Expresso: Building Responsive Interfaces with Keyframes

Rebecca Krosnick[1], Sang Won Lee[1], Walter S. Lasecki[1,2], Steve Oney[2,1]
[1]*Computer Science & Engineering*, [2]*School of Information*
University of Michigan | Ann Arbor, MI USA
{rkros,snaglee,wlasecki,soney}@umich.edu

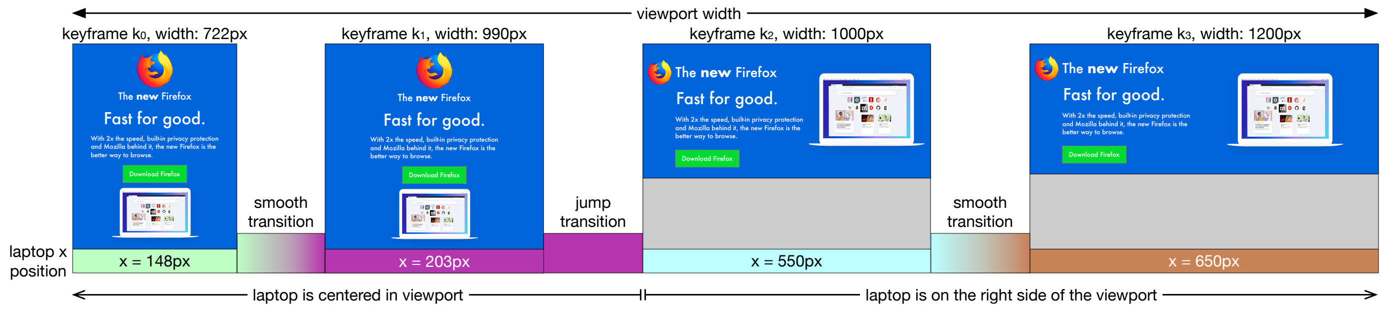
viewport width
keyframe k₀, width: 722px
keyframe k₁, width: 990px
keyframe k₂, width: 1000px
keyframe k₃, width: 1200px
The new Firefox
Fast for good.
Download Firefox
smooth transition
jump transition
smooth transition
laptop x position
x = 148px
x = 203px
x = 550px
x = 650px
laptop is centered in viewport
laptop is on the right side of the viewport

Fig. 1: *Expresso* allows users to create responsive interfaces by defining keyframes (which specify how the interface should look for a particular viewport size) and specifying how element properties should transition between keyframes. In this illustration, there are four keyframes (from left to right: k_0, k_1, k_2, and k_3). k_0 and k_1 specify that the page layout should be stacked vertically and centered for narrow viewports, such as mobile phones. k_2 and k_3 specify a two-column layout for wider viewports, such as full desktop browsers. This illustration highlights the x-position for the laptop image for all four keyframes and how it transitions between keyframes. In the resulting UI, the laptop is centered for viewport widths $<$ 1000 pixels and is on the right side of the page for widths $\geq$ 1000 pixels.

***Abstract*—Web developers use responsive web design to create user interfaces that can adapt to many form factors. To define responsive pages, developers must use Cascading Style Sheets (CSS) or libraries and tools built on top of it. CSS provides high customizability, but requires significant experience. As a result, non-programmers and novice programmers generally lack a means of easily building custom responsive web pages. In this paper, we present a new approach that allows users to create custom responsive user interfaces without writing program code. We demonstrate the feasibility and effectiveness of the approach through a new system we built, named Expresso. With Expresso, users define "keyframes" — examples of how their UI should look for particular viewport sizes — by simply directly manipulating elements in a WYSIWYG editor. Expresso uses these keyframes to infer rules about the responsive behavior of elements, and automatically renders the appropriate CSS for a given viewport size. To allow users to create the desired appearance of their page at all viewport sizes, Expresso lets users define either a "smooth" or "jump" transition between adjacent keyframes. We conduct a user study and show that participants are able to effectively use Expresso to build realistic responsive interfaces.**

***Index Terms*—responsive layouts, web programming, CSS**

I. Introduction

Web User Interfaces (UIs) often need to work across a variety of form factors and viewport sizes: from small mobile devices to large high-resolution displays. Web developers use *responsive design* to build websites that can adapt to any viewport size and window configuration. Cascading Style Sheets (CSS)—a language for specifying web pages' appearance—supports responsive design through "media queries" (`@media`), which specify style rules that apply for particular form factors.

CSS is an expressive language but is complex to use, especially in the context of creating responsive designs. This is because: 1) various rules need to be applied to each of multiple elements in the Document Object Model (DOM) hierarchy to achieve a particular visual appearance, 2) developers must be able to envision how new rules interact with existing rules, including from third party libraries, and elements in the context of the given HyperText Markup Language (HTML) hierarchy, and 3) developers must understand how rules affect page appearance across different page sizes and states [1], [2].

In this paper, we present Expresso, a tool that allows users to create custom responsive UIs. Expresso introduces the idea of using *keyframes* to define responsive layouts. Keyframes have had a long and successful history in computer-aided animation [3], where they greatly reduce animators' workload by allowing a computer to generate smooth transitions between drawings. With Expresso, users define keyframes that specify how a UI should look for a particular viewport size. Expresso then generates a responsive UI that satisfies the layout of every keyframe and infers the layouts for viewport sizes between keyframes. Expresso also gives users control over how their UI should transition between keyframes. In this paper, we contribute the following:

- The idea of defining responsive UI behavior by specifying the UI appearance at specific viewport sizes (keyframes), interpolating (smooth) transitions between them, and sup-

porting discontinuous (jump) transitions between significantly different layout states.
- An instantiation of our idea in Expresso, a system that allows users to encode requirements for a responsive web UI through direct manipulation.
- Evidence from a user study that participants without relevant programming background were able to effectively specify responsive web UIs with Expresso.

