FrameWire: A Tool for Automatically Extracting Interaction Logic from Paper Prototyping Tests

Yang Li [1*] **Xiang Cao** [2] **Katherine Everitt** [1] **Morgan Dixon** [1] **James A. Landay** [1]
[1] DUB Institute, University of Washington, Seattle, WA [2] Microsoft Research Cambridge, UK
{yangli, xiangcao}@acm.org, katherine.everitt@gmail.com, {mdixon, landay}@cs.washington.edu

ABSTRACT
Paper prototyping offers unique affordances for interface design. However, due to its spontaneous nature and the limitations of paper, it is difficult to distill and communicate a paper prototype design and its user test findings to a wide audience. To address these issues, we created FrameWire, a computer vision-based system that automatically extracts interaction flows from the video recording of paper prototype user tests. Based on the extracted logic, FrameWire offers two distinct benefits for designers: a structural view of the video recording that allows a designer or a stakeholder to easily distill and understand the design concept and user interaction behaviors, and automatic generation of interactive HTML-based prototypes that can be easily tested with a larger group of users as well as "walked through" by other stakeholders. The extraction is achieved by automatically aggregating video frame sequences into an interaction flow graph based on frame similarities and a designer-guided clustering process. The results of evaluating FrameWire with realistic paper prototyping tests show that our extraction approach is feasible and FrameWire is a promising tool for enhancing existing prototyping practice.

Author Keywords
Paper prototyping, programming by demonstration.

ACM Classification Keywords
H.5.2 [User Interfaces]: Prototyping; D.2.2 [Design Tools and Techniques]: User interfaces.

General Terms
Design, Human Factors.

INTRODUCTION
Paper prototyping is a widely used technique for the early stages of user interface design. It allows a designer to acquire early user feedback about a design. A designer can create an interface mockup using paper artifacts such as hand drawings and test an early-stage idea with a user in a visual and tangible way [18, 19]. During a test, a designer, playing the role of the "computer", presents an interface screen (e.g., drawn on a piece of paper) to a user according to the user's actions. The user interacts with the interface by pointing at elements they would use as if it were a real system, and the designer reacts by changing the paper screens that reflect the interface. By asking the user to think aloud about their actions and the interface design as the test proceeds, a designer can get qualitative feedback as well as see how users will use the design.

Although paper prototyping allows fluid and rapid iteration on a design, its spontaneous and physical nature makes it difficult to analyze and communicate a design and test findings to a wider audience. Designers often videotape a test to capture rich interaction scenarios and the conversation with a user [16]. In practice, seeing the users get confused and voice their concerns has more impact than simply hearing someone describe what happened [19].

However, it can be time-consuming to watch and edit the resulting video clips, and there are no tools for analyzing the tests and distilling interaction flows and scenarios of interest. This is why designers are quite willing to videotape a test but reluctant to review recorded video clips. For example, a designer might want to find out how much time is spent on each interface screen or quickly retrieve and replay what a user said when a particular screen was presented. Currently, a designer would have to manually scan through a video clip to locate a segment of

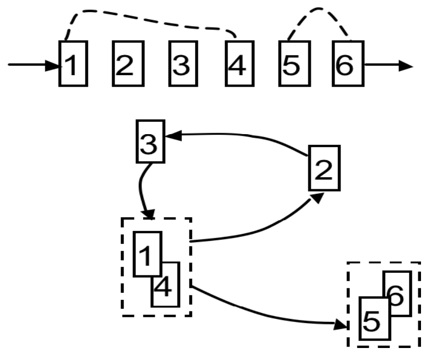


Figure 1. A sequence of video frames (top). If Frame 1 shows the same or a similar interface screen to Frame 4, we consider them repetitions of the same screen and "wire" them together. The wiring results in the bottom graph, which generalizes the sequence above and illustrates the interaction flow of a paper prototyping test. When multiple sequences are available, this process is likely to produce a more complete graph.

* Yang Li is now a Research Scientist at Google Research.

interest, or manually compute the time spent on each screen.

In addition, although a paper prototyping test allows a rich, situated communication between a designer and a user, its manual, face-to-face nature makes it difficult to scale to more extensive user testing. As a design idea matures, an interactive, more automated prototype is often needed so that a more extensive user test is affordable, user feedback can be more detailed, and testing in a more realistic situation can occur. The current process of transitioning from a paper prototype to an interactive prototype requires additional technical skills and effort on the part of the designer.

To address these two issues, we created FrameWire, a system that provides a high level, structural view over one or more paper prototype test video clips and generates interactive HTML-based prototypes from these clips. These features are realized using automatic extraction of interaction logic from raw video clips of multiple paper prototyping tests. The name of the system comes from our approach of "wiring" video frames together to reflect the interaction flow of the design that is demonstrated in these clips, which are sequences of frames (see Figure 1).

Research Challenges and Contributions

Much of the prior work in rapid prototyping support has focused on electronic prototyping tools [3, 7, 12, 17], which often require designers to either give up the familiar practice of paper prototyping or manually bridge the gap between paper and electronic prototyping themselves. Although there are many electronic tools intended to imitate the paper prototyping experience, the affordances of physical paper prototyping is still irreplaceable.

In our early exploration, seven interaction designers were asked to use an electronic tool we developed that closely replicated physical paper prototyping concepts. However, although our participants liked the advanced editing and analysis features of the tool, it was observed that they still preferred to start a design on paper. These observations motivated us to build a tool to enhance paper prototyping while preserving the existing practice as much as possible.

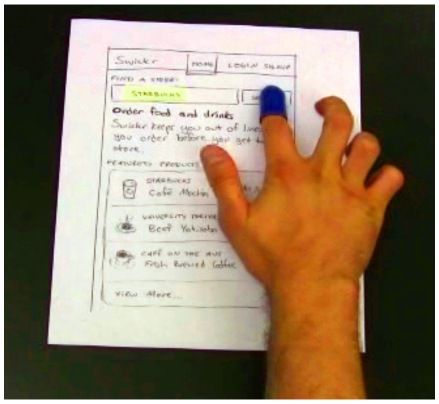


Figure 2. A user clicks on an interface component by tapping her finger on the paper drawing. The blue finger tip distinguishes the user's action finger from other objects in the scene.

