Creating Dynamic Prototypes from Web Page Sketches

Creating Dynamic Prototypes from Web Page Sketches

Tommaso Calò
Politecnico di Torino
Torino, Italy
tommaso.calo@polito.it

Luigi De Russis
Politecnico di Torino
Torino, Italy
luigi.derussis@polito.it

Abstract

While web designers draw user interface sketches as a first step toward creating a Web application, transforming those sketches into a prototypical coded interface is still a manual and time-consuming task. Recently, researchers focused on easing this part of the design process by applying machine learning techniques to generate code from sketches automatically. These methods effectively convert a sketch into a skeleton structure of the web page but are not designed to deal with dynamic behaviors of the page, such as links, buttons, or dropdown menu. Indeed, to our knowledge, they only allow the creation of static prototypes. In this paper, we move the first steps to support the creation of dynamic prototypes from sketches. We introduce both a set of symbols that a designer can use on their sketches to model dynamic behaviors and the related implementation to generate dynamic prototypes. Finally, we test our method on a few sketched components to assess the suitability of the approach.

CCS Concepts: • **Human-centered computing → Graphical user interfaces**; *Interface design prototyping*; • **Computing methodologies → Machine learning**; **Computer vision**.

Keywords: machine learning, web elements, user interface, convolutional neural network

ACM Reference Format:
Tommaso Calò and Luigi De Russis. 2022. Creating Dynamic Prototypes from Web Page Sketches. In *Proceedings of Programming Abstractions and Interactive Notations, Tools, and Environments Workshop (PAINT 2022).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Designers of web sites typically go through a process of progressive refinement [1]. They tend to think about the larger picture, such as the overall site layout, at first, and then progressively focus on finer details, such as the specific look of page elements, typefaces, and colors.

The design process often includes rapid exploration early on, with designers creating many low-fidelity sketches on paper. There are several benefits of sketching during this phase of design. Sketches allow the designer to focus on basic structural issues instead of unimportant details. Sketching is quick, so designers can rapidly explore different ideas and iterate on those. In addition, user studies using rough prototypes tend to find the same usability problems as do tests with more finished prototypes [2, 3]. However, transitioning from those sketches to a coded interface with a suitable look-and-feel is still a manual and time-consuming task [4, 5].

Supporting this transition is challenging due to the diversity of sketches and the complexity of coded graphical user interfaces (GUIs). The research community, therefore, has a high interest to find methods and tools able to support designers in the process of moving between prototypes of the user interface. Several research projects, indeed, have tried to automate this translation. For instance, Beltramelli [6] proposed Pix2code, an end-to-end approach based on Convolutional and Recurrent Neural Networks that allows the generation of code from a mock-up screenshot taken as an input. Robinson [7], instead, presented sketch2code, a system to automatically transform hand-drawn sketches into coded GUIs. Both these works capture well the overall structure of the user interface and translate it into code, but they are not designed to consider the *dynamic behavior* (e.g., links between pages, dropdown menus) of the generated interface, which remains a manual and expensive task to be applied. Indeed, the chance to embed the dynamic behavior directly in the sketch might further empower the designers in their creative work.

In this paper, we put forward a novel approach where we focus on translating a sketch of a Web interface to the related code, allowing the designer to specify the *dynamic behavior* of the sketched elements directly in the sketch. To do so, we introduce a set of symbols that a designer can use on their sketches to model such dynamic behaviors. The symbols are in some cases well established in Web visual languages, e.g., the down-facing arrow to indicate a drop-down menu, while in other cases are introduced from scratch. In our proposed method, we segment the input sketch to derive the single components of the web-based interface and their positions;

PAINT 2022, December 5–10, 2022, Auckland, New Zealand

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/XXXXXXX.XXXXXXX

then, for each component, we run a Convolutional Neural Network (CNN) to classify its structural properties and identify the relative position and type of the symbols used to model dynamic behaviors. Finally, a parser elaborates the information derived by the network to generate the backbone code of the interface. We tested the method on a subset of the presented symbols to evaluate its effectiveness.