II. Related Work

A. Terminology

In our discussion of related work, we differentiate between three types of UI layouts. A *fluid* UI is one where elements' dimensions are proportional to the dimensions of the viewport. An *adaptive* UI is one where the programmer creates a different layout per form factor (e.g., as different HTML files), and the platform determines which layout to serve (e.g., the mobile layout or the tablet layout). A *responsive* UI — the focus of this paper — is one where the programmer creates one UI (e.g., as one HTML file) and specifies rules for how its layout should respond to different viewport sizes. In responsive UIs, individual elements' visibility, size, and position often will change for different viewport sizes. Adaptive UIs generally support less fine-grained control through all viewport sizes.

B. Constraint-Based Layouts

Constraints have long been used in UI specification [4]. Constraints allow developers to specify relationships between elements' visual properties that are maintained automatically. Many UI builders, including XCode [5] and Android Studio [6], allow developers to specify a limited set of layout constraints. Specifically, they allow users to define constraints through visual constraint metaphors, like "springs and struts" that expand and contract with the viewport. These models allow developers to define *fluid* layouts, where elements reside based on the viewport size. However, this model is not appropriate for *responsive* layouts because the constraints that they enable are not expressive enough to support rearranging, moving, or toggling element visibility for different viewport sizes. Although previous research has proposed constraints that could vary by UI and viewport state [7], [8], it is still difficult to author these constraints for responsive UIs. Expresso instead infers constraints for responsive UIs from keyframes.

C. WYSIWYG Interface Builders

"What You See Is What You Get" (WYSIWYG) interface builders allow users to specify a UI layout visually. Interface builders were first proposed in academic research [4], [9] and have achieved widespread commercial usage. Many modern web UI programming tools integrate interface builders including Dreamweaver [10], Webflow [11], and Bootstrap Studio [12]. Each of these tools provide live, editable previews of websites as developers write HTML and CSS. They also provide widgets to view and edit CSS properties. However, none of these tools allow responsive behaviors to be specified through direct manipulation. Although they lower the floor for developers, creating responsive UIs in these tools still requires conceptual CSS knowledge, as they still use the underlying mechanisms of CSS properties and media queries.

D. Programming by Example Interface Builders

Programming by Example (PbE) (sometimes also known as Programming by Demonstration, or PbD) is a paradigm that allows non-programmers to write programs by giving examples of its behaviors. PbE has been used for a variety of applications, such as dynamic user interface creation [13]–[15], script and function creation [16]–[19], and word processing [20]–[22]. Existing systems have used a variety of user interaction and inferencing techniques [23], [24]. Important aspects of user interaction include how the user creates and modifies a demonstration, how the PbE system provides feedback to the user, and how the user invokes a program [24]. PbE systems also vary in the inferencing techniques they use; some use minimal inferencing (requiring the user to explicitly specify generalizations), while others use simple rule-based inferencing, and even others use Artificial Intelligence (AI) [23]. Expresso uses PbE to create UIs that are responsive to viewport width. End-users create a demonstration, or "keyframe", for each viewport width they want to explicitly specify the appearance of and then directly manipulate UI elements in the WYSIWYG editor. End-users have the ability to see previously created keyframes and modify them, and they can view the page appearance at different viewport sizes by dragging to resize the viewport. As its inferencing algorithm, Expresso uses linear interpolation of two bounding keyframes to determine page appearance for an intermediate viewport width, adjusting which linear rule is used for a viewport width range when a "jump" transition is specified; "smooth" and "jump" transitions are discussed in detail in the "Transition Behaviors" section below.

The prior PbE work that is most relevant to Expresso are tools that can infer linear constraints between elements and viewports from multiple snapshots or demonstrations: Peridot [13], Inference Bear [25], Chimera [14], and Monet [26]. Although these systems support building fluid UIs, they do not support building responsive UIs, as there is not support for discontinuous jumps between different responsive behavior ranges. Expresso enables building responsive UIs through its smooth and jump transition menu (Fig. 2c).

E. End-User Programming for the Web

More generally, much research has explored end-user programming for the web, including for both custom UI creation and automation, and using a variety of interaction techniques. Chickenfoot [27] allows users to customize existing websites using a simplified language based on UI-oriented keywords and pattern matching. Systems like Inky [28] and CoScripter [29], [30] move closer to supporting natural language interaction with the web, leveraging sloppy programming to allow users to complete and automate web tasks. More recent systems powered by crowdsourcing truly allow end-users to create and interact with web UIs without programming experience. Arboretum [31] allows users to complete web tasks by making natural language requests and handing off controlled parts of a page to crowd workers for completion. Apparition [32] and SketchExpress [33] enable a user to prototype UI appearance and behavior via natural language and sketch descriptions; this is made possible by crowd workers who fulfill these specifications using WYSIWYG and demonstration tools.

F. Automatic UI Layout

Finally, previous research has proposed automatically generating UI layouts based on developer-specified heuristics [34], [35]. Unlike fully automated UI generation tools, Expresso only generates the transitions between user-specified keyframes, which gives the user more control over the appearance of their interface for different viewport sizes.

III. Motivational CSS Study

A. Setup

To better understand the challenges of creating responsive websites, we conducted a study with 8 participants (3 female and 5 male, $\mu = 25$ years old, $\sigma = 5.07$ years). Almost all of our participants were experienced programmers: three participants had at least five years of programming experience in any language, four participants had 2–5 years, and one participant had 6–12 months. Participants had widely varying levels of experience with CSS: two participants had 1–2 years of CSS experience, one had 1–3 months, one had 1–7 days, two had less than one day, and two had no prior CSS experience.