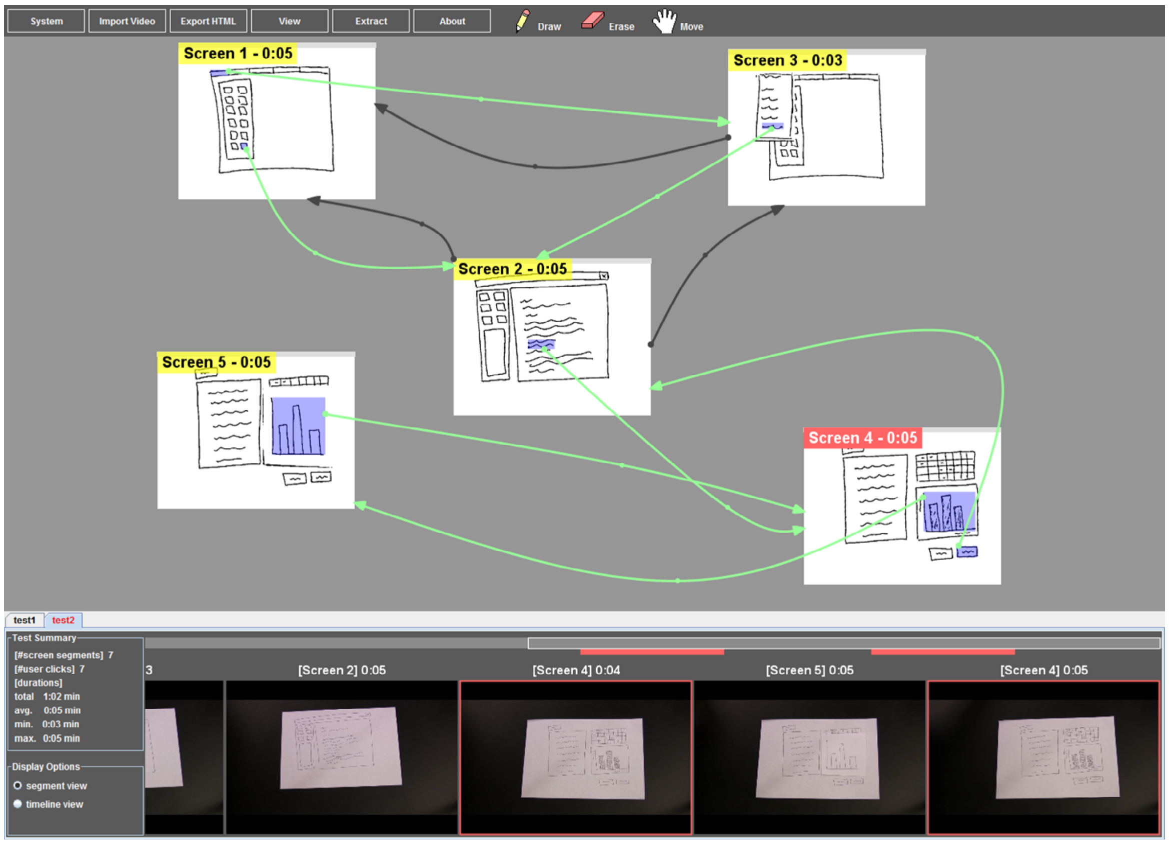
To this end, our goal was to provide a tool to help designers to analyze existing paper prototyping tests, to create a structural, easy-to-grasp view of the resulting video recording and to transform a paper design into an electronic, interactive form. To preserve the flexibility of paper prototyping practices, we also chose not to resort to additional sensors or special devices such as digital pens [2]. We rely only on a regular video camera that is already used in existing paper prototype tests. This goal poses the technical challenge of understanding interaction logic from video clips that merely consist of streams of raw image frames.

FrameWire contributes a novel approach for extracting interaction logic from video streams that combines computer vision processing and automatic clustering with human-driven interactive refinement. The extracted information includes interface screens, aggregated transitions, and their associations with the original video streams. Once a designer imports the video recording of one or more tests (see Figure 2), the interaction logic is automatically extracted and presented as an editable interaction flow graph (see Figure 3). FrameWire's interaction flow graphs consist of screens and transitions. A screen represents a state of the target interface. A transition specifies what state becomes current when an event occurs, e.g., on a button click.

With these extracted structures, FrameWire allows a designer to directly index, replay and annotate the video frames associated with each interface screen or transition, and understand how the user's time is spent during each session. This feature is also useful for others in addition to the designer herself to easily understand the design and observations from user tests. FrameWire works differently and addresses different issues than prior work such as d.Tools [7], which also provides rich features for analyzing test video. For example, both FrameWire and d.Tools allow a user to index video frames in a structured way. However, d.Tools correlates a test video stream to predefined interaction logic that a designer has to specify explicitly before a test, while FrameWire automatically derives the logic and its association with video frames from the video itself. This feature required solving additional technical challenges beyond the prior work.

FrameWire also contributes a new feature for turning a paper prototype into an interactive electronic prototype in the form of a set of linked HTML pages. Together, these two features allow a designer to analyze and present a set of raw video clips in an efficient, structured manner and to easily take a paper prototype design farther into extensive or remote user tests using the generated HTML pages.

In the remainder of the paper, we first describe FrameWire from an interface designer's point of view. We next describe how FrameWire extracts interaction logic from a set of video clips and generates interactive prototypes. We then report on the studies we conducted to understand the feasibility of our approach and how designers would react to FrameWire. This is followed by a discussion of the limi-



Test Summary
[#screen segments] 7
[#user clicks] 7
[durations]
total 1:02 min
avg. 0:05 min
min. 0:03 min
max. 0:05 min

Figure 3. FrameWire extracts an interaction flow graph from the video recording of paper prototype tests. The bottom pane shows all the occurrences of the extracted screens in the video clips. The interface also shows the statistics of a test, such as the total duration, the number of screen changes and user clicks, and the minimum and the maximum duration spent on a screen.

tations of FrameWire and plans for future work. We finish with a discussion of related work and our conclusions.

