Playbook: Revision Control and Comparison for Interactive Mockups

Stephen Oney[1], John Barton[2], Brad Myers[1], Tessa Lau[2], and Jeffrey Nichols[2]

[1] Carnegie Mellon University
5000 Forbes Ave. Pittsburgh, PA 15213
{soney,bam}@cs.cmu.edu
[2] IBM Almaden Research Center
650 Harry Rd. San Jose, CA 95120
{bartonjj,tessalau,jwnichols}@us.ibm.com

Abstract. When designing interactive interfaces and behaviors, interface designers compare and contrast multiple design ideas, often creating and testing many intermediate user interface prototypes before deciding on a final design. However, existing interface prototyping and creation tools do not effectively let designers explore, compare, or keep track of older versions of interface mockups, implicitly making the assumption that the users of these tools will work with one design alternative at a time. To explore how to enable designers to work with multiple designs in a prototyping tool, we created Playbook, a new system oriented towards helping interface designers keep track of, compare, and create interactive mockups. In this paper, we describe Playbook and discuss ways that future prototyping tools can better support the workflow of designers.

Keywords: prototyping, mockups, interface design, versioning.

1 Introduction

In the intermediate stages of interface design, designers typically produce a large number of design artifacts: site maps, story boards, static mock-ups, interactive prototypes, etc. [1]. Although a number of surveys and empirical evidence have shown that designers need better tools to manage and evaluate these design artifacts [1][2][3], designers are still using ad-hoc versioning solutions, like manually renaming files [1][4]. This may be for two reasons: first, revision control systems do not effectively fit in with interface designers' workflows or the tools they most frequently use for creating mockups. Second, while methods for keeping track of and comparing static artifacts, like site maps, are relatively straightforward, there are no appropriate methods for keeping track of *interactive* artifacts, like interactive mockups, and this presents a significant research challenge.

Interactive artifacts are ill suited for tracking with traditional revision control methods because they are defined by both their appearance and behavior. Although many imperative languages conflate the two, interface designers should ideally be able to define and modify each independently. Having separate revision control repositories for appearance and behavior is not practical because the two aspects are

M.F. Costabile et al. (Eds.): IS-EUD 2011, LNCS 6654, pp. 295–300, 2011.

interdependent. Particular revisions of the interface behavior are only compatible with a subset of the versions of the interface appearance because, for example, the code for the behavior will be dependent on the presence of specific interface elements in the appearance. Thus, a revision control system for interactive prototypes should be able to keep track of not only revisions of appearances and behaviors, but also compatibility between the two.

In addition to keeping track of revisions, a revision control system should enable comparisons to be made across different revisions. For static artifacts, this is largely a solved problem, as evidenced by the large number of textual and image-based difference systems. For interactive artifacts, however, comparisons between mockups are more difficult to make in a meaningful way, beyond changes in appearance, or beyond textual differences in the code responsible for the behavior.

We created Playbook to explore solutions to the aforementioned issues of revision tracking for interactive interface mockups. To manage revisions of both appearance and behavior, Playbook keeps track of layered images that define the appearance of the mockup and "scripts" that define the behavior of the mockup. To effectively fit in with interface designers' existing workflows and tools, Playbook allows designers to upload layered images quickly and directly from Photoshop. To add interactivity to these static layered images, Playbook uses a scripting language inspired by that of CoScripter [5], where high-level scripts describe the behaviors of a mockup. Our scripting language describes behaviors on a sufficiently high level that one script may be applied to mockups with very different appearances. Playbook scripts are grouped around the specific "tasks" that they enable the user of the interface to perform. For example, one task for a mockup of a movie rental website might be "rent a movie." Scripts grouped under this task describe how different interactive mockups react as the user goes through the steps of renting a movie. If the interface changes dramatically between revisions, very different scripts may be required to describe how the interface behaves when the user is performing a task. To allow interactive mockups to be compared, Playbook allows for mappings between these scripts within a task to define equivalent parts of different scripts, as is shown in Figure 1. This paper demonstrates that design tools can help designers manage and compare different revisions of interactive prototypes and presents Playbook, a tool for creating interactive prototypes that embodies this idea.