Every participant was given two tasks in an order that was counterbalanced between participants. We adapted our tasks from real-world web pages to ensure that they were realistic. One task involved replicating features of the Mozilla web page (shown in Fig. 1). The other task involved replicating features of a shoe shopping web page (shown in Fig. 2). Both tasks are described in further detail in the Expresso user study task descriptions below, as both tasks were re-used in our evaluation of Expresso.

For each task, participants were given a static (non-responsive) version of the web page and were asked to write CSS to make it responsive. We gave participants animated GIFs demonstrating how the UI should respond to varying width as a user resizes the window. Participants were encouraged to use any online resources (e.g., search engines, tutorials, or libraries) they found helpful and used their preferred code editor. We scheduled study sessions for approximately 1 hour each, and gave participants up to 22 minutes per task to allow time for setup, instructions, and survey.

B. Results and Discussion

We evaluated participants' final web pages according to a rubric (discussed further in the Expresso Evaluation section). Participants achieved a mean accuracy of 45.7% ($\sigma = 23.5\%$). If we calculate task accuracy by participant experience with CSS, we find that the five participants with one week or less CSS experience achieved a mean accuracy of 35.5% ($\sigma = 22.2\%$). The three participants with one month or more of CSS performed better, achieving a mean accuracy of 62.7% ($\sigma = 13.8\%$). These results suggest that, even with the abundant resources (e.g., example code, tutorials, Stack Overflow answers) that are available online, programming responsive UIs is difficult.

To better understand the challenges of writing CSS that participants faced, we analyzed the screen recordings of each participant. One challenge is that a lack of background knowledge makes it difficult to describe the desired rule for a search query. If a participant did not know the name of a relevant CSS keyword, they needed to semantically describe the behavior, which did not always enable them to find the right syntax quickly. For example, in one case, participants were asked to make an element "jump" to the bottom of the page for small page widths. Participants used a variety of search queries, including: "HTML resize to fit screen", "HTML overflow elements", "move item to next line responsive", and "CSS wrap on resize". These search queries are far from the correct CSS keywords — `flex`, `@media`, or `float`. Further, search terms such as "overflow" may semantically make sense but conflict with an existing CSS property name (i.e., `overflow`). Desired behavior can be easily demonstrated visually (e.g., through a GIF or sketch), but variation in the programmers' language descriptions and not knowing relevant domain-specific terms can be barriers in searching for answers to responsive web design questions. The challenge of finding the correct CSS keywords and applying them appropriately is exacerbated by the fact that a relatively small set of CSS properties can have widely different effects on a UI's layout depending on how they are used or combined.

Another challenge was that changing the page's CSS for one viewport size could affect the layout of other viewport sizes. As a result, the intermediate process of correcting the layout for one viewport size could break the layout for other viewport sizes. The fact that existing code runs the risk of breaking something seemed to be discouraging to participants. During the study, we witnessed many incidents where participants found the right CSS properties to set, but in the end decided not to use them as their initial attempt made the website look worse than it had previously for other viewport sizes.

In sum, this study simulates practical situations where non-professional programmers create an initial version of their website without considering responsive design. The results suggest that even for experienced programmers, it can be challenging to build responsive web pages using CSS.

IV. The Expresso System

We created Expresso to enable people with little to no CSS experience to quickly and easily create responsive web pages. Previous research has found that it is often easier to specify the *appearance* of a web page (how elements should display on a page) than it is to define its *behavior* (how the appearance changes depending on user input and page state) [36]. We designed Expresso to let users define a UI's responsive behavior by specifying its appearance in a series of keyframes and specifying how the UI should appear in the states between these keyframes. Given this information, Expresso infers how the UI

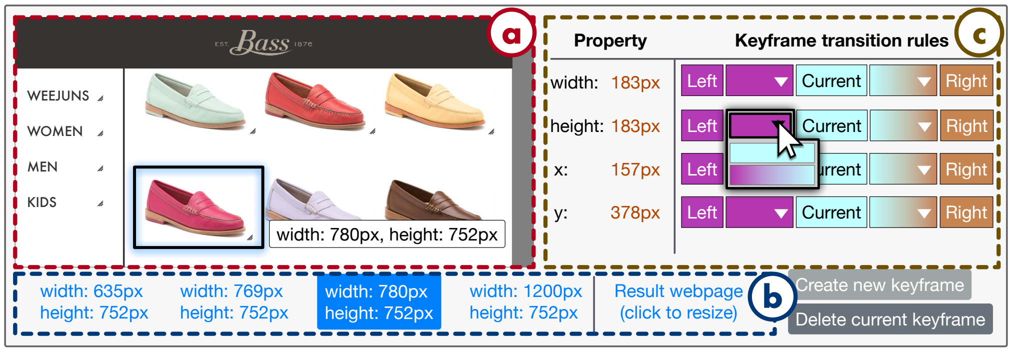
a
Bass
WEEJUNS
WOMEN
MEN
KIDS
width: 780px, height: 752px
c
Property
Keyframe transition rules
width: 183px
height: 183px
x: 157px
y: 378px
Left
Current
Right
b
width: 635px height: 752px
width: 769px height: 752px
width: 780px height: 752px
width: 1200px height: 752px
Result webpage (click to resize)
Create new keyframe
Delete current keyframe

Fig. 2: The Expresso user interface includes (a) a responsive web page viewing area, (b) a menu for switching between existing keyframes (indexed by their viewport width and height) or switching to a resizable preview mode, and (c) a menu for setting element property values and transition behaviors. Here, the keyframe with viewport width 780 pixels is shown with the pink shoe image element selected. The right menu indicates the current transition behaviors between this keyframe and adjacent ones, namely, a "jump" transition between this keyframe and the next smallest, and a "smooth" transition between this keyframe and the next largest.

should appear for those viewport sizes not explicitly defined by a keyframe. In Expresso, users specify a UI's appearance for a given keyframe by simply dragging and resizing elements on a visual canvas, similar to how they manipulate objects in drawing or presentation software. This natural interaction allows users to specify complex rules without ever writing display rules or formulae.