THE FRAMEWIRE SYSTEM

Here we describe how a designer would use FrameWire to analyze one or more tests of a paper prototype design to create a structural, communicative view of the test results, and to extract an interactive prototype from these tests.

Paper Prototyping

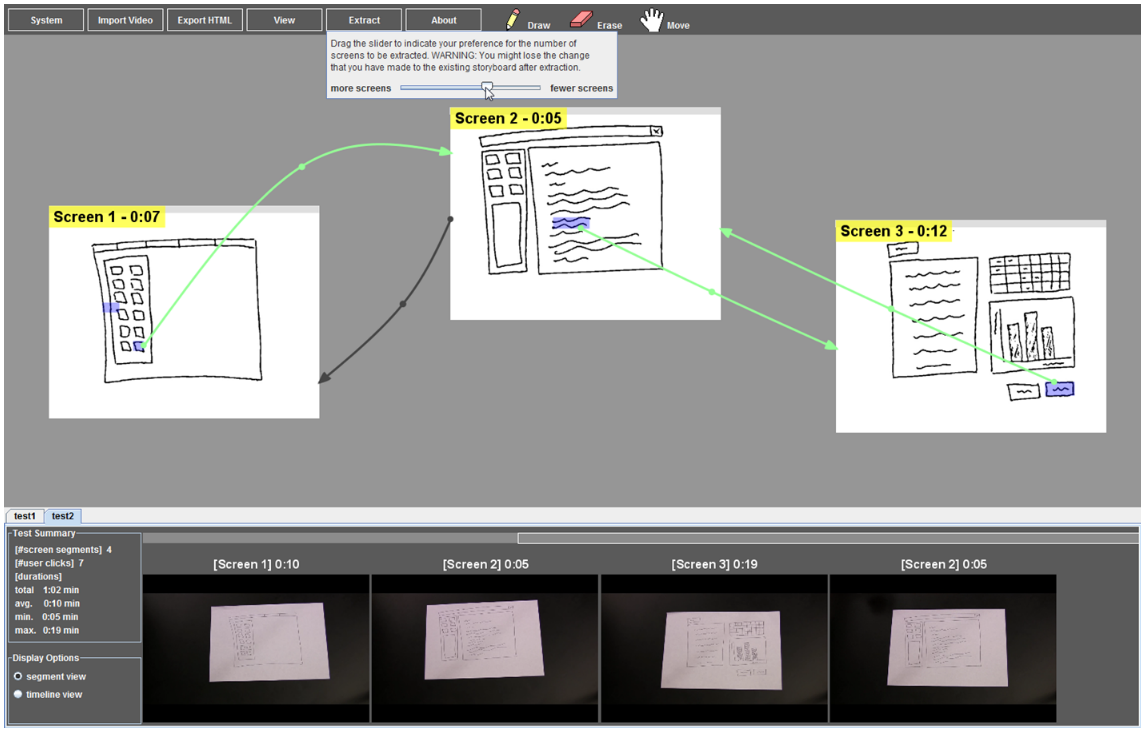
FrameWire does not require designers to modify their current paper prototyping practices. To use FrameWire, they create interface mockups with physical paper by hand drawing, cutting, copying and pasting. They test a design by presenting different paper sheets of interface components on a table in front of a user, in response to the user's actions. A single video camera mounted above the table captures the interface screens (i.e., paper sheets) presented, the user's and the designer's hand activities on the table, and what is said during the test.

To simplify the detection of the user's input on each screen with a single camera, FrameWire requires a user to wear an inexpensive, commercially available, blue rubber finger tip to indicate the cursor (see Figure 2), and to dwell for a short period of time for a click action. Currently, this is the only special requirement of FrameWire. The use of the blue fingertip avoids multi-finger confusion and is more robust than using skin detection alone. As we found in our user studies, this requirement did not bother the users.

Importing Video Clips for Analysis & Authoring

Once a test is videotaped, a designer can import the video clip into FrameWire for analysis. FrameWire can load video clips from multiple tests of a design, each of which may have followed different paths through the design. The tool automatically extracts interface screens, user clicks and the transitions between screens from the clips (see Figure 3).

In the main canvas, FrameWire shows the extracted logic as an interaction flow graph. Each screen is labeled by a screen name that corresponds to its occurrences in the video clips at the bottom, and a duration that shows the average amount of time spent on the screen during user tests. Each transition is represented by an arrow between screens. A transition that starts with a blue translucent patch over a screen element indicates an inferred interface component that a user has clicked on (e.g., a button or an icon). Because not every path of a design might have been covered during paper prototype tests, these functional transitions may only capture a subset of the transitions in a design. To address this issue, in addition to these functional transitions, FrameWire also proposes suggestive transitions (shown as dark arrows). FrameWire adds a suggestive transition based on the assumption that if there are transitions from screen A to B, there might be a need to have at least



Drag the slider to indicate your preference for the number of screens to be extracted. WARNING: You might lose the change that you have made to the existing storyboard after extraction.

more screens — fewer screens

Test Summary
[#screen segments] 4
[#user clicks] 7
[durations]
total 1:02 min
avg. 0:10 min
min. 0:05 min
max. 0:19 min

Display Options
○ segment view
● timeline view

Figure 4. A designer can adjust the Extract slider to have FrameWire generate more or fewer screens. This figure shows that some similar screens in Figure 3 are merged as the designer drags the slider towards "fewer screens".

one transition from screen B to A to allow a user to go back to the previous screen. A designer can choose to add a functional transition based on a suggestion (by drawing a line between screens) or simply ignore it. A designer can also decide to add functional transitions herself. At a high level, the graph is constructed based on combining similar frames as illustrated in Figure 1. We will elaborate on the extraction details in the Algorithms section.

