HUB: A HTML UI Builder for Rapid Website Prototype Development

Seong Wook Choi
Georgia Institute of Technology
School of Computer Science
schoi330@gatech.edu

Yeon Tae Chung
Georgia Institute of Technology
School of Computer Science
ychung79@gatech.edu

ABSTRACT
Creating functional user interface (UI) layouts can be challenging for designers who are inexperienced in programming. Established tooling for this process has historically provided first-class support for adding UI components to interfaces via text-based markup, but not for doing so in the same visual domain the UI components occupy. The asymmetry between the domain designers conceptualizing interface layouts and that of the underlying representation designers as developers need to create them makes the process of creating UI layouts a major bottleneck in the prototyping stage of design thinking. In this paper, we propose a UI building software that addresses this issue of asymmetry in the development of websites by allowing users to incrementally piece together functional website layouts in the space of the layouts themselves rather than through a Hyper-Text Markup Language (HTML) document. Critically, the HTML UI Builder (HUB) still outputs source code that can be portably deployed in live websites.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces.

General terms: Design**,** Human Factors

Keywords: Sketching, prototyping, gestures, HTML

INTRODUCTION
In the framework of the five stages of design thinking where researchers empathize with their target audience, define the problem space and user needs, ideate a solution, prototype a solution, and test the solution, the prototyping stage can uniquely require technical expertise. This is due to existing prevalent methods of creating functional user interface (UI) layouts requiring designers to be paired with developers or assume the role of developers themselves to program UI layouts using a markup language like the Hyper-Text Markup Language (HTML) or alternatively a general purpose programming language like C/C++. Without the necessary technical skills for creating UI layouts for high-fidelity prototypes, designers are left with no choice but to settle with low-fidelity prototypes even when high-fidelity prototypes are preferable. This is because the development of high-fidelity prototypes takes too much time if designers as developers are learning about the development process at the same time. This, in turn, would make any semblance of iteration in the research workflow impractical.

When considering that the prototype stage of design thinking may be revisited multiple times throughout research, it could be the case that a lot of time may be spent on development as opposed to research itself. This can be especially problematic for research with tight schedules and strict deadlines. For these kinds of research that are highly sensitive to time, the overall productivity of the research is likely to be highly correlated with how well researchers can overcome the challenges involved in developing prototypes in the prototyping stage of design thinking as opposed to those of the other stages. Still, in some cases, low-fidelity prototypes may suffice to explore a research question; however, when low-fidelity prototypes are not adequate for testing, the creation of UI layouts become a major bottleneck in the prototype stage of design thinking. This means that the consequences of not resolving this problem could lower productivity levels overall for research teams that need to develop high-fidelity prototypes instead of low-fidelity ones. This latter case is also more common because high-fidelity prototypes are almost always preferred over low-fidelity prototypes when given the choice from having enough resources to justify the cost. Therefore, there is already a strong incentive to make the development of high-fidelity prototypes easier even when not considering productivity.

The core problem is then, that there is not an intuitive way for designers to develop high-fidelity prototypes. There currently exists an asymmetry in the domain designers operate on and the domain developers operate on. Whereas designers conceptualize UI layouts visually in the same domain as the prototype, it is common for developers to program UI layouts in text-based languages through a text editor or an integrated development environment (IDE). Resolving this asymmetry by allowing designers to draw out their intentions and have the interface understand them would allow designers to create prototypes faster and allocate more of their limited time and energy on other stages of design thinking that may be more pertinent to achieving their research objectives. However, this is not to say that text-based markup and general programming languages are not useful for prototyping, but, rather, that

their successful use is predicated on users' familiarity and mastery of them. This is a condition that is hard to meet because a significant amount of prior training and practice is required to attain such levels of technical skill, making it all the more unlikely for designers in research to have them. Moreover, new standardizations for existing tooling, programming languages, and development strategies are drafted and adopted on a regular basis. Consequently, many methodologies for creating UI layouts become deprecated, rendering technical skill that may have been useful at one point no longer relevant as time passes on. This means that not only is the technical skill required to create UI layouts hard to attain, it is also hard to sustainably maintain even after attaining it at one point.