2 Related Work

2.1 Automatic Generation of Code from Sketches

Although the generation of computer programs is an active research field, program generation from visual inputs (like sketches) is a relatively under-explored area. The problem of generating code from visual inputs is strictly related to the problem of automatically reverse-engineering GUIs: reverse-engineering approaches are mainly applied to generate code from GUI mock-ups or screenshots. Nguyen and Csallner [11], for instance, proposed a method to reverse-engineering Android user interfaces from their screenshots. However, their method heavily relies on heuristics and expert knowledge to be implemented successfully, so its applicability is restricted to a limited domain of interfaces. Similar approaches have been used to create tools able to generate code from hand-drawn wireframes. These tools [12, 13] are useful for designers who wish to quickly sketch and prototype possible interface layouts.

A more complex version of this task is generating code from complete screenshots, as it requires that the system handle the stylistic and structural variation present in real-world app screens. Pix2code [6] was one of the first works attempting to address the problem of GUI code generation from visual inputs by leveraging machine learning to learn latent variables instead of engineering complex heuristics. To exploit the graphical nature of the input, Pix2code approaches the problem of converting screenshots to code as an image captioning problem. Another work very close to ours is Sketch2Code [17]. Sketch2Code approaches the problem similarly to Pix2code, with the difference that the author trains the model on a specially-prepared dataset of GUI sketch images.

As depicted, none of these previous works focus their attention on the behavior modeling of prototypes. Our approach aims at filling this gap by introducing a method similar to Sketch2Code since we start from sketches and we implement a CNN to infer the structural properties of the sketched Web component, but we consider and detect the symbols which models dynamic behaviors. We then use a parsing procedure to generate the backbone code of the prototype, dynamic behaviours included.

2.2 Behavior Specification

In the comparative study by Silva et al. [22] *behavior specification* is defined as the ability to add dynamic behaviors to prototypes. "Behavior" is described as a set of states that prototypes can reach by the means of transitions between states. Very few prototyping tools model the dynamic behavior of the prototype, the majority allow to create static mock-ups, only. As described in [22], the main methods to specify the behavior of the prototype are setting hotspots on images, and events handling on widgets. Hotspots are areas highlighted on top of the sketch of the prototype to capture events triggered by the user [8]. Designers need to create one hotspot for each part of the interface they want to make interactive. The problem with this method is that hotspots are associated with graphical areas that are not semantically linked with the graphical element represented in the image, but only on the coordinates of the hotspot.

Wireframe tools use widgets to build the interface [9] and they model the dynamic behavior of the prototype directly on the widgets with event handlers. The event handlers usually specify an action required to trigger the event and the behavior the event triggers. Balsamiq [15], ActiveStory Enhanced [14], SILK [13] and DENIM [16], are examples of tools supporting wireframe interactions. Tools like AppSketcher [18] or JustInMind [19] allow to specify conditions, edit properties, or use variables; Appery.io [20] and ScreenArchitect [10] allow also to program code.

Our approach models the dynamic behavior of the GUI directly from the sketch itself with the usage of a set of specific symbols. It supports wireframe interactions without the need of adding widgets or hotspots, in a later stage. Moreover, to our knowledge, the proposed method is the first attempt to model behavior specification directly from a sketch by using convolutional neural networks.

3 Method

In this section, we present the method to automatically generate code from sketches, along with the novel introduction of a first set of symbols and the related procedure to model dynamic behavior.

3.1 Modeling Dynamic Behavior

To model the dynamic behavior of the prototype, we introduce a set of symbols that represent different dynamics behaviors. Such symbols are supposed to be drawn directly on the sketch and are chosen based on the fundamental dynamic properties emerged in literature, i.e., from [13, 22]. The following set is chosen to demonstrate the applicability of the model, and will be expanded in future works to model a wider range of dynamic behaviors:

Default Selected Element indicates the item that is selected by default in the sketched interface. An example of such a item is the "Home" button in an horizontal navigation bar of a web application.

Dropdown Menu indicates that the element opens a dropdown menu.

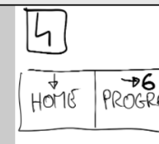
SYMBOL	FUNCTION	MOTIVATION	USAGE	EXAMPLE
	Default selected element.	The symbol has been chosen for its visual similarity with an anchor.	The symbol must be drawn upon the element to be selected by default.	
	Dropdown menu	The symbol has been chosen because it is the standard representation of the dropdown menu in literature.	The symbol must be drawn below the element that activates the menu	
	Page Indicator	NOTE: the symbol can be any number.	The number must be written inside a square in the upper left of the sketched page that it uniquely identifies	
	Link	The symbol has been chosen for its visual meaning of motion.	The symbol must be drawn above the button and should be followed by the page unique number to which the link points to.	