Imported video clips are displayed in the bottom pane of the FrameWire interface in one of two forms. In the Segment View, FrameWire only shows video segments that contribute to extracted screens and transitions (see Figures 3 & 4). In the Timeline View, FrameWire shows the entire clip, including the frames in which a designer adds, removes and positions paper mockups, which do not contribute to the extraction. Each segment is represented by a thumbnail showing the first video frame of the segment.

The screen that each video segment contributes to as well as the time spent in that segment during the test is displayed above the segment. As a result, a designer can easily discover which screens have appeared in a test, the sequence of their occurrences, and the time spent on each occurrence.

A designer can right click on any video segment thumbnail to retrieve and replay it. Thus, a designer can easily find and view the video segment that is associated with a screen occurrence and watch what a user did and said.

Selecting a screen in the graph will also highlight all the occurrences (segments) of the screen in the video clips at the bottom, e.g., Screen 4 is selected in Figure 3. The red bars beneath the scrollbar indicate the temporal distribution of these occurrences within the entire clip. A designer can quickly locate a highlighted segment by clicking on a red bar. This is useful when the segment is not on screen. Similarly, selecting a video segment at the bottom will highlight the corresponding screen in the graph.

Refining & Communicating Extractions

If the automatic extraction does not fully capture the designer's intent, the designer can interactively refine and annotate the extracted logic. The value of the Extract slider (Figure 4, top) globally determines how similar two frames must be to be considered repetitions of the same screen, which influences how FrameWire should "wire" frames. A designer can also choose to refine a specific set of screens to merge or reveal minor changes such as popup menus.

Designers can move or resize the extracted blue patches to correct the inferred components that users can click on (in the interactive HTML prototype exported later) to trigger a transition. Designers can also add or delete transitions and screens to illustrate a desired flow.

Once a designer is satisfied with the interaction flow graph, she can save the graph and easily send the results to her colleagues who can view the interaction flow graph along with the video recording and make further edits.

Generating Interactive HTML Prototypes

In addition, a designer can export a graph as an interactive prototype. FrameWire parses an interaction flow graph and renders it into HTML pages (see Figure 5). A generated HTML prototype uses each screen as the background image of an HTML page and overlays clickable "hot areas"

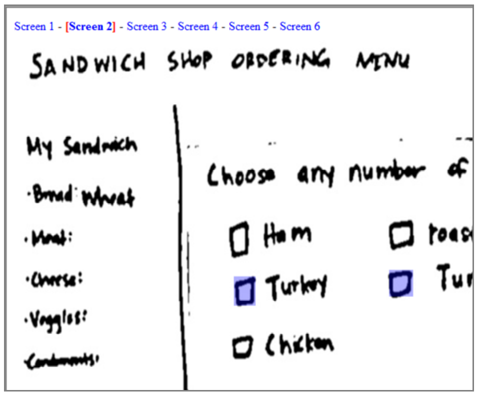


Figure 5. An HTML page generated from the video recording of a paper prototype test is loaded in a web browser.

of hyperlinks for the transitions, which correspond to the blue patches generated in FrameWire. The generated HTML pages can be tested in any web browser. As a result, with a small amount of editing, a designer can easily create an interactive prototype from recorded video clips.

ALGORITHMS

In this section, we describe how FrameWire extracts an interaction flow graph from a set of video clips.

Our Intuition

An interaction flow graph is computationally equivalent to a State-Transition Diagram (STD), where each screen in the graph is identical to a state in a STD and a click on an interface component represents the event that triggers the transition between states. If we know the STD for a design, we can generate valid interaction sequences. Note that there could be an infinite number of possible sequences when there are loops in a STD. However, the problem that we address here is the reverse. We intend to uncover an unknown STD from a finite set of known sequences.

Since a designer can reuse screens during a paper prototype test, there can be many similar frames in a set of imported video clips. If we consider those similar frames as repetitions of a single interface screen, we can then easily fold sequences of frames into a graph. The graph represents interaction flows manifested by these tests (see Figure 1). Although this approach does not allow us to recover all the transitions of a hypothetical graph (as some transitions may never happen during a test), we are able to extract a skeleton of a graph, which can then be completed by the designer. In the rest of the section, we discuss the process of extracting such a graph from video.

Preprocessing a Video Clip

The preprocessing stage performs several computer vision analyses to extract the interface screens and user clicks in one or more video clips, and generates a similarity matrix that describes the similarities between each pair of extracted frames. These intermediate results are used as input for the later stages of processing.

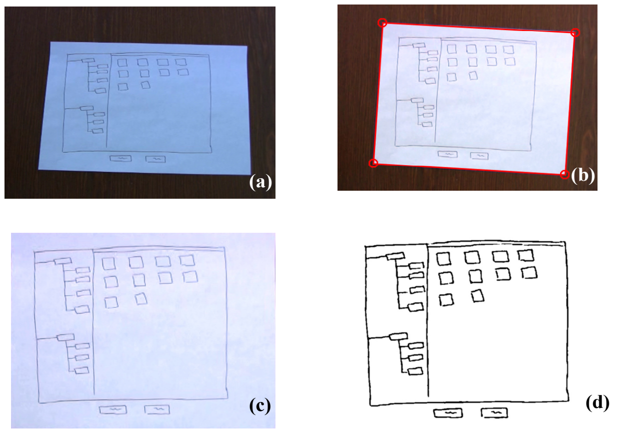


Figure 6. Screen frame extraction steps. (a) Original video frame. (b) After perspective/color correction & screen detection. (c) Extracted screen image. (d) Generated line drawing.

Initialization. Having loaded a clip, the system first semi-automatically learns the setting under which the clip was taken. This involves two steps. First, FrameWire records the background scene, i.e., an image without any paper sheets or hands. This is by default the first frame of the clip, or it can be chosen by the designer. Second, FrameWire calibrates the camera settings using a frame chosen by a designer that contains an interface screen sheet but no hands. After the designer specifies the size of the screen sheet for calibration (e.g., A4), the system finds the optimal perspective and color transform that makes the sheet appear upright, of the desired size, and white-balanced. This corrects the perspective and color distortion caused by the camera angle and environmental lighting for any planar objects placed on the table.