A solution to this problem would be to make the role of the developer obsolete by allowing designers to use the same design process for developing low-fidelity prototypes for developing high-fidelity prototypes. This can be made possible through software that can either directly convert low-fidelity prototypes into high-fidelity prototypes or interactively recognize smaller-scale drawings in the form of user gestures to concurrently develop UI components of high-fidelity prototypes. Not only would this be a major paradigm shift away from adding UI components to an interface via text-based markup, but this would also eliminate the need for niche technical expertise of building software in research conforming to the design thinking framework. While the development of prototypes can be outsourced, this is not suitable for all research teams due to different research institutions receiving different levels of funding and employing varying models of financing. In this sense, cutting out the middle man entirely in the prototyping stage of design thinking via a new mode of interaction has the potential to significantly alleviate the financial burdens imposed on research. It would not be then too far-fetched to suggest that interfaces affording such interactions would become the norm among research teams across the research institutions at a global scale.

The benefits of solving this asymmetry extend beyond research in academia. That is to say, the demanding levels of technical skill demanded from teams wishing to create high-fidelity prototypes is not exclusive to design thinking, but is also true across any context where prototypes may be developed and tested. Outside of the context of research of human-centered computing (HCC), such software capable of translating low-fidelity prototypes into high-fidelity prototypes could potentially reduce the cost of technical debt across all domains utilizing prototypes. This could happen if the cost of developing new prototypes from scratch is greatly reduced to the extent that the development of new prototypes becomes more economical than the maintenance of legacy prototypes. In such conditions, the maintenance of legacy prototypes would become optional and the eventuality of high-fidelity prototypes being as prevalent as low-fidelity prototypes would become more probable.

With such far-reaching implications for this kind of software, we are motivated to conceptualize and develop one such software in this paper as a first step towards resolving the asymmetry of design and development once and for all.

BACKGROUND

One of the most common environments prototypes are developed for is the web platform. Not only are websites cross-platform and can be easily accessed across a wide range of devices, they are also less invasive than native applications which may require separate software for installing them in specific operating systems. A major focus in the development of prototypes is making them portable and easy for users to use them for feedback that will inform the design of future iterations during the testing phase of design thinking. The ease with which testers can access websites via a uniform resource locator (URL) and the ubiquity of web browsers in personal computers and mobile devices makes the web platform a highly appealing target for research using prototypes. This is the reason we chose the web platform as the primary target of our concerns for rapid prototype development.

With that being said, designing and developing functional UI layouts for the web platform can be tricky because of this fact as well. With there being countless number of mobile device form factors and the notion of resizable windows in desktop environments, websites can take many different forms. For example, some websites are designed specifically for mobile devices and afford a great user experience (UX) for users accessing them on those devices but may not provide a similar quality of experience for users accessing them on desktop environments. This could be due to a lack of foresight on the part of the designers of those websites, a lack of resources to exhaustively support all possible environments through which users can access the website, or a combination of both. The problem illustrated here is that it can be difficult to create UI layouts. In order to help alleviate this issue, we specifically chose to make the rapid development of website UI layouts the main concern of our software. We call the proposed software after this very purpose of creating UI layouts for the web as the HTML UI Builder (HUB).

At a high level, HUB allows designers to incrementally sketch out pieces of a UI in the same space as the overall UI using mnemonic gestures and outputs HTML source code that can be used to create websites. The reason HTML was chosen as the output of source code instead of some other document markup format was because virtually all websites use HTML to some degree to visually display content. In fact, HTML source code is the lowest level of abstraction used by web browsers to display visual content. In other words, HTML source code is a fundamental building block of websites. This fact is also unlikely to change in the near future with there being a countless number of websites and an immense amount of infrastructure relying on it. All this suggests that HUB will

remain relevant for the foreseeable future because it targets HTML source code as an output.