Figure 1. The proposed set of symbols to model dynamic behavior in sketch prototypes: The *default selected element* symbol is used to model the item that is selected by default in the given interface; the *dropdown menu* symbol indicates that the element opens a dropdown menu; the *page indicator* symbol is used to link together different page sketched by the designer; the *link* symbols, links a sketched element into an indicated page.

- **Page Indicator** uniquely represents the sketched interface, e.g., the page destination of a link.
- **Link** represents the connection between the sketched element and an indicated page.

Figure 1 reports the four introduced symbols, along with their functions, the motivations behind the chosen representations, their usage, and an example for each symbol. The figure could ease the understanding of how the symbols are implemented in a real sketch of a GUI.

3.2 Prototype's Interface Generation

The task of generating the code of an interface from a sketch can be split into three sub-problems.

First, the problem of segmenting the sketch by semantic elements, e.g, navbar, list, carousel (Section 3.2.1). Secondly, for each given semantic component, the problem is to understand the sketch's structural properties, and infer which are the present symbolic elements, and their positions (Section 3.2.2). Finally, the last challenge is to generate the code of the resulting component, taking into account the expressed dynamic behavior (Section 3.2.3).

The presented approach has been implemented using Python 3.8. The neural network has been implemented using PyTorch, the images processing with PIL and OpenCV, while the data processing has been conducted with Pandas and Numpy.

3.2.1 Sketch Understanding. Understanding the sketch is a computer vision task that, given the sketch of a Web-based interface, consists of the detection and identification of the included components (e.g., buttons, navbars, etc.) and their relative position.

For this task, we adopted the same method of Sketch2Code [17], which uses RetinaNet [21], a popular single-stage detector that is accurate and runs fast. RetinaNet can simultaneously predict both the class and the box position of the object under detection. Figure 2 (1) displays the network architecture.

3.2.2 Components Understanding. Given the segmented sketch of the user interface, we predict the structural properties of the components, along with the presence of symbols to model dynamic behavior.

In detail, as depicted in Figure 2 (2), we implement a convolutional neural network, specific for each component, trained to classify the structural properties of the sketched component, e.g., in the case of a navbar, the number of elements floating left and right. We use the same network to predict which (*symbol type*) and where (*in which element*) symbols are present. To link multiple pages, we used the page indicator, i.e., a unique number, written on the top right of the sketched page.