A. Adding Keyframes to Make a Website Responsive

The input to Expresso is a static web page, which is represented as one keyframe in Expresso's user interface. This simulates the scenario where a user wants to modify a static, non-responsive web page to make it responsive. With only one keyframe, the appearance of the web page's elements in this keyframe applies to all viewport sizes, as this is the only knowledge Expresso has about the web page. When the user adds another keyframe, Expresso's default behavior is to create a smooth transition gradient between the two keyframes, meaning that elements move and resize linearly between the keyframes. However, the user can customize how their interface transitions between these keyframes, explained more in the Transition Behaviors section. As the user adds more keyframes, Expresso generates a set of piecewise functions representing element property behaviors and the corresponding responsive UI. The user can view the responsive UI by resizing the web page viewport area.

B. Expresso User Interface

The Expresso interface (Fig. 2) consists of a container on the left for viewing the in-progress web page at different viewport widths, a menu at the bottom for navigating existing keyframes and creating new ones, and a menu on the right for modifying element property values and transition behaviors. The user can change the viewport size in which the web page is viewed by selecting a previously created keyframe from the bottom menu or by resizing the viewport via a drag handle. The web page view area is a WYSIWYG editor which allows direct manipulation of elements. When the user has created a new keyframe or selected an existing one, they can then select elements on the page and drag to reposition and resize them. When an element is selected, the right side menu populates with the element's properties (e.g., dimensions, position, color), current values, and transition behaviors.

The widgets in the right menu (Fig. 2c) support setting range behaviors through the analogy of colors and gradients. The keyframe currently in view is represented as the turquoise "Current" label, the next smallest keyframe is represented as the magenta "Left" label, and the next largest keyframe is represented as the orange "Right" label. The range between colored labels can be either their color gradient or one solid color as chosen from a dropdown widget. In Fig. 2, there is smooth, linear interpolation behavior between the "Current" keyframe and the "Right" keyframe as represented via the turquoise-to-orange gradient. There is a discontinuity in behavior between the "Left" and "Current" keyframes as represented by a solid color; specifically, the solid color of magenta represents behavior continued from the left range of the "Left" keyframe. Fig. 3 illustrates the element property behaviors that each solid and gradient color option encode. Below, we discuss how to set these transitions and their meaning.

C. Viewport Sizes Between Keyframes

User-created keyframes specify the required UI layout at particular viewport sizes. Expresso infers layouts for the other viewport sizes by inferring how every element property transitions between keyframes. For example, for the "Fast for good" text in Fig. 1, the behaviors for the text's font size, x-position, and y-position are inferred individually. Together, these inferred property values define the behavior of the text element across different viewport sizes, and the inferred behaviors of all elements together define the page layouts.

D. Transition Behaviors

We infer element property behavior over the range between two adjacent keyframes. By default, we infer a linear interpolation behavior between two adjacent keyframes. For example, in Fig. 1, the laptop has an x-position of $x_2 = 550$ pixels in keyframe k_2 of viewport width $w_2 = 1000$ pixels, and an x-position of $x_3 = 650$ pixels in keyframe k_3 of viewport width $w_3 = 1200$ pixels. Expresso infers a linear interpolation rule

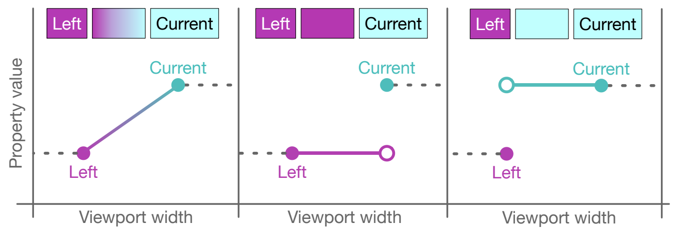

Fig. 3: Graphs illustrating the property behaviors the gradient and solid color dropdown options shown in Fig. 2 encode. Each solid dot represents a keyframe and the line in each graph corresponds to the behavior for the range between the "Left" and "Current" keyframes.

($x = mw + b$) for the laptop x-position for viewport widths $w \in [w_2, w_3]$. The slope m and constant b are calculated based on the (w_2, x_2) and (w_3, x_3) data points provided. Expresso currently only infers linear rules, but other rules, such as higher-order polynomial rules, could be applied under this approach, as we discuss in the Scope section below.

Expresso's linear interpolation inference as described above results in a continuous transition between two keyframes, but not all responsive UI behavior can be represented in this way; some responsive behaviors require consistent properties within a range and discontinuous jumps between ranges. Expresso lets the user encode discontinuous jumps in element property behavior between two adjacent keyframes k_i and k_{i+1}. The location at which the discontinuity occurs affects the behavior for the range of viewport widths $w \in [w_i, w_{i+1}]$.

As a way of specifying the discontinuity, the Expresso user interface allows the user to set the behavior of an element property between two keyframes. For the range $r_{i,i+1}$ between k_i and k_{i+1}, the behavior could be:

- the linear interpolation behavior between k_i and k_{i+1} (which is the default) (Fig. 3, left),
- continued linear behavior from smaller viewport widths (range $r_{i-1,i}$ between k_{i-1} and k_i) (Fig. 3, middle), or
- continued linear behavior from larger viewport widths (range $r_{i+1,i+2}$ between k_{i+1} and k_{i+2}) (Fig. 3, right).