LITERATURE REVIEW

In this section we examine the current state of research in UI prototyping software and explain how HUB differs or takes inspiration from past attempts at accelerating prototype development.

Bassil et al. proposed a system of autonomic HTML interface generator for websites which parses an Extensible Markup Language (XML) schema and a stylesheet containing rules and policies that could be updated easily to maintain a website with minimum user effort [1]. While it automates the process of generating UI based on HTML source code, the focus of this system was to reduce the maintenance cost of existing interfaces, rather than building entirely new ones. This is in stark contrast to the goal of HUB, which is to allow designers to eschew dependency on developers and reliance on technical skill in programming altogether. While working within the framework of existing standards has its merits, it is only a solution to a symptom and not the source of the problem—the problem being the asymmetry in the domain in which designers and developers work.

Pegasus is a drawing system that enables users to construct precise geometries using interactive beautification and predictive drawing [5]. Instead of using human hand drawn images that may pose difficulties to existing computer vision techniques, Pegasus generates standardized shapes and diagrams that can greatly reduce false detection of shape recognition algorithms. The snapping capabilities of HUB was greatly inspired by the autocorrecting features of Pegasus. HUB improves upon this concept by applying more restrictions on the possible configurations of line segments.

DENIM, is a software designed to aid web developers design low-level details of a website, such as navigation and interaction, visually [9]. However, the complexity introduced to the interface to support such advanced features betrays the simplicity designers expect from the interface. Therefore, HUB attempts to solve a smaller, well-defined subset of the larger problem.

SketchWizard, is another system that helps designers to create interfaces [3]. In SketchWizard, both the designers and the end users share a canvas allowing the designer to simulate the intended activities of the end user. This helps developers to identify the functionalities that the users want from the software and modify existing functions based on user interactions. While they can generate visual feedback of low-level details, both DENIM and SketchWizard do not generate source code that may be expanded upon for a final product after the prototyping phase in the software development cycle. We reason that source code native to the target platform is the lowest common denominator for portable, high-fidelity interfaces.

Sketching Interfaces Like Krazy (SILK) was one of the early attempts at transforming drawings into code [6, 7, 8]. SILK recognizes sketches on the input canvas using gestures and converts them into either Visual Basic 5 source code or Common Lisp source code. It allows designers to quickly visualize what their sketches might look like when converted into a functional UI; However, the outputed source code are no longer useful in building functional interfaces today because the languages they are targeting are overshadowed or have since been superseded by newer languages and are no longer in use. To be relevant, HUB, instead, generates HTML source code.

Both Reverse Engineering Mobile Application User Interfaces (REMAUI) and MobiDev, focus on developing interfaces for mobile applications [11, 13]. REMAUI transforms high-resolution screenshots into deployable code using a series of computer vision techniques such as edge detection using Canny's algorithm, edge dilation, and contouring to identify atomic visual elements and reconstruct the intended UI component hierarchy. MobiDev offers two different modes in creating interfaces. The first mode is a typical UI builder where users can choose to lay down various UI components onto the screen to create their desired layout. The second mode is called "SketchBuilder" where a picture of the drawing is taken and a program interprets and generates the corresponding interfaces.

Pix2code is another UI prototyping software that transforms high-resolution screenshots of drawings into source code for interfaces targeting both mobile applications and websites [2]. Through the use of recurrent neural networks (RNN) it is able to replicate the intentions of the original UI conceptualized by designers with approximately 77% accuracy. Further improvements on the accuracy of recognition was made by Liu et al. through the adaptation of bidirectional LSTM which increased the accuracy to 85% [10, 14].

Sketch2code is an interface generator that transforms human sketches into source code using deep learning neural network models, namely the artificial neural network (ANN) model, to successfully parse and identify the various UI components from an image [4, 12]. The neural network model training process is divided into three steps: pre-processing, segmentation, and post-processing. This yields promising results in converting drawings into HTML source code.