Extracting Interface Screen Frames. The system then analyzes each frame in a clip in order (see Figure 6a). After perspective and color correction, a background subtraction step extracts the foreground scene. If the foreground region is large enough, the system searches for a rectangular shape with a feasible size for a screen sheet by examining the geometric layouts of corner features detected in the foreground. If such a rectangular shape is found, it concludes there is an interface screen in the frame (Figure 6b). The system extracts the detected screen by cropping and rotating the rectangle to an upright orientation (Figure 6c). Finally, we generate a line drawing of the elements on the screen using an adaptive thresholding algorithm on the extracted screen image (Figure 6d).

Since we only need one representative frame for each time a screen is presented in the clip, the system ignores adjacent frames[1] that contain very similar screens. This significantly reduces the size of the similarity matrix

[1] Notice that the preprocessing evaluates the similarity between frames in a local, greedy way. As a result, it only considers two frames the same when they are extremely similar. A designer-driven, high-level clustering is followed after the preprocessing.

needed for the next stage, which makes the computation of the latter high-level extraction tractable. As a result, each extracted screen frame corresponds to a range of video frames and represents an occurrence of an interface screen. The system ignores frames with hands inside the screen boundary (detected by skin color), since they correspond to either the designer updating the screens, or the user clicking on the screen (detected separately as below).

Detecting User Clicks. If the current frame contains an interface screen, the system also attempts to detect the user's blue finger tip by searching for a blue-colored region within the screen boundary. If the finger tip is present and has been static for a period of time, e.g., 0.2 seconds by default, a click is recorded, and the finger tip position is translated into a position in the current extracted screen.

Calculating Screen Frame Similarities. After all of the screens are extracted, the system generates a similarity matrix across all of the imported clips. To calculate the similarity between two screen frames, the system compares the line drawings that it has generated for both, which have been geometrically aligned and are insensitive to lighting conditions. To compensate for slight misalignment caused during extraction, both line drawings are first blurred using a Gaussian filter. The similarity metric (ranging between 0 and 1) is then defined as the percentage of corresponding pixels that have similar values in both images.

Many of the analysis parameters, such as the screen sheet size and the finger dwelling period, can be adjusted by the designer to suit the specific environment of the user test. Note that the perspective transformation is calculated once for an entire video clip while the rotation of paper sheets is continuously computed for each frame. If the camera happens to be moved significantly during a test session, a recalibration from that point might be needed.

Extracting Unique Interface Screens

Based on the similarity matrix of screen frames generated by the preprocessing, FrameWire clusters all the frames

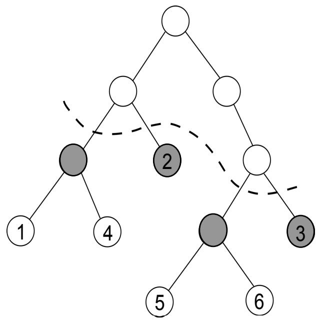


Figure 7. An illustration of a possible cluster hierarchy for the example in Figure 1. Frames are clustered hierarchically based on their similarity. Leaf nodes (numbered with frame indices) represent frames. The nodes below the dashed line have an average similarity larger than the threshold specified by the designer. Here four unique screens are extracted based on four groups of frames (represented by the gray nodes).

into groups, and the frames in each group are considered repetitions of the same screen. Hereafter, we use "frame" for "screen frame" for brevity. Notice that frames are a subset of the original raw video frames selected for clustering.

An important question here is to answer how similar a set of frames must be for them to be considered repetitions of the same screen. It is challenging to determine an appropriate threshold in a completely automatic way, because an optimal threshold may be design specific. For example, when the only difference between two frames is whether a checkbox is selected or not, a designer can consider them either as the same screen or different depending on the granularity of the analysis she wants. As a result, our design principle in FrameWire is to first automatically identify a potentially optimal threshold for generating an initial extraction. We then allow designers the option to choose an appropriate threshold for their design by adjusting the Extract slider shown in Figure 4.

We employ hierarchical clustering [20] to group frames into a tree structure (see Figure 7). In each step of a hierarchical clustering process, the most similar frames are clustered. The leaf nodes are frames that can come from one or more video clips. Each internal node represents a group of frames. Each node has an average similarity S ($0 \leq S \leq 1$) that indicates how similar the frames in the group are, where $S = 1$ for a leaf node. As the designer drags the Extract slider, FrameWire locates a level in the cluster tree (see the dashed line in Figure 7) in a top-down manner based on whether the S of a node is larger than the threshold specified by the designer.

We currently identify an initial threshold using a simple heuristic. We search through the space of all possible thresholds given a cluster hierarchy, and select a threshold that will cause the most significant increase in S averaged across all the candidate groups, e.g., the four gray nodes in Figure 7. The intuition behind this heuristic is to keep the S of each group as large as possible, i.e., the frames in each group should be as similar as possible, while at the same time allowing as few groups (unique screens shown in an interaction flow graph) as possible.

The cluster hierarchy, an internal structure, maintains the association between the extracted graph and frames. It enables designers to easily select a set of similar frames. If a designer clicks on a frame that is already selected, the most similar frames will be selected as well and each additional click will move the selected group node up one level in the hierarchy. For example, in Figure 7, if Frame 1 is selected, clicking on Frame 1 will also select Frame 4.

Extracting Transitions

Once all the unique screens are identified, FrameWire extracts the transitions based on the temporal adjacency of their occurrences in the video clips. For example, if Screen A appears right before Screen B in the video, we assume there should be a transition from A to B, although there is no guarantee that a user click will be detected on Screen A. A transition might, for example, be triggered by a timer.

Although the preprocessing identifies user click positions, it does not analyze the higher level semantics of what interface components a user has clicked on. The motivation for identifying these interface components is for generating interactive prototypes. For example, in a generated HTML prototype, a user should be able to trigger a transition by clicking on any part of a target component instead of the specific click point acquired by tracking the blue finger tip.