2 Related Work

A number of tools have been created to enable the creation of interactive mockups of various fidelities, including Adobe's Flash Catalyst, DENIM [6], and Designer's Outpost [7]. However, our focus in Playbook is on how to enable revision tracking and comparison, rather than on how the interactive behaviors are defined originally. Playbook is only concerned with the language describing mockup behavior to the extent that it is simple enough to be used by interface designers without a programming background and allows for revision control and comparison between interactive mockups.

Other systems have been built with the intention of exploring and comparing designs. Cogtool [8] supports estimating expert performances on mockup user interfaces

and displays a grid of interface designs along with their performance in doing user-specified tasks. Whereas Cogtool allows interface designers to compare prototypes using task completion times, Playbook allows designers compare different prototypes more qualitatively by being able to interact with the prototypes side-by-side. Juxtapose [9] is another system that lets designers compare different possible designs side-by-side. However, Juxtapose only allows low-level user input events to be replicated, such as mouse clicks at a particular (x,y) location on the screen, when comparing different prototypes. This limits the differentiation that is permissible when comparing interactive mockups side-by-side with Juxtapose.

3 Design

Playbook mockups start out as layered Photoshop images. Each interactive element of the interface must be in a separate layer. When the designer has a mockup they are happy with, the next step is to upload that mockup to the Playbook server, which keeps track of every uploaded revision. To make this step as simple as possible, we created a Photoshop plugin that adds a menu item to the Photoshop interface's File menu that does this. After the user clicks this menu item, the file is uploaded and the user's web browser is opened, pointing to the Playbook web page, so it operates like a versioned save feature.

After uploading the layered Photoshop file to the Playbook server, the next step is to create scripts to make the mockups interactive. Every script is then classified under a "task" group, which describes, on a high level, what the scripts in that group enable the user to do on the mockup. For example, in a mockup for a clothing website, one could write a set of scripts for the task of buying a t-shirt. Every script in that task group would describe how a user would buy a t-shirt in a particular version of that mockup (click the 'mens' button, and then the menu overlay should appear…click the 't-shirts' button, and a list of t-shirts should appear, etc.). These scripts can be thought of as interaction traces through a mockup. Every script is a set of "behaviors" which consist of one "stimulus" and any number of "responses." A stimulus is a user action to which the mockup will respond. For example "mouseover the 'womens' layer" or "click the 't-shirt' layer." Each response is a simple reaction to a stimulus. Only two responses are currently supported: hiding and showing layers. While the set of responses is limited, evidence from popular prototyping tools such as Balsamiq shows that even with these simple responses, designers can mock-up many of the desired behaviors that appear in web interfaces.

When writing a script for the first time, every behavior is specified by the designer. Designers can write these behaviors by demonstrating the stimuli on the interface mockup, and can later go back and edit these scripts manually if necessary. As the user demonstrates an action, the currently-selected behavior in the script updates itself by setting its stimulus to the action the user just performed. For example, if the user demonstrates a click on a particular layer, the currently-selected behavior's stimulus is set to clicking on that layer. The stimulus options for behaviors are: mouseover, mouseout, mousedown, mouseup, and click. Playbook does not do any inferencing or reasoning on the stimulus-response pairs the user writes.