These three behaviors are illustrated in Fig. 3[1]. In this paper, we refer to the first transition type (linear interpolation) as being a "smooth" transition and the second two transition types as being "jump" transitions. For example, the laptop in Fig. 1 should jump in y-position between keyframe k_1 (of width $w_1 = 990$ pixels) and keyframe k_2 (of width $w_2 = 1000$ pixels), as there is a major layout rearrangement between these two keyframes. Thus, the user uses the transition rules menu to specify that for range $r_{1,2}$, the laptop y-position should be that of an adjacent viewport range. In this case they choose for the laptop y-position in $r_{1,2}$ to be that of narrower viewports (i.e., range $r_{0,1}$), where the laptop is at the bottom of the page. This is indicated in the figure as the magenta jump transition, and this viewport range between 990 and 1000 pixels is contained in the region labeled "laptop is centered in viewport".

[1]If the continued behavior from range $r_{i-1,i}$ is chosen for $r_{i,i+1}$, but there is already a discontinuity between $r_{i-1,i}$ and k_i, then the behavior for range $r_{i,i+1}$ will just be the constant value specified at keyframe k_i.

E. *Rule Representation*

As discussed above, in Expresso the behavior of an element property over a viewport size range takes the form of a linear equation. Whether the behavior over a range is a linear interpolation or is continuing that of an adjacent range, the behavior will be linear. Therefore, Expresso represents each element property behavior as a piecewise function, with a sub-function $propertyValue = mw + b$ defined for the range between each pair of adjacent keyframes.

F. *Scope of Supported Behaviors*

1) Single Dimension Dependent: In our examples with Expresso, we limit responsive behavior to be dependent on only one viewport dimension: width. We chose to support responsive behavior with respect to viewport width because we observed that responsive UIs most often react to changes in viewport width, as vertically scrolling a website to view more content is common. Supporting responsive behaviors dependent on one variable only (e.g., viewport width, viewport height, or scroll position) is straightforward, requiring only a first-order polynomial (which Expresso already supports) for fit. To support responsive behavior for a given element property dependent on both viewport width and height would require a higher-order polynomial to be fully expressive.

2) Types of Transitions: In the current implementation, we limit the kinds of transitions to smooth linear interpolation and discontinuous jumps. Other responsive behaviors can be supported using our approach (e.g., quadratic or exponential relationships), but for Expresso we chose linear slopes and jumps since these support continuous and discontinuous transitions, respectively. Future versions of our tool could support more transition behaviors to suit additional use cases.

3) Types of Properties: Expresso currently supports specifying x-position, y-position, width, height, font size, text color, and background color in keyframes. In the future we plan to explore adding other properties to Expresso. For example, many responsive UIs change elements' visibility depending on viewport size to hide or swap out elements. This would essentially be a degenerate case of the current linear equation and discontinuity representation: a UI element's "visibility" attribute would be either "visible" or "hidden" for each continuous range.

G. *Implementation*

Expresso is implemented as a Node.js web application. Raw data about property values for each element for each keyframe are stored in a JavaScript object, which is updated as the user adds keyframes, modifies elements in the UI, and sets transition metadata. An initial, static web page can be loaded into Expresso as a JavaScript Object Notation (JSON) object containing one keyframe. As the user makes updates to their keyframes, Expresso recomputes a piecewise function per element property as explained in the "Transition Behaviors" section above. When the user resizes the page viewport, Expresso updates element CSS property values according to the piecewise functions. Currently, Expresso uses JavaScript to update CSS property values rather than generating dynamic CSS. Future versions could instead generate responsive CSS and relationships via `calc` and the viewport `vw` unit.

Note that elements and their property values in Expresso are represented as a flat hierarchy. Currently there is no notion of elements belonging to a common parent container. Raw element position values in the JavaScript object are relative to the top-left corner of the web page viewport container. Elements are therefore absolutely positioned, independent of each others' positions. Future versions of Expresso could potentially represent elements in a hierarchical manner to better match typical HTML structure, especially if we support importing existing code.

V. Evaluation

We conducted a laboratory study to evaluate whether Expresso can help individuals with minimal CSS experience to build responsive UIs. In our study, we asked participants to use Expresso to build two responsive web pages, for which we provided visual specifications.

A. Participants

We recruited six participants[2] (two female and four male, $\mu = 22.3$ years old, $\sigma = 3.35$ years) with minimal CSS experience. Two participants reported over five years of general programming experience, three participants reported 2–5 years, and one participant reported 1–2 years. All participants reported one year or less of CSS experience; four participants reported a week or less, one reported 2–4 weeks, and one reported 3–6 months.

B. Study Design

The primary goal of our study was to determine how feasible it is for users — particularly those with minimal CSS experience — to build responsive web pages using Expresso. We first gave participants a tutorial of Expresso and then presented them with two responsive web page building tasks to learn how feasible the tool was to use for a variety of responsive behaviors.

1) Tutorial: We gave participants a 15 minute tutorial at the beginning of each session to familiarize the participant with the features of Expresso. In this tutorial, we showed participants an example responsive web page at different stages of its development in Expresso, demonstrating how to achieve different responsive behaviors. In particular, we explained the concept of transitions between two keyframes and how to encode "smooth" and "jump" transitions.

2) Tasks: We presented participants with two tasks each, for which we counterbalanced the order. Each task had two smooth transitions and one jump transition. For each task, participants were given a starter web page with one keyframe (therefore no responsive behavior) and a set of GIFs demonstrating the desired responsive behavior for the web page. Participants were shown four GIFs per task: one GIF illustrating the overall responsive behavior, and three GIFs illustrating the behavior of every transition (one GIF per transition). We used these broken down GIFs in order to help convey the behaviors that they should be building without providing clues about the solution. Participants were asked to encode the responsive behavior in Expresso and were instructed to inform the researcher when they felt they had completed the task or if they could no longer make progress.

We chose to use pre-determined tasks, as opposed to open-ended tasks ("make this static page responsive, however you see fit"), to allow us to better evaluate participants' performance. With open-ended tasks, it would have been difficult to determine if a participant implemented a particular behavior because it was what they wanted or because it was easy.