While Pix2code and Sketch2code both yield promising results in transforming high-resolution images of sketches and drawings into useful source code, they lack the ability to convert drawings into interfaces in a deterministic manner. Furthermore, Sketch2code does allow conversion of drawings into HTML source code similar to HUB, but it lacks the capability to modify the generated layout in real-time, and changes can only be made through submitting a new drawing. In this sense, HUB stands out with its translation and real-time rendering capabilities.

SYSTEM DESCRIPTION

As mentioned before, HUB generates HTML source code from hand-drawn sketches drawn on a digital canvas representing the space of a website. The digital canvas where the interface receives user input shows the UI components that have already been added to the current layout by the user from previous interactions. We achieve this by superimposing a drawing canvas viewport on top of an HTML rendering viewport such that user input can be captured as if the user is drawing on top of the UI layout just as one would draw on paper when creating a low-fidelity prototype. The illusion of drawing on top of the UI layout is achieved by allowing users to draw in the same space relative to the position of the rendered UI components in the rendering of the visual UI layout. In this dual arrangement of viewports, the drawing canvas viewport is responsible for receiving and parsing user inputs and the HTML rendering viewport is responsible for rendering the website UI components as they would appear in the main rendering viewport of a web browser. The main advantage of this approach to laying viewports on top of each other to receive user input and render in the same space is that the resulting layout allows users to directly draw UI components where they want components to be in addition to what size they want them to be. This is important because it makes the process of conceptualizing visual UI layouts and the process of implementing them symmetric in that they happen in the same visual domain

It is also worth noting that the resulting UI layouts are static and do not dynamically adjust to the width and height of the rendering viewport or, correspondingly, the width and height of the final website. The motivation for making the UI components static is to make it less confusing to the designer how the UI components behave in different widths and heights sizes. This way, there is no ambiguity in how the UI components that designers draw onto the UI layout behave in different device form factors. This is connected to the fact that the drawings that designers draw onto the UI layout are inherently static in nature and do not contain information about how the UI components react to different widths or heights of the final website display viewport or changing widths and heights of the final website display viewport over time. This is also in line with how Wizard of Oz prototyping employed by low-fidelity prototypes are composed of static UI components which may or may not have dynamic properties depending on how they are manually, physically moved around.

Additionally, the UI layouts are statically positioned relative to the origin position of the global space on the top-left corner of the final website viewport, which effectively makes them anchored to this corner. This means that when or as the width and the height of the final website viewport changes, the statically positioned UI components get repositioned with respect to the top-left corner. This mirrors the default behavior of the default UI components natively supported by HTML and the prevalent convention of the top-left corner being the origin with the coordinate of (0, 0) in the context of a 2-dimensional Cartesian plane representing the viewport of the final website.

Despite all UI components generated in HUB being static and globally anchored to the top-left corner of the final website viewport, they can be selected and dynamically manipulated by the user. Users can select UI components by simply clicking with a mouse or tapping with one of their fingers anywhere on the space occupied by the UI components. The space of a UI component is bounded by the rectangle bounding box aligned perpendicularly to the horizontal and vertical axis of the final website viewport. When a UI component is selected, its bounding box becomes visible in the form of dotted lines tracing the edges of its outermost boundary. Only one UI component can be selected at a time, meaning that when another UI component is selected, the previously selected UI component gets deselected. When a UI component gets deselected, its bounding box no longer becomes visible. This makes the dotted lines highlighting the bounding box of the currently selected UI component a reliable visual indicator of whether a UI component is selected and which UI component is currently selected.