In general, it is challenging to recognize a hand-drawn interface without human intervention [1, 12]. Instead of parsing the entire screen to recognize all the interface components, we conduct parsing in a lazy, local processing manner. FrameWire first clusters user clicks that are close to each other and detected in the occurrences of the same screen. FrameWire then treats each cluster as a seed region and incrementally expands the region to search for a surrounding interface component. A blue patch is then generated to cover the inferred component, indicating a clickable area on the screen. As this greedy search can be error-prone, FrameWire allows a designer to easily modify the parsing result by adding, deleting, moving or resizing a blue patch to indicate a desired clickable region.

EVALUATIONS

To understand whether our interaction logic extraction algorithms were feasible and how designers would react to such a tool, we evaluated FrameWire in three stages, in settings ranging from idealistic to realistic.

Initial Trials

Before analyzing realistic paper prototype tests, we tested the performance of FrameWire on three mockup sessions of 1-2 minutes each recorded in our research lab, using simple paper prototypes we created. FrameWire achieved 100% accuracy on detecting all interface screen frames and user clicks. The clustering algorithm merged frames that had the same screen with 100% accuracy. The ink parsing algorithm was able to correctly infer an active interface component (i.e., the blue patch) for 76% of the user clicks. This encouraged us to examine how our system performed with more complex and realistic paper prototypes.

Feasibility Tests with More Realistic Data

We collected three paper prototypes developed by undergraduate students in a user interface design class. The prototypes were a mobile task sharing system, a workout tracker, and a mobile ordering system. We then conducted and videotaped the tests of these paper prototype designs with a user who was not familiar with the project and had never participated in a paper prototyping test before. After a brief background introduction to each prototype, we asked him to perform a given task by clicking on the paper interfaces. We instructed him to wear the blue finger tip and avoid reaching onto the paper until the screen had changed. He had no difficulty following these instructions or with the presence of a video camera. The video camera was set on a tripod pointed down, and automatic exposure adjustment was turned off to maintain consistent brightness throughout the video clips. We used FrameWire to analyze the results from these three sessions. The sessions ranged from 1-2 minutes long and included 9-11 interface screens.

Results. Here we focus on reporting the objective performance of FrameWires's extraction algorithms and discuss the feasibility of extracting interaction logic from video.

For the preprocessing stage, FrameWire took 187 seconds on average to preprocess every minute of video on a 2GHz Centrino Core Duo computer. This was slower than real time and there is much room for algorithm optimization. Nonetheless, the preprocessing only needs to be invoked once, i.e., when a video clip is imported into FrameWire.

Table 1 summarizes the preprocessing results for each session. The ground truth (actual #) was obtained from a manual analysis of the video clips. In detecting screen frames and user clicks, there may be four types of errors:

- *Duplicate*: One event instance is detected as multiple identical or similar instances. This can occur when the paper sheet is somehow crinkled, or when a user dwells too long for a click.
- *False-alert:* An event is detected when it is not actually present, such as when the user pauses her finger but does not intend a click.
- *Wrong:* The event is correctly identified but the extracted information (e.g., click position) is erroneous.
- *Miss:* The system fails to identify the event, such as when the screen sheet is outside the camera view, or the user dwells too short for the click.

Session	Task sharing system		Workout tracker		Ordering system	
Event	screen	click	screen	click	screen	click
actual #	**11**	**11**	**9**	**9**	**11**	**10**
correct	10	9	9	8	11	8
duplicate	6	3	3	3	5	4
false-alert	0	3	0	0	0	0
wrong	0	1	0	0	0	0
miss	1	1	0	1	0	2

Table 1: Preprocessing results on user test sessions.

The preprocessing achieved a high rate of correct events on both screen frame detection (97%) and user click detection (83%), and few *false-alert, wrong*, and *miss* events occurred. The screen frame extraction was able to distinguish screens differentiated only by ticking a checkbox. Some of the reasons that caused the slightly lower rates than the mockup sessions may include: user hesitation when selecting interface components, more complex interface screens, and suboptimal lighting conditions in the test room.

We would also like to note that the preprocessing stage detects events individually and does not reason about their interrelationship. However, in the later stages, many of the detection errors can be easily eliminated by considering the higher-level structure of the prototype and user behavior. For example, the frequently detected duplicate screen frames can be naturally grouped into the same screen by the frame clustering algorithm. Duplicate clicks can provide additional cues in extracting clickable interface components on the screen (i.e., blue patches). Missing clicks can be compensated for by adding suggestive transitions between screens and the designer can specify the activation area later. Finally, combining the results from multiple test sessions of the same prototype will also likely result in more complete coverage of the screens and transitions.

Based on the results of the preprocessing, FrameWire performs three major computations for high-level processing: hierarchical clustering, transition tracing and click clustering. These computations are performed in real time. The initial similarity threshold, which FrameWire automatically determined based on the cluster hierarchy, correctly merged 80% of the frames that showed the same screen in these tests. The Extract slider can be used to achieve a desired granularity with more or fewer screens.

Implications of the Performance Results. These results showed that although there is still room for improving extraction accuracy, it is feasible to extract interaction logic from the video recordings of paper prototype tests with FrameWire. The performance results indicated that the vast majority of interface screens and transitions could be extracted correctly by FrameWire. This was further confirmed by our participants in the qualitative study described in the following section. In addition, FrameWire gives a designer the flexibility to control the automatic analysis process and to refine the inference results. With these positive performance results, we sought to collect qualitative feedback on FrameWire from interface designers who tried it in more realistic user sessions.

Acquiring Qualitative Feedback in Realistic Sessions

We asked four interaction designers to use FrameWire in their own paper prototyping projects. The purpose of the study was to solicit qualitative feedback from them on FrameWire's design and the new experience that it enables.

Each trial involved one realistic paper prototype created by the designer, who tested the design with three different users in separate sessions. All sessions were video-recorded, resulting in twelve video clips, each 2-3 minutes in length. The twelve user participants were recruited from a university campus for these tests. The four designs that were tested in these experiments included a touch screen alarm clock, a control panel to operate a lecture room, a video chat tool, and a restaurant ordering system.