3.2.3 Code Generation. Given the structural properties of the component, along with the type and the position of

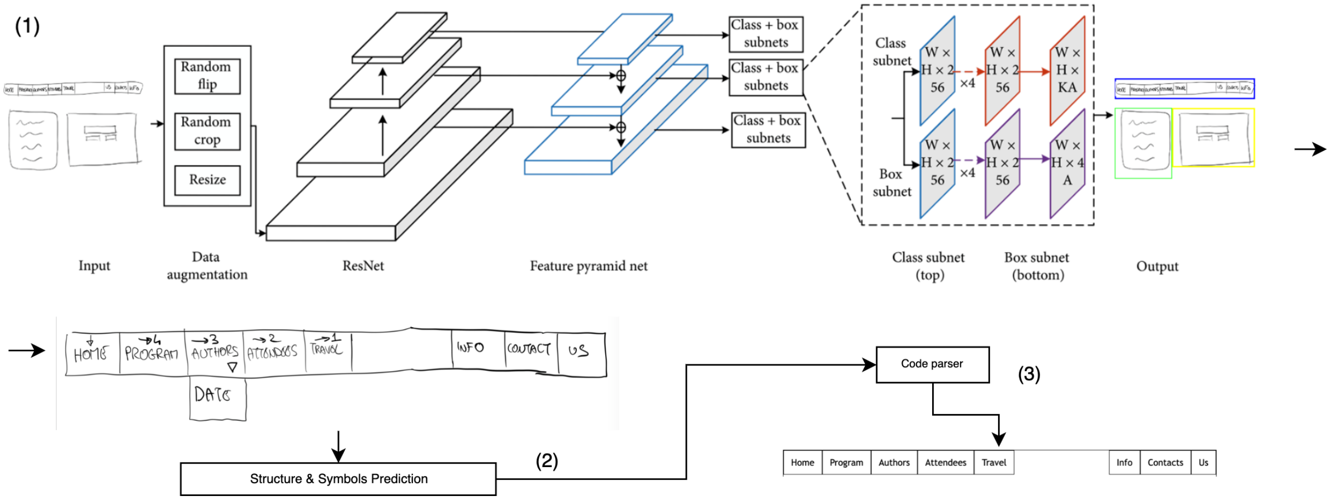


Figure 2. Method overview. Starting one or multiple sketches of interfaces, in (1) we perform a segmentation of the single sketch in sub-components. Then, for each component, we use a convolutional neural network to infer its structural and dynamic properties (2). Eventually, with the help of a parser, we translate the predicted properties into the backbone code of the sketched component. (3) shows the rendered Web element stemming from the entire process.

symbols present to model the dynamic behavior, we proceed to generate the code of the backbone using a parsing function. Figure 2 (3), shows the final rendered component.

4 Experiments

We want to verify that our method correctly generates the navbar's code with the inserted dynamic behavior, as well as the structural properties of the sketch, and can correctly recognize the symbols that model such a dynamic behavior. Due to the unavailability of a dataset of sketched Web interfaces, we train our method over a synthetic dataset and then we fine-tune it over a collection of 50 real sketched navigation bars (navbars). Since the segmentation and reconstruction methods were adopted and already validated by Sketch2Code [7] and UICode [23], we do not report the performance of those methods in the paper.

Dataset. To test the effectiveness of the method we built a synthetic dataset of 3,000 navbars' sketches. Each navbar can have at most five items floating left and three items floating right. Regarding the symbols to model the dynamic behaviors, we had only one default selected element per navbar, multiple links, multiple dropdown menus, and a unique page indicator. The aim of the structure prediction model is to infer the number of rights and left items. In addition, we fine-tuned the resulting model to 500 real-sketched navbars to evaluate the performances of the model in a realistic scenario. The real sketches dataset presents more variability of the synthetic dataset, with hand drawn lines, overlapping and mispositioned elements.

Measures. The Convolutional Neural Network (CNN) must be able to classify correctly the structural features as well as the type and position of the symbols in the sketched component in order to parse them into code. We utilize *accuracy* as the main measure of performance.

Experiments. To evaluate the performance of the CNN for the *sketch structure prediction*, we split the synthetic sketch dataset into 2,500 train samples and 500 test samples, we trained the network 20, 30, and 50 epochs with pre-trained weights on ImageNet [24] and we then fine-tuned the network on 400 real sketches, and tested on 100.

Epochs	Synthetic Sketches	Real Sketches
20	0.991	0.968
30	0.995	0.973
50	0.998	0.982

Table 1. Accuracy Results over Synthetic and Real Sketches Datasets

As reported in Table 1, the performances of the convolutional network in distinguishing the structural features of the sketched component achieve very good results with a top 0.982 accuracy over the real sketches set after 50 epochs of training, meaning that the network can effectively recognize the structure of the sketched component, the structure can be parsed directly to code, making this technique promising for real world applications.

In addition, to further understand the quantitative results of our method, in Table 2 we analyzed the performance of the network in recognizing each of the proposed symbols.

Dataset	Default Selected Element	Dropdown Menu	Link	Page Number
Synthetic Sketches	0.995	0.996	0.989	0.992
Real Sketches	0.987	0.991	0.972	0.989

Table 2. Accuracy Results for each symbol

The analysis would motivate us to change some symbols' designs in order to achieve higher accuracy.

As reported in Table 2, the "Page Number" is the symbol most easily recognized by the classifier, while "Dropdown Menu" and "Default Selected Elements" shows comparable results. The least recognized symbol is "Links", probably because in a few samples it overlaps with text. In further works, we may improve its design or position specifications to achieve better results.

5 Conclusion and Future Works

This paper presents a method which can support designers in generating web pages from a sketch, while describing the dynamic behaviors of the pages. Our approach consists of four main parts: a set of symbols to use in sketched web pages; a deep learning architecture for segmentation of the sketched pages into components; a classification algorithm that infers the structural properties of the components and recognizes the symbols that model dynamic behavior; and a parsing algorithm that resembles the information obtained by the network to generate the final code. Among its advantages, it is fully integrated and easily adaptable for different sketches in various domains. To our knowledge, it is the first method that may allow designers to model the dynamic behavior of the sketched directly in sketch-to-code translation algorithms, while using deep learning techniques.

The proposed approach has some limitations that could eventually be addressed in future research. First of all, the structural content features of the components are handcrafted, thus the model cannot generalize out of the sketching specification. This is done to obtain good results due to the complexity of structural specifications in web components. Future work will include the implementation of techniques that allow code generation with language models instead of procedural methods, since language models can generalize out of handcrafted features in this specific task, e.g., as shown by Beltramelli [6]. Secondly, the modeling of dynamic behavior is limited to a subset of the dynamical behavior of a real web application. Future research should focus on enhancing the capabilities of our method to model a wider range of dynamic behaviors. Finally, the depicted method needs to be tested with a diverse set of sketches, hand-drawn symbols, and web elements, as well as to be included in a tool for designers. Such a tool will, then, be evaluated in user studies to assess the usefulness of the overall approach.

To conclude, sketch-to-code translation of user interfaces is closer to being implemented in real-world applications, and our work moves the first steps towards allowing designers to model the dynamic behavior of web interface elements. We do so by leveraging machine learning techniques, to deliver a more integrated approach able to support designers in easing this time-consuming part of their work.

References

[1] Newman, M.W. and J.A. Landay. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In Proceedings of DIS 2000: Designing Interactive Systems. New York, New York. pp. 263-274, August 2000.
[2] Hong, J.I., F.C. Li, J. Lin, and J.A. Landay. End-User Perceptions of Formal and Informal Representations of Web Sites. In Proceedings of Human Factors in Computing Systems: CHI 2001 Extended Abstracts. Seattle, WA. pp. 385- 386, March 31-April 5, 2001.
[3] Virzi, R.A., J.L. Sokolov, and D. Karis. Usability Problem Identification Using Both Low- and High-Fidelity Prototypes. In Proceedings of Human Factors in Computing Systems: CHI '96. Vancouver, BC, Canada. pp. 236-243, April 13–18, 1996.
[4] Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. 2019. Eve: A Sketch-Based Software Prototyping Workbench. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (Glasgow, Scotland UK) (CHI EA '19). Association for Computing Machinery, New York, NY, USA, 1–6.
[5] Fidelity or Low-Fidelity, Paper or Computer? Choosing Attributes when Testing Web Prototypes. Proceedings of the Human Factors and Ergonomics Society Annual Meeting 46, 5 (2002), 661–665.
[6] Tony Beltramelli. 2018. Pix2code: Generating Code from a Graphical User Interface Screenshot. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (Paris, France) (EICS '18). Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages.
[7] Alex Robinson. 2019. Sketch2code: Generating a website from a paper mockup. arXiv:1905.13750 [cs.CV]
[8] Marvel. https://marvelapp.com/. [Accessed 20 08 2022]
[9] Pidoco. https://pidoco.com/en. [Accessed 20 08 2022]
[10] Screen Architect. https://www.screenarchitect.com. [Accessed 20 08 2022]
[11] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse Engineering Mobile Application User Interfaces with REMAUI. In Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (Lincoln, Nebraska) (ASE '15). IEEE Press, 248–259.
[12] Benjamin Wilkins. 2017. Airbnb Sketching Interfaces. https://airbnb.design/sketching-interfaces.
[13] James A. Landay. 1996. SILK: Sketching Interfaces like Krazy. In Conference Companion on Human Factors in Computing Systems (Vancouver, British Columbia, Canada) (CHI '96). Association for Computing Machinery, New York, NY, USA, 398–399.
[14] Hosseini-Khayat, Ali Ghanam, Yaser Park, Shelly Maurer, Frank. (2009). ActiveStory Enhanced: Low-Fidelity Prototyping and Wizard of Oz Usability Testing Tool. Lecture Notes in Business Information Processing. 31. 257-258.
[15] Balsamiq. https://balsamiq.com. [Accessed 20 03 2022]
[16] Lin, James Newman, Mark