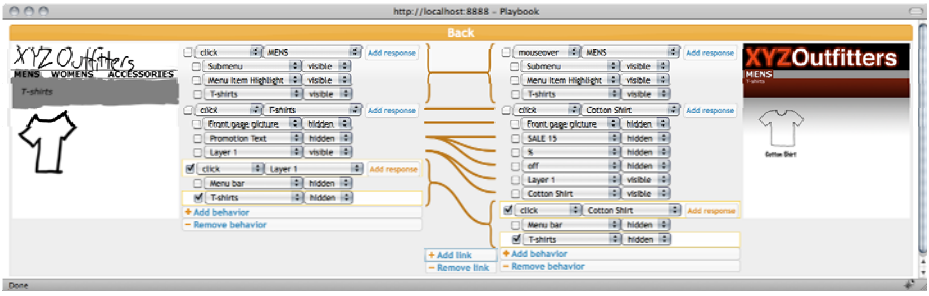

Fig. 1. The user is mapping stimuli and responses from the script on the left to the script on the right. Equivalent objects have a line between them. For example, a click on the "MENS" layer in the left mockup is equivalent to a mouseover of the "MENS" layer in the right mockup. When whole behaviors are equivalent, Playbook uses curly braces and a single line between the behaviors to reduce clutter. The mappings between mockup stimuli and responses are used to allow the user to interact with multiple interactive mockups simultaneously.

Every script is tied to a particular mockup (but may be used across versions of that mockup), because it uses layers that may only be in that mockup. However, we wanted to give the designer the ability to easily apply old scripts to new mockups. For example, if a designer writes a script for version 1 of a mockup, and makes some minor tweaks to the graphics between version 1 and version 2, we did not want the designer to have to rewrite the scripts that they wrote for version 1. Thus, after the user writes a script, Playbook automatically tries to apply the script to new versions of that mockup, and generates scripts for them. This is done by trying to replicate all of the behaviors from the previous script by looking for layers with the same name on the new version. If a behavior, or part of a behavior, refers to a layer name that is not in the new version, Playbook omits that part of the behavior. The designer must then verify these newly generated scripts before they can be used. As they are verifying the script, they can make any desired changes to its content.

One of the novel features of Playbook is the interface "compare" feature, which allows designers to compare interactive versions of their mockups by interacting with both simultaneously. We designed the compare feature to allow mockups with very different user interfaces to be compared. Allowing the designer to specify which actions are 'equivalent' on different interfaces enables this. When designing the compare feature, the first important design issue that needed to be addressed was: at what granularity should designers be able to specify equivalence: complete behaviors or individual stimuli and responses? We decided to allow designers to specify equivalencies of stimuli and responses because it increased the flexibility of what could be equivalent. Between interfaces, stimuli can be marked as equivalent to other stimuli, and responses can be marked as equivalent to other responses in a many-to-many relationship (any number of stimuli or behaviors can be marked as equivalent to any number of stimuli or behaviors in another mockup).

Another design issue was how and when designers would view and edit the equivalency relationships. We believe that the most natural way of visualizing equivalency relationships is by drawing lines between equivalent stimuli and responses, as shown in the center of Figure 1.

These equivalency relationships between actions can only be specified in scripts within a task. For example, the 'buy a t-shirt' script in mockup version 1 can only have equivalency mappings with the 'buy a t-shirt' script in mockup version 2; mappings cannot be made across tasks. When Playbook generates a script for a new mockup based on an old script, it automatically generates a set of equivalency relationships that can be verified, discarded, or augmented by the designer.

The equivalency connections are the basis for how Playbook allows the user to interact with multiple prototypes at the same time. When the user performs an action (stimulus) on one interactive mockup, Playbook then looks for the equivalent stimulus on any other mockups that are running. If there is an equivalent stimulus, Playbook simulates the stimulus on that prototype. If not, Playbook looks at the responses for the original stimulus. For every response, Playbook looks for equivalent responses in the other mockup, and simulates the stimulus responsible for that response. If the layer responsible for that stimulus on the equivalent prototype is not visible, Playbook still executes the stimulus, giving the designer a warning.

One of the benefits of Playbook being a web platform is that it enables multiple people to easily share the interactive prototypes. Playbook provides menu items to allow users to download a previous mockup as a Photoshop file, make changes to the mockup through Photoshop, and re-upload it to the Playbook server as a new version. Further, Playbook generates small HTML snippets that allow these interactive mockups to be easily embedded into other webpages. One could, for example, embed an interactive mockup into a wiki page that describes the interface and use the interactive mockup as a working example.