After the designer finished the three user test sessions, the video recordings of the tests were imported into FrameWire. Out of the twelve sessions, eight were successfully processed by the preprocessor. The other four were affected by non-preferred camera/environment settings, or the designer incidentally violated some of the system requirements such as rearranging the table background. This showed that our preprocessing algorithm is yet to be perfected to adapt to unexpected situations that may exist in real life scenarios. Nonetheless, these provided enough data for each designer to proceed with FrameWire.

Designers were asked to use FrameWire to analyze their tests, edit the results generated by automatic analysis if necessary, and export the design as HTML prototypes. After each trial, we asked the designer to comment on what they liked, and disliked about FrameWire and how they would imagine using FrameWire in their design practice.

What our participants liked the most about FrameWire was the extraction of interaction flow graphs, test statistics and generating HTML pages. As one designer commented:

> *It's a neat way to make the paper prototype interactive on a computer. It seemed like a nice way to be able to view everything at a high level…This seems useful for running a user study a few times live, then letting many more people use it through the web.*

Our participants' least favorite issues with FrameWire included usability issues, such as "*The controls were a little confusing. It was hard to tell what the Hand tool was exactly pointing at*", and the lack of functionality that they needed. The designers requested a way to quickly merge redundant arrows and the ability to zoom into the video. These issues will be easy to address in a final system.

In terms of the Extract slider, one designer complained that it was difficult to find an ideal number of screens by just dragging the slider. We also observed that another participant started his search for a desired graph by dragging the slider all the way to the fewest screens, and then slowly incremented until it was close to what he wanted. This inspired the idea that it might be a useful feature to animate through the different interaction flow graphs that the system could possibly extract.

Our participants also hoped to use FrameWire in their design practice, as one participant commented:

> *I would use FrameWire after a few paper iterations, when I had a better idea about the design in general and needed a lot of feedback from a wider audience or to be able to present the prototype to developers to explain the concepts.*

However, although all our participants appreciated being able to refine extracted contents such as adjusting a click area, they did wish the automatic extraction could be more accurate and capture more detail about their designs. They also wished FrameWire was able to extract multi-finger interactions, a recent trend in user interfaces. This raises a new research question for future work.

DISCUSSION & FUTURE WORK

FrameWire brings easy-to-create paper prototypes quickly into a digital, shareable form and maintains their associations with original video recordings. Here we discuss its limitation and potential follow-up research.

Limitations

The inherent limitation of our approach is that it is not able to infer transitions that did not occur during the user tests. Although FrameWire uses suggestive transitions to address this issue, more sophisticated inference might be necessary to further complete the interaction flow graph. This may require the semantic analysis of sketches [1, 12], which is not our focus with FrameWire. Currently, FrameWire approaches this issue by leveraging human intelligence. By presenting the extracted logic in a human-friendly manner, a designer can easily adjust automatic extraction, analyze test videos, and modify or complete the extracted logic.

Another limitation of FrameWire is that it currently only detects click events. Nevertheless, user clicks often contribute a major portion of user actions in a paper prototyping test, especially for web or desktop interface design. In addition, FrameWire does not constrain a design or a user test in any way. A user can perform any interactions that are meaningful to a design such as gesturing or dragging. FrameWire still captures these scenarios in the video clips but does not attempt to interpret them. We observed that, although a user may perform various interactions on an interface, most transition-triggering events are still clicks. As a result, clicks are reliable delimiters in an event stream. By detecting click events, FrameWire can effectively extract the interaction flow of a user test, and meanwhile group unrecognized interactions (e.g., dragging) into click-delimited segments that can be indexed and viewed. This establishes a solid baseline to which the extraction of other interaction events can be added in the future.

Using Additional Sensors

Anoto pens [2] have been used in many systems aimed at integrating physical and digital workspaces [6, 13, 22]. This avoids difficult tasks such as low-level image processing. However, using Anoto alone has several serious limitations in our case. First, Anoto requires non-trivial overhead for designers to make a design and prepare a test. All the pages of a design need to be either printed using Anoto software or drawn with an Anoto pen beforehand. If a designer constantly iterates on paper by erasing and redrawing things (a common practice), the changes have to be tracked and updated with Anoto, which loses the fluidity of paper prototyping. Secondly, an Anoto pen alone does not provide all of the input needed for our system. The Anoto pen detects a page only when it touches or hovers on the page. All other contextual information is lost such as when a page is presented or removed. This context is essential for segmenting and structuring the video stream. In addition, pages that are presented, but not touched by the user will not be detected by Anoto. These are commonplace in paper prototype tests, such as transition screens, or Post-It notes attached to the main page to update part of the screen (e.g., drop-down menus or checkboxes). In contrast, a video camera is able to continuously capture this additional information during a test, and nicely integrate into the existing practice of recording user tests. In the future, it would be valuable to consider *adding* other sensors, such as an Anoto pen, to the camera to further improve the quality of extraction.

Robustness & Test Complexity

A major challenge for most vision-based systems is that the parameter setting of the system tends to be ad hoc to different environmental factors, e.g., lighting and background interference, and is sometimes subject to manual adjustment. FrameWire is not immune from these issues. However, the process for setting vision parameters has been carefully streamlined in FrameWire and requires relatively low effort by the designer. These parameters can be set up once and used repeatedly for each test setting.

FrameWire was designed to be robust to several common issues that may occur during user tests. It is not sensitive to movement of the paper sheet as it continuously detects and updates the sheet location in all video frames. With the background subtraction of the preprocessing, clutter on the desk also does not pose a problem, as long as the irrelevant objects remain static. By considering finger orientation, click position can be accurately determined regardless of which side the user is sitting on. In terms of duplicate clicks, FrameWire only considers the last click in each screen occurrence to trigger a transition. This improves the reliability of the transition extraction.

The paper prototype tests that we studied so far are short in length (1-3 minutes). However, a longer test does not necessarily imply a harder task for FrameWire. Short tests tend to have few repetitions, if any, which results in sparse data. Thus, some interactions reflected in a design do not occur or only occur once. This makes interaction logic extraction difficult. In fact, the longer a test is, the more data is available for FrameWire to use.