3) Task web pages: The two web pages we chose for the study are adapted from real web pages, represent different layout styles, and include realistic behaviors. These responsive behaviors include: element resizing relative to the page width, element centering, flexible grid behavior, and arbitrary element rearrangement. The two tasks were:

- Task A: The Mozilla web page[3] (Fig. 1), which consists of a laptop, white text, and a blue background. For wide page viewports, the top half of the page is filled with a blue background, and the text occupies the left side of the viewport and the laptop the right side of the viewport. The laptop remains centered in its blue area on the right. For narrower viewports, the full page height is filled with the blue background, and the text and laptop are stacked vertically and horizontally centered. Each of these two layouts therefore has smooth transition behavior. At a viewport width of 1000 pixels, the layout immediately jumps from one layout to the other.
- Task B: The Bass web page[4] (Fig. 2), which consists of a set of six shoes, a brown banner with "Bass" text, and a left menu. The brown Bass banner always appears at the top of the page with the "Bass" text horizontally centered. For wide page viewports, the six shoes appear in a 3×2 grid, with the shoes shrinking in size and becoming closer together as the page narrows. The left menu also shrinks in width as the page becomes narrower. For narrower viewports, the six shoes appear in a 2×3 grid, with the shoes initially large and then shrinking, and the left menu has a constant width. At a viewport width of 780 pixels, the layout immediately jumps from one layout to the other, resulting in an immediate jump from the 3×2 to 2×3 grid, as the transition widget in Fig. 2c shows.

C. Results

We evaluated the web pages participants created in Expresso against the same rubric we used in the motivational CSS study. Elements that shared the same kind of behavior (e.g., all of the

[2]There was no overlap in participants with the motivational study.

[3]Adapted from
https://web.archive.org/web/20180428062643/https://www.mozilla.org/en-US/

[4]Adapted from
https://web.archive.org/web/20170928121043/https://www.ghbass.com/category/g+h+bass/weejuns/women.do

Statement	Mean rating (1 to 7)	Standard deviation
Using this tool in my job would enable me to accomplish tasks more quickly.	6.33	0.471
Using this tool would improve my job performance.	5.67	0.745
Using this tool would enhance my effectiveness on the job.	5.83	0.898
Using this tool would make it easier to do my job.	6.17	0.373
I would find this tool useful in my job.	6.17	0.373
Learning to operate this tool would be easy for me.	6.67	0.745
I would find it easy to get this tool to do what I want it to do.	5.50	0.957
My interaction with this tool would be clear and understandable.	6.33	1.11
I would find this tool to be flexible to interact with.	6.17	1.07
It would be easy for me to become skillful at using this tool.	6.33	0.745
I would find this tool easy to use.	6.67	0.471

TABLE I: Results of the Technology Acceptance Model (TAM) questionnaire we presented participants, with each statement rated on a scale from 1 (extremely unlikely) to 7 (extremely likely).

white text in the Mozilla example were either all left-aligned or center-aligned), fell under one rubric item. Note that we evaluated accuracy of tasks by reviewing work completed by the 22.7 minute mark. We retroactively chose this cutoff time based on the earliest time we asked a participant to end their work before they had finished. For the Mozilla task (Fig. 1), participants achieved a mean accuracy of 80.7% ($\sigma = 15.9\%$), with a mean completion time of 12.5 minutes ($\sigma = 4.95$ m). For the Bass task (Fig. 2), participants achieved a mean accuracy of 72.2% ($\sigma = 24.6\%$), with a mean completion time of 17.3 minutes ($\sigma = 2.87$ m). Overall, participants achieved a mean accuracy of 76.5% ($\sigma = 21.2\%$), with a mean completion time of 14.9 minutes ($\sigma = 4.70$ m).

After participants completed their tasks, we asked them to complete a TAM questionnaire, with each statement to be rated on a scale from 1 (extremely unlikely) to 7 (extremely likely). When presented with the statement "I would find this tool useful in my job", participants responded with a mean rating of 6.17 ($\sigma = 0.373$). When presented with the statement "I would find this tool easy to use", participants responded with a mean rating of 6.67 ($\sigma = 0.471$). Average results for the full set of TAM statements are reported in Table I.

We also conducted a short interview with participants to better understand their experience using Expresso and how it compared to other user interface building tools they had used. Most participants expressed satisfaction with Expresso, finding that it was easier to use than CSS while also supporting greater customizability than other tools they used:

> P2: *"If I was making a website where I wanted custom control of how all the elements bounced around and I didn't want to constrain myself to some given library that did it all automatically, then I would use this tool..."*

> P4: ["How does your experience using Expresso compare to your experience using other tools?"] *This is definitely much easier. Because with templates, sometimes I will want to add new functionality to that. When that happens, it becomes much more complicated, because I need to first find example code online, how to do that, and then I need to copy that code into my template and debug to make it work for the current template.*

Participants also generally commented positively on the keyframe and transition paradigm that Expresso uses:

> P1: *"In general, just thinking about how you can break up something that has complex behavior into a single keyframe is beneficial because you don't have to worry about everything at once, you can kind of focus on one aspect... Getting the first animation working was fluid and quick because you just start somewhere and end somewhere and you just specify what kind of transition you want."*

> P2: *"The idea of using keyframes seemed very intuitive to me because I've used that sort of design with video editing and animations."*

However, participants did also experience some challenges when using keyframes:

> P1: *"When I was using the tool...I found it kind of hard to think about the different stages of my UI in terms of keyframes."*

> P3: *"If I didn't think about keyframes I actually needed, it became more difficult as I tried to add keyframes later on... I think it would be easier for me if I actually thought about what I was doing first, like making an outline."*

> P5: *"The difficulty was how to select the keyframes. You need to pick out the keyframes at the right time...if you want to shrink smoothly and then suddenly change from 3 columns to 2 columns, in fact you need to insert 2 keyframes here like with similar pixels, but at the beginning I didn't know that because I was not familiar with this pipeline. I only inserted 1 keyframe and found that it was unable to do the job, so I noticed I needed another one."*

Participants also expressed desire for authoring features common in commercial products to make the tool more usable, in particular element alignment, snapping, and centering.