4 Implementation

Because it is a web-based interface, Playbook was implemented almost entirely in JavaScript. Using Photoshop's built-in JavaScript plugin capability, we added an "Upload to Playbook server" menu item to the Photoshop File Menu. On the Playbook server, the Photoshop file is processed, splitting the layers into separate image files. The Playbook server runs a copy of Photoshop and uses another Photoshop script to create the web version of the layer image so it can be used by the mockup.

The Playbook server stores all of the information it contains about each mockup, layer, script, etc. in a database. Playbook's web interface, in turn, periodically updates itself by querying the database. This allows users to stay updated if other team members are using the Playbook interface at the same time. The web interface also periodically saves any changes that are made to scripts back to the database, so that the users' scripts will still be on the server after they leave the page.

5 Conclusion and Future Work

This paper presented Playbook, a system that allows designers to maintain and compare multiple revisions of interactive prototypes. Playbook is a proof-of-concept that shows that it is feasible to enable revision control and comparison of interactive mockups while working with the tools that interface designers currently use. We

believe that many of the ideas behind Playbook can be used to augment future prototyping tools, including giving designers the ability to keep track of old designs and design alternatives, and allowing designers to compare prototypes. For future research, we plan to explore alternative ways to highlight differences between interactive mockups, expand the range of what can be prototyped using our scripting language without raising the floor of knowledge required, and explore ways to use Playbook as a "boundary object" to improve communication between designers and developers.

Finally, while Playbook is a system especially for interface designers, we also plan on exploring ways of applying some of the principles behind Playbook to development and prototyping systems for End User Development. To the extent that these systems permit separation of concerns between appearance and behaviors, augmenting them with some of Playbook's features may enable end users to better explore their design space and create more thoroughly designed artifacts

Acknowledgments. We thank our collaborators at IBM and the conference organizers and reviewers for their helpful advice. In addition, we thank the ARCS and Ford foundations for their generous support. This research was also partially supported by the National Science Foundation under grants IIS-0757511 and CCF-0811610.

References

1. Ozenc, F.K., Kim, M., Zimmerman, J., Oney, S., Myers, B.: How to support designers in getting hold of the immaterial material of software. In: ACM CHI Conf., pp. 2513–2522 (2010)
2. Grigoreanu, V., Fernandez, R., Inkpen, K., Robertson, G.: What designers want: Needs of interactive application designers. In: IEEE VL/HCC, pp. 139–146 (2009)
3. Newman, M.W., Landay, J.A.: Sitemaps, storyboards, and specifications: a sketch of Web site design practice. In: Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS), pp. 263–274 (2000)
4. Carter, A., Hundhausen, C.: How is User Interface Prototyping Really Done in Practice? A Survey of User Interface Designers. In: IEEE VL/HCC, pp. 207–211 (2010)
5. Leshed, G., Haber, E.M., Matthews, T., Lau, T.: CoScripter: automating & sharing how-to knowledge in the enterprise. In: ACM CHI Conf., pp. 1719–1728 (2008)
6. Lin, J., Thomsen, M., Landay, J.: A visual language for sketching large and complex interactive designs. In: ACM CHI Conf., pp. 307–314 (2002)
7. Klemmer, S.R., Newman, M.W., Farrell, R., Bilezikjian, M., Landay, J.A.: The designers' outpost: a tangible interface for collaborative web site. In: ACM Symposium on User Interface Software and Technology (UIST), pp. 1–10 (2001)
8. Hudson, S., John, B., Knudsen, K., Byrne, M.: A tool for creating predictive performance models from user interface demonstrations. In: ACM Symposium on User Interface Software and Technology (UIST), pp. 93–102 (1999)
9. Hartmann, B., Yu, L., Allison, A., Yang, Y., Klemmer, S.R.: Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In: ACM Symposium on User Interface Software and Technology (UIST), pp. 91–100 (2008)