Once a UI component is selected, it can be manipulated in multiple ways. One way a UI component can be manipulated is via drag gestures through the mouse or touch. When a user initiates a drag gesture and the the initial position of contact is within the interior area of the bounding box and not on the bounding box border marked by dotted lines, the selected UI component is moved and repositioned by the same distance and direction of the drag gesture—giving the impression that the UI component is being dragged around in the final website viewport. The distance and direction are calculated by taking the 2-dimensional vector difference of the current position of the user point of contact with the final website viewport by the initial position of contact that started the drag gesture. This calculation happens in real-time with the position of the UI component updating as often as possible to provide the user visual feedback that the drag gesture is happening and a means for the user to visually confirm that the drag gesture is working and that the UI component is in the intended position.

The drag gesture can also be used to resize UI components in the final website viewport. When a user initiates a drag gesture and the initial position of contact is on the dotted lines marking the bounding box border and not within the interior area of the bounding box, the the selected UI component is resized by the same distance and direction of the drag gesture with respect to the opposite corner or side of the bounding box. This means that UI components being dragged from the top-left corner are resized with the bottom-right corner fixed, UI components being dragged from the top side are resized with the bottom side fixed, UI components being dragged from the top-right corner are resized with the bottom-left corner fixed, UI components being dragged from the right side are resized with the left

side fixed, UI components being dragged from the bottom-right corner are resized with the top-left corner fixed, UI components being dragged from the bottom side are resized with the top side fixed, UI components being dragged from the bottom-left corner are resized with the top-right corner fixed, and UI components being dragged from the left sided are resized with the right side fixed. The distance and direction of the drag gesture are calculated in the same manner as the drag gesture for repositioning UI components and the timing of the updates likewise happens in real-time. In this context, the real-time feedback provides a user a means to know UI components are being resized and visually verify that the width and height of the UI component being manipulated is the intended length and proportions.

Selected UI components can also be deleted through keyboard shortcuts. Specifically the currently selected UI component is deleted when the user presses the "Delete" key on the keyboard. The current UI layout can be deleted and an entirely new UI layout can be created from a blank slate with the "Ctrl+W" keyboard and "Ctrl+N" keyboard shortcuts that close the current window and create a new window respectively.

Another way a UI component can be manipulated is via the context menu. Users can pull up the context menu via a right click with a mouse. Analogously to the drag gesture allowing users to resize UI components, the context menu provides the options to change the width or height dimensions of the currently selected UI component via a pop-up window allowing users to input numerical values for the width and height in pixels. The context menu also provides a means to change the z-order of UI components. By default, UI components are ordered in the order they are created with the most recently created UI components being closer to the front. Additionally, the context menu allows users to change the contents of the selected UI component as well as the type of the UI component via a pop-up window. For example, if the UI component visualizes textual content, this option of the context menu will allow users to change the type of text and modify the text visualized by the selected UI component. Finally, the context menu allows users to delete the currently selected UI component.

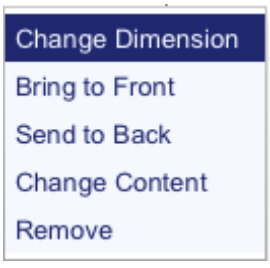

Figure 1: The context menu providing the option to change the dimensions, z-order, and content of selected UI components in addition to the option to remove them.

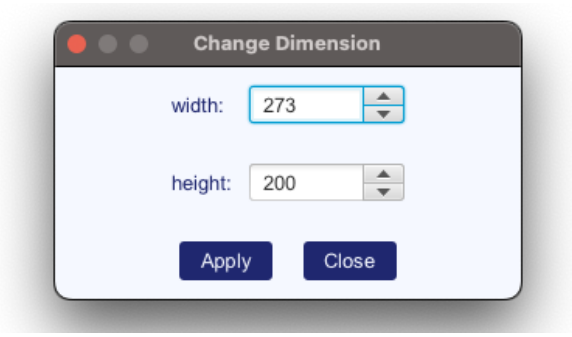

Figure 2: The pop-up window allowing users to input numerical values for the currently selected UI component's width and height in pixels.