RELATED WORK

The contributions of FrameWire lie at the intersection of several research areas.

Integrating the Physical & Digital Worlds

There has been a large body of work in bridging the physical and digital worlds [6, 9, 11, 13, 21]. For example, Kim et al. developed a system to track paper documents on the desk and maintain their linkage with electronic documents [10]. However, their SIFT-based similarity measure cannot reliably handle documents with too little texture, such as the line drawings used in paper prototyping. In addition, none of the prior systems address the issue of extracting interaction logic from video recordings of paper prototyping tests, especially when no additional sensors (such as a digital pen) are employed.

Our studies with FrameWire show it contributes a feasible approach to addressing this problem that can bring new knowledge to designers during early stage prototyping.

Early Stage Prototyping Tools

A number of tools have been developed for the early stages of user interface design for various domains [3, 7, 11, 12, 15, 17]. The work most closely related to FrameWire is the Designers' Outpost [11], which allows designers to create a UI design using physical paper in a vision-based system. However, like other prior work, the Designers' Outpost requires a designer to explicitly specify the interaction flow of a design beforehand. This is different from the prototyping experience that FrameWire supports. Traditional paper prototyping has the merit of allowing a designer to start a design process by thinking about interaction scenarios as opposed to interaction logic, and revise the designs on the fly during tests in response to the users' behaviors. FrameWire preserves the practices and advantages of traditional paper prototyping and meanwhile brings a paper prototype forward into an editable and interactive electronic form.

Programming by Demonstration

FrameWire generalizes interaction scenarios captured in video clips and generates interactive prototypes. It essentially addresses a Programming By Demonstration (PBD) problem [5, 14]. In contrast to prior PBD work, FrameWire works with a specific kind of input and output: a set of video clips of paper prototyping tests and an interaction flow graph. A long-standing problem in PBD is how to effectively present inference results to users so that they can correct or confirm these results. FrameWire exemplifies one solution for this problem by allowing a designer to interactively refine an inferred interaction flow graph and to easily retrieve the examples that contribute to an inferred component. Video Pupperty [4] allows a user to create sequential animation by demonstrating paper drawing in front of a camera. In contrast, FrameWire extracts a full interaction flow graph instead of merely sequential steps.

Interaction Analysis Tools

Previously, interaction analysis tools were designed for understanding the usability and the usage of an interface that is either a built interface (e.g., [8]) or an interactive electronic prototype (e.g., [7]). In contrast, FrameWire supports a designer in analyzing the video recordings of paper prototypes that include no semantics or interaction logic other than sequences of frames. This assists designers in conducting a deeper analysis of user behaviors previously only possible with electronic, interactive prototypes.

CONCLUSIONS

We presented FrameWire, a tool that extracts a meaningful interaction flow graph from raw video clips of paper prototyping tests. FrameWire does not limit what can be drawn or presented on paper. It contributes a novel method for understanding paper prototyping test video clips. This innovation makes it possible to 1) allow a designer to easily analyze and communicate test video by viewing generated statistics and by directly indexing video segments related to each interaction behavior, and 2) transform a paper prototype into an interactive HTML-based prototype in a semi-automatic way.

REFERENCES

1. Alvarado, C. and Davis, R. SketchREAD: a multi-domain sketch recognition engine. *UIST'04*. p. 23-32.
2. Anoto. http://www.anoto.com.
3. Bailey, B.P., et al. DEMAIS: designing multimedia applications with interactive storyboards. *ACM Multimedia'01*. p. 241-250.
4. Barnes, C., et al. Video puppetry: a performative interface for cutout animation. *ACM Trans. Graph.* 27, 5 (Dec. 2008), 1-9.
5. Cypher, A., ed. *Watch What I Do: Programming by Demonstration*. 1993, MIT Press.
6. Guimbretière, F. Paper Augmented Digital Documents. *UIST'03*. p. 51-60.
7. Hartmann, B., et al. Reflective physical prototyping through integrated design, test, and analysis. *UIST'06*. p. 299-308.
8. Hong, J.I. and Landay, J.A. WebQuilt: a framework for capturing and visualizing the web experience. *WWW'01*. p. 717-724.
9. Johnson, W., et al. Bridging the paper and electronic worlds: the paper user interface. *CHI'93*. p. 507-512.
10. Kim, J., et al. Video-based document tracking: unifying your physical and electronic desktops. *UIST'04*. p. 99-107.
11. Klemmer, S.R., et al. Integrating Physical and Digital Interactions on Walls for Fluid Design Collaboration. *Human-Computer Interaction*, **23**: p. 138-213.
12. Landay, J.A. and Myers, B.A. Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer*, 2001. **34**(3): p. 56-64.
13. Liao, C., et al. Papiercraft: A gesture-based command system for interactive paper. *TOCHI*, 2008. **14**(4): p. 1073-0516.
14. Lieberman, H., ed. *Your Wish Is My Command: Programming by Example*. 2001, Morgan Kaufmann.
15. MacIntyre, B., et al. DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences. *UIST'04*. p. 197-206.
16. Mackay, W. E. and Fayard, A. L. Video brainstorming and prototyping: techniques for participatory design. In *CHI '99 Extended Abstracts*. CHI '99. p. 118-119.
17. Newman, M.W., et al. DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Computer Interaction*, 2003. **18**(3): p. 259-324.
18. Rettig, M. Prototyping for Tiny Fingers. *Communications of the ACM*, 1994. **37**(4): p. 21-27.
19. Snyder, C. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. 2003, Morgan Kaufmann.
20. Ward, J.H. Hierarchical Grouping to Optimize an Objective Function. *J. Am. Statist. Assoc.*, 1963. **58**: p. 236-244.
21. Wellner, P. Interacting with paper on the DigitalDesk. *Communications of the ACM*, 1993. **36**(7): p. 87-96.
22. Yeh, R., et al. ButterflyNet: a mobile capture and access system for field biology research. *CHI'06*. p. 571-580.