a user clicks on a link or presses a submit button; these events typically trigger a new page to be loaded. For an Ajax site, a new page may never be loaded and the transition to a new pagelet may occur following any type of interaction. Recording changes to the web page may allow us to track when pagelet changes should occur and allow the creation of applications from Ajax sites.

With many data entry tasks, some fields will always contain the same value (e.g., your address) while some fields (e.g., the item you are searching for) will change each time the task is run. Currently, Highlight prompts the user to enter data into every field in the mobile application, placing an unnecessary burden on the user. However, in future work we plan to improve this process by learning from the user's previous behavior on this task and identifying fields that always contain the same value on every run through the task. One option would be to pre-fill form fields with the most common value, enabling the user to change it if necessary but accept the default with no effort; another more drastic change would be to remove constant-valued fields from the pagelet displayed to the user, while still filling in the target value on the proxy server.

We found in our user studies that the user interface of the Designer is not very forgiving when users make mistakes or want to explore the existing web site. One of our initial assumptions was that users would be experts with the sites they are mobilizing, but this may not always be the case and certainly it should be possible to recover from mistakes. We may be able to address some of these issues by providing more interaction through the storyboard interface, such as by allowing extra pagelets to be removed. It may also be valuable to provide a separate interface that shows a verbal description of the interaction steps that have been recorded, similar to CoScripter, and allows the user to edit their application at a different level.

The Highlight Designer is explicitly not a model-based tool. When users create applications however, they do identify both the interactive elements of a useful application and the contextual information that is needed to use the application. This information could be useful for creating a model of the web site. If multiple applications were created for the same site, then we might be able to combine information from those applications to create a more detailed model. Such a model could allow previous work in model-based research to be more easily applied to existing sites.

Conclusion

We have presented Highlight, a system enabling end users to create task-based mobile web applications simply by demonstrating how to perform a task on an existing application. Highlight uses the trace of a user's interaction to automatically clip the relevant controls for presentation in the mobile application, and enables users to visually point and select non-interactive content for inclusion in the application as well. Moreover, we have integrated Highlight with an existing repository of traces from the CoScripter system, which can be used to create Highlight applications with little or no additional effort. An informal user study shows that novice users are able to use Highlight to create useful mobile applications of their own choosing. An empirical evaluation shows that for a broad range of tasks, Highlight is capable of creating an application that is smaller and easier to use on mobile devices. In short, Highlight enables ordinary users to mobilize their tasks and take them on the road wherever they go.

Acknowledgements

We thank our user study participants for helping us evaluate Highlight, and the CoScripter team for their valuable comments.

References

1. Anderson, C.R., Domingos, P., and Weld, D.S. Personalizing Web Sites for Mobile Users, in *Proceedings of the 10th international conference on World Wide Web.* 2001: 565-575.
2. Banavar, G., Bergman, L., Cardone, R., and Chevalier, V., An Authoring Technology for Multidevice Web Applications. *IEEE Pervasive Computing*, 2004. **3**(3): 83-93.
3. Bickmore, T.W. and Schilit, B.N. Digestor: Device-independent Access to the World Wide Web, in *Selected papers from the sixth international conference on World Wide Web.* 1997: 1075-1082.
4. Bila, N., Ronda, T., Mohomed, I., Truong, K.N., and Lara, E.d. PageTailor: Reusable End-User Customization for the Mobile Web, in *Proceedings of MobiSys.* 2007: 16-25.
5. Bouillon, L., Vanderdonckt, J., and Souchon, N. Recovering Alternative Presentation Models of a Web Page with Vaquita, in *Computer-Aided Design of User Interfaces III.* 2002: 311-322.
6. Clark, J. and DeRose, S., "XML Path Language (XPath), Version 1.0," 1999. http://www.w3.org/TR/xpath.
7. Faaborg, A., *A Goal-Oriented User Interface for Personalized Semantic Search.* Media Arts and Sciences Massachusetts Institute of Technology, 2006, Boston, MA. http://alumni.media.mit.edu/~faaborg/files/thesis/draft/complete/faaborg_thesis.pdf.
8. Fujima, J., Lunzer, A., Hornbaek, K., and Tanaka, Y. Clip, connect, clone: combining application elements to build custom interfaces for information access, in *Proceedings of the 17th annual ACM symposium on User interface software and technology.* 2004: 175-184.
9. Hartmann, B., Wu, L., Collins, K., and Klemmer, S.R. Programming by a Sample: Rapidly Creating Web Applications with d.mix, in *Proceedings of the 20th annual ACM symposium on User interface software and technology.* 2007: 241-250.
10. Leshed, G., Haber, E., Matthews, T., and Lau, T. CoScripter: Automating & Sharing How-To Knowledge in the Enterprise, in *CHI.* 2008: To Appear.
11. MDA, "Q3 2006 – Mobile Internet Figures Continue To Grow," 2007. http://www.themda.org/PressReleases/Page_Press_PressReleases_LatestStats.asp.
12. Nichols, J., Hua, Z., and Barton, J. Highlight: A System for Creating and Deploying Mobile Web Applications, in *UIST'2008.* 2008: 249-258.
13. OPA, "Going Mobile: An International Study of Content Use and Advertising on the Mobile Web," 2007. http://www.online-publishers.org./media/176_W_opa_going_mobile_report_mar07.pdf.