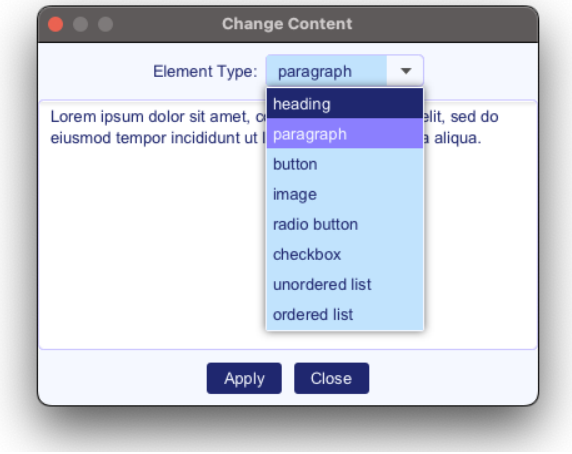

Figure 3: The pop-up window allowing users to modify the content of the currently selected UI component or change its type.

Similarly to the context menu, a tool panel separate from the final website viewport allows for fine-grained control of selected UI components. The tool panel provides the same options as the context menu for changing the dimensions, z-order, and content of selected UI components. The tool panel also provides the same option as the context menu for removing UI components from the UI layout as well. Similarly to the outcome achieved with the keyboard shortcuts for closing and creating a new window, the tool panel also provides the option to remove all UI components in the UI layout regardless of whether they are currently being selected. Although there is redundancy in the options the context menu and tool panel provides, the tool panel is important because it allows users who are not using a mouse to interact with HUB to be able to have easy access to the covered options. We realize that a computer mouse may not be the optimal choice of input for an application that requires gesture drawings. Therefore, HUB is designed to work well with both a computer mouse and a stylus. While a desktop user can simply use the left click of the mouse to draw gestures on the canvas and use the right click to transform gestures into html components or edit component properties, a user with a stylus can utilize the

tool panel on top to achieve the same user experience. The tool panel essentially includes all the features that can be called by using the right click on the component, which opens a context menu. Options unique to the tool panel is the option to make the interface interpret the strokes drawn onto the viewport and the option to remove all of them from the viewport.

The menu bar also provides similar options as the keyboard shortcuts, context menu, and tool panel. The “File” submenu option allows users to close and create new windows like the aforementioned keyboard shortcuts. The “Edit” submenu option allows users to change the dimensions, z-order, and content of the currently selected UI component, as well as removing it and removing all of the UI components in the UI layout. An option uniquely provided by the menu bar is toggling the bounding boxes of all UI components in the UI layout under the “View” submenu option. Another option uniquely provided by the menu bar is the option to view the mappings of all possible gestures and UI components under the “Help” submenu option.

After any changes are made to the viewport, the HTML rendering in the final website viewport is updated to reflect those changes. Synchronizing UI components created in the virtual canvas and those in the HTML rendering happens instantaneously in real-time to provide users with feedback on the outcome of the changes. This enables users to always be aware of the correct UI layout from the latest changes. The status bar serves to give further confirmation on the most recent action made by users and changes made on the UI layout.

The HTML source code editor is another component of the HUB interface. Any updates to the UI layout such as the creation or deletion of UI components or the repositioning or resizing of UI components are reflected in the HTML source code in the HTML source code editor in real-time. This section of the interface provides users with the option to conveniently highlight and copy the HTML source code generating the indicated UI layout in the final website viewport to the clipboard. Directly editing the source code is not possible, however. This one-way dependency of updates is deliberately designed to force users to development of UI layouts in the visual domain as opposed to in a text-based format. This way, the problems associated with asymmetric development of UI layouts can be avoided through deliberate limitations in affordances. For the same reason, syntax highlighting of the HTML source code is not supported in the HTML source code editor. Again, we stress that HUB is not a complete solution to the problem of asymmetry in the domains of design and software engineering and is rather positioned as a first step towards a complete end-to-end solution for designers in need of high fidelity prototypes. With the output of HUB being HTML source code and the end goal of users being to rapidly create functional website prototypes, it is important that users have an easy way to access the most up-to-date HTML source code and the HTML source code editor affords this.

In summary, HUB consists of two main sections: a preview rendering of the generated HTML source code that also acts as a virtual canvas users can draw on with mouse click or finger tap and drag gestures and an HTML source code editor. HUB also incorporates standard interface components like a context menu, a tool panel, and a menu bar for quick access to options and controls that manipulate UI components in the UI layout or the behavior of the interface. A status bar is also incorporated to provide users with continuous feedback.

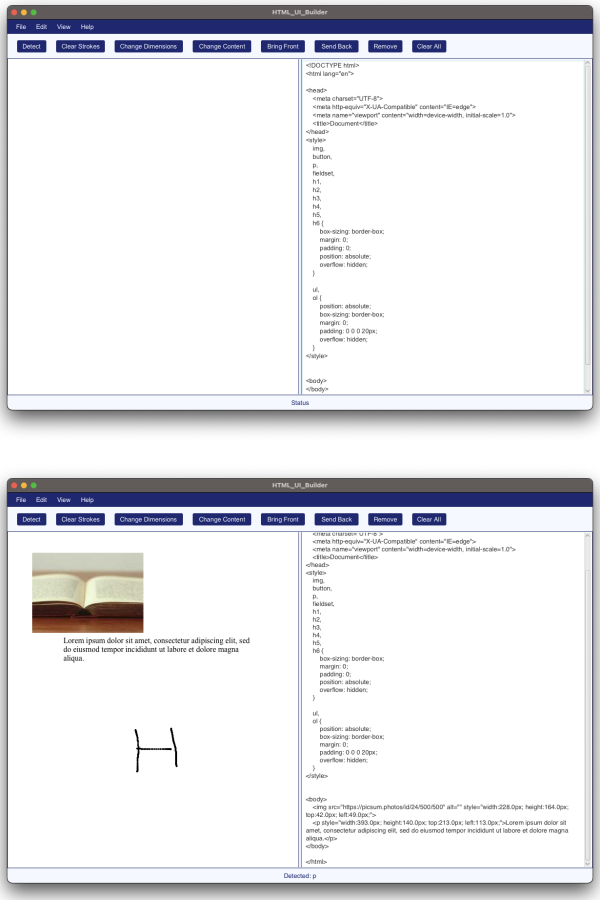

Figure 4: The HUB UI layout consists of an HTML rendering viewport and an HTML source code editor, in addition to other ancillary components: a tool panel, a menu bar, and a status bar.

There are eight UI components supported by HUB, namely images, headings, paragraphs, buttons, radio buttons, check boxes, unordered lists, and ordered lists. Each UI component has a unique gesture associated with it. Strokes for these gestures can be drawn by the user through mouse click or touch and drag gestures. When a user clicks or touches and drags anywhere on the virtual canvas viewport that is not in the space occupied by an existing UI component, a strake is drawn. When a user is done with

drawing the gesture, they can use the detection option in the tool panel to have HUB detect the UI component corresponding to the gesture made. Once a UI component is identified, the corresponding HTML source code with the correct placement with respect to the sketch is injected into the generated HTML document structure.

Figure 5: The pop-up window accessible through the "Help" submenu option that shows a list of gestures labeled by the UI components they are mapped to.

Figure 6: The default HTML rendering of an image UI component.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed

Figure 7: The default HTML rendering of a heading UI component.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 8: The default HTML rendering of a paragraph UI component.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et

Figure 9: The default HTML rendering of a button UI component.

default text
default text
default text

Figure 10: The default HTML rendering of a radio button UI component.

default text
default text
default text

Figure 11: The default HTML rendering of a checkbox UI component.

- default text
- default text
- default text

Figure 12: The default HTML rendering of an unordered list UI component.

1. default text
2. default text
3. default text

Figure 13: The default HTML rendering of an ordered list UI component.

$Q, an upgraded version of $1 recognizer is used for the gesture detection due to its ability to handle unistrokes and multistrokes equivalently regardless of the stroke order, meaning the users do not have to worry about following the exact number or order of strokes when drawing the shapes on the virtual canvas [15, 16]. The following is the basic workings of the $Q recognizer:

1. Normalizing the user input into a set number of points.
2. Matching each point in the candidate with the closest point in the template.
3. Calculating the total difference in distance of all the pairs..
4. Repeating steps 2 to 3 for all predefined templates.

In order to increase computation efficiency, the $Q recognizer utilizes a look up table and pruning, making it suitable for not only desktop applications, but also for low resource devices.

CONCLUSION

To conclude, HUB helps designers rapidly develop UI layouts for high-fidelity website prototypes without requiring them to have technical expertise in web development or be familiar with HTML and manipulating HTML source code. In this way, the design process and development of functional prototypes is seamlessly combined and the possible static UI layouts that can be generated through the interface is limited only by the creativity of the designers using it.

REFERENCES

[1] Youssef Bassil and Mohammad Alwani. 2012. Autonomic HTML interface generator for web applications. arXiv preprint arXiv:1202.2427 (2012).

[2] Tony Beltramelli. 2018. pix2code: Generating code from a graphical user interface screenshot. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, 1–6.

[3] Richard C Davis, T Scott Saponas, Michael Shilman, and James A Landay. 2007. SketchWizard: Wizard of Oz prototyping of pen-based user interfaces. In Proceedings of the 20th annual ACM symposium on User interface software and technology, 119–128.

[4] John J Hopfield. 1988. Artificial neural networks. IEEE Circuits and Devices Magazine 4, 5 (1988), 3–10.

[5] Takeo Igarashi, Sachiko Kawachiya, Hidehiko Tanaka, and Satoshi Matsuoka. 1998. Pegasus: a drawing system for rapid geometric design. In CHI 98 conference summary on Human factors in computing systems, 24–25.

[6] James A Landay. 1996. SILK: sketching interfaces like krazy. In Conference companion on Human factors in computing systems, 398–399.

[7] James A Landay and Brad A Myers. 1995. Interactive sketching for the early stages of user interface design. In Proceedings of the SIGCHI conference on Human factors in computing systems, 43–50.

[8] James A Landay and Brad A Myers. 2001. Sketching interfaces: Toward more human interface design. Computer 34, 3 (2001), 56–64.

[9] James Lin, Mark W Newman, Jason I Hong, and James A Landay. 2000. DENIM: Finding a tighter fit between tools and practice for website design. In Proceedings of the SIGCHI conference on Human factors in computing systems, 510–517.

[10] Yanbin Liu, Qidi Hu, and Kunxian Shu. 2018. Improving pix2code based Bi-directional LSTM. In 2018 IEEE International Conference on Automation, Electronics and Electrical Engineering (AUTEEE), IEEE, 220–223.

[11] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse engineering mobile application user interfaces with remaui (t). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 248–259.

[12] Alex Robinson. 2019. Sketch2code: Generating a website from a paper mockup. arXiv preprint arXiv:1905.13750 (2019).

[13] Julian Seifert, Bastian Pfleging, Elba del Carmen Valderrama Bahamóndez, Martin Hermes, Enrico Rukzio, and Albrecht Schmidt. 2011. Mobidev: a tool for creating apps on mobile phones. In Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services, 109–112.

[14] Richard Szeliski. 2022. Computer vision: algorithms and applications. Springer Nature.

[15] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. 2018. $ Q: a super-quick, articulation-invariant stroke-gesture recognizer for low-resource devices. In Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services, 1–12.

[16] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. 2007. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In Proceedings of the 20th annual ACM symposium on User interface software and technology, 159–168