We also observed some interesting usage patterns.

1) Keyframes straddling "jump" were often close in size: All six participants, in at least one task each, placed their two middle keyframes (surrounding the expected "jump" transition) close to each other. The difference in viewport size of the two keyframes was 23 pixels or less in 10 of 12 trials, and was 3 pixels or less in 6 of 12 trials. There are a couple possible reasons for this pattern. In the Expresso tutorial example we presented, there was a 4 pixel difference in viewport size for the two keyframes surrounding the jump transition, so maybe this biased the participants. However, perhaps participants found a small range between the two layout specifications to be advantageous, to have more control over the UI behavior in this viewport size range, or to minimize the viewport range affected by a "jump" transition (which was still a new concept to participants).

2) Trouble when keyframes straddling "smooth" were close in size: In a couple cases each, two participants created keyframes very close in viewport size that straddled a "smooth" transition. They then created significant UI element position and size changes between the adjacent keyframes, which they later realized were not appropriately proportionate to the change in viewport size. When they resized the viewport for testing, side effects included shoes shrinking or growing too quickly (quickly becoming minuscule or taking up the full viewport), or elements flying off the page. In the future perhaps Expresso could warn users when it notices a large UI element property change over a small range, or Expresso could support modifying a keyframe's viewport size after it's been created (i.e., to move the two keyframes further apart).

D. Discussion

As reported in their TAM scores and interviews, participants generally found Expresso to be useful and easy to use. This improvement in self-efficacy can help engage non-programmers in technical problem solving and potentially be usable as a scaffold for teaching computing and programming concepts to non-experts [37].

Although participants reported high TAM scores for Expresso (see Table I), the web pages they built were not perfect according to our rubric (e.g., 76.5% overall accuracy). However, these accuracy scores represent a lower-bound on participants' ability to use Expresso because the error rate includes not only user mistakes in using the tool, but also errors in user intent due to most participants overlooking some aspect of system behavior in the GIFs. Since the participants saw the GIFs for the first time during the task and did not design the web pages and behaviors themselves, they first needed to interpret the behaviors in the GIFs before encoding them with Expresso. One example of a commonly missed behavior was the somewhat subtle shrinking of the left menu in the Bass task.

The relatively high TAM scores indicate that participants found the tool easy to use and useful. This also suggests that participants mostly built what they intended to, even if they misinterpreted the behavior specified by the instructional GIFs. This appeared to be the case from the recordings: when users attempted to demonstrate a behavior, they generally succeeded, and most of the failures we recorded appeared to be due to not taking any intentional steps towards adding it. Since we envision real users to be individuals who already know what specific responsive behaviors they want their user interface to have, the ability to use Expresso to encode intended behaviors is the most relevant success measure. Further, it suggests that understanding and communicating system state and current behavior is a key need for supporting non-programmers. We discuss this further in the next section.

VI. Future Work

Participants were generally successful encoding the necessary transitions into Expresso to complete their tasks, but did not always encode them correctly on their first try, or took time to determine which dropdown menu item they needed to select in order to achieve the desired discontinuity behavior. Future work may explore how to devise and evaluate visualizations to help users better understand the current global behavior of elements across the state space and plan for future modifications. The visualization should also be interactive to support some of these behavior modifications.

Also, as mentioned in the "Scope of Supported Behaviors" section, our keyframe and transition approach could be adjusted to support building web pages that are responsive in both their viewport width and height. One approach would be to use a system of equations with higher-order polynomials (e.g., quadratic functions) to calculate element behavior definitions that satisfy all keyframes in two-dimensional space. This would require additional demonstrations to fully specify. Alternatively, some websites' responsive behavior should be strict per dimension, regardless of the other dimension's value. Supporting separate rules per dimension could be beneficial, but would need to be designed such that conflicts between viewport width and height rules are avoided or easily fixed.

VII. Conclusion

In this paper, we introduced Expresso, a system for creating responsive UIs by specifying keyframes over a UI property (e.g., page width) and setting transitions between them. These keyframes and transitions are used to generate responsive layout rules. We found that even individuals with little to no CSS experience are able to specify complex responsive UIs with Expresso, achieving a mean accuracy of 76.5% in their tasks, and rating it highly on the TAM scale as useful and easy to use. Meanwhile, individuals with similar experience who tried to build these same responsive UIs using CSS were much less successful. More broadly, our work takes a step toward a future in which users can provide intuitive demonstrations to guide the automatic creation of complex UI behaviors.

VIII. Acknowledgements

We thank Yan Chen and Stephanie O'Keefe for their help editing this paper; Jordan Huffaker, Xiaoying Pu, and Kayla Wiggins for their feedback on the Expresso UI; and our study participants for their time and effort. This work was supported in part by Clinc, Inc., and the University of Michigan.

References

[1] H. S. Liang, K. H. Kuo, P. W. Lee, Y. C. Chan, Y. C. Lin, and M. Y. Chen, "Seess: seeing what i broke–visualizing change impact of cascading style sheets (css)," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 353–356.

[2] D. Mazinanian, "Refactoring and migration of cascading style sheets: Towards optimization and improved maintainability," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 1057–1059. [Online]. Available: http://doi.acm.org/10.1145/2950290.2983943

[3] N. Burtnyk and M. Wein, "Computer-generated key-frame animation," *Journal of the SMPTE*, vol. 80, no. 3, pp. 149–153, 1971.

[4] B. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 1, pp. 3–28, 2000.

[5] Apple, Inc. (2003) Xcode. [Online]. Available: https://developer.apple.com/xcode/

[6] Google, Inc. (2013) Android studio. [Online]. Available: https://developer.android.com/studio/index.html

[7] S. Oney, B. Myers, and J. Brandt, "Constraintjs: programming interactive behaviors for the web by integrating constraints and states," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 2012, pp. 229–238.

[8] S. Oney, B. Myers, and J. Brandt, "Interstate: a language and environment for expressing interface behavior," in *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 2014, pp. 263–272.

[9] D. A. Henderson Jr, "The trillium user interface design environment," *ACM SIGCHI Bulletin*, vol. 17, no. 4, pp. 221–227, 1986.

[10] Adobe Systems. (1997) Dreamweaver. [Online]. Available: https://www.adobe.com/ca/products/dreamweaver.html

[11] Webflow, Inc. (2013) Webflow. [Online]. Available: https://webflow.com/

[12] Zine EOOD. (2016) Bootstrap studio. [Online]. Available: https://bootstrapstudio.io/

[13] B. A. Myers, "Peridot: creating user interfaces by demonstration," in *Watch what I do*. MIT Press, 1993, pp. 125–153.

[14] D. Kurlander and S. Feiner, "Inferring constraints from multiple snapshots," *ACM Transactions on Graphics (TOG)*, vol. 12, no. 4, pp. 277–304, 1993.

[15] A. Repenning and T. Sumner, "Agentsheets: A medium for creating domain-oriented visual languages," *Computer*, vol. 28, no. 3, pp. 17–25, 1995.

[16] H. Lieberman, "Tinker: A programming by demonstration system for beginning programmers," *Watch what I do: programming by demonstration*, vol. 1, pp. 49–64, 1993.

[17] H. Lieberman, "Mondrian: a teachable graphical editor." in *INTERCHI*, 1993, p. 144.

[18] T. Lau, L. Bergman, V. Castelli, and D. Oblinger, "Sheepdog: learning procedures for technical support," in *Proceedings of the 9th international conference on Intelligent user interfaces*. ACM, 2004, pp. 109–116.

[19] A. Cypher, "Eager: Programming repetitive tasks by demonstration," in *Watch what I do*. MIT Press, 1993, pp. 205–217.

[20] A. F. Blackwell, "Swyn: A visual representation for regular expressions," in *Your wish is my command*. Elsevier, 2001, pp. 245–XIII.

[21] T. Lau, S. A. Wolfman, P. Domingos, and D. S. Weld, "Learning repetitive text-editing procedures with smartedit," in *Your wish is my command*. Elsevier, 2001, pp. 209–XI.

[22] R. C. Miller and B. A. Myers, "Multiple selections in smart text editing," in *Proceedings of the 7th international conference on Intelligent user interfaces*. ACM, 2002, pp. 103–110.

[23] H. Lieberman, *Your wish is my command: Programming by example*. Morgan Kaufmann, 2001.

[24] A. Cypher and D. C. Halbert, *Watch what I do: programming by demonstration*. MIT press, 1993.

[25] M. R. Frank, P. N. Sukaviriya, and J. D. Foley, "Inference bear: designing interactive interfaces through before and after snapshots," in *Proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques*. ACM, 1995, pp. 167–175.

[26] Y. Li and J. A. Landay, "Informal prototyping of continuous graphical interactions by demonstration," in *Proceedings of the 18th annual ACM symposium on User interface software and technology*. ACM, 2005, pp. 221–230.

[27] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller, "Automation and customization of rendered web pages," in *Proceedings of the 18th annual ACM symposium on User interface software and technology*. ACM, 2005, pp. 163–172.

[28] R. C. Miller, V. H. Chou, M. Bernstein, G. Little, M. Van Kleek, D. Karger *et al.*, "Inky: a sloppy command line for the web with rich visual feedback," in *Proceedings of the 21st annual ACM symposium on User interface software and technology*. ACM, 2008, pp. 131–140.

[29] G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan, "Koala: capture, share, automate, personalize business processes on the web," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2007, pp. 943–946.

[30] G. Leshed, E. M. Haber, T. Matthews, and T. Lau, "Coscripter: automating & sharing how-to knowledge in the enterprise," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 1719–1728.

[31] S. Oney, A. Lundgard, R. Krosnick, M. Nebeling, and W. S. Lasecki, "Arboretum and arbility: Improving web accessibility through a shared browsing architecture," in *Proceedings of the ACM Symposium on User Interface Software and Technology*. ACM, 2018.

[32] W. S. Lasecki, J. Kim, N. Rafter, O. Sen, J. P. Bigham, and M. S. Bernstein, "Apparition: Crowdsourced user interfaces that come to life as you sketch them," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 1925–1934.

[33] S. W. Lee, Y. Zhang, I. Wong, Y. Y., S. O'Keefe, and W. Lasecki, "Sketchexpress: Remixing animations for more effective crowd-powered prototyping of interactive interfaces," in *Proceedings of the ACM Symposium on User Interface Software and Technology*, ser. UIST. ACM, 2017. [Online]. Available: https://doi.org/10.1145/3126594.3126595

[34] J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol, "Generating remote control interfaces for complex appliances," in *Proceedings of the 15th annual ACM symposium on User interface software and technology*. ACM, 2002, pp. 161–170.

[35] K. Gajos and D. S. Weld, "Supple: automatically generating user interfaces," in *Proceedings of the 9th international conference on Intelligent user interfaces*. ACM, 2004, pp. 93–100.

[36] B. Myers, S. Y. Park, Y. Nakano, G. Mueller, and A. Ko, "How designers design and program interactive behaviors," in *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*. IEEE, 2008, pp. 177–184.

[37] D. Loksa, A. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett, "Programming, problem solving, and self-awareness: Effects of explicit guidance," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '16. ACM, 2016, pp. 1449–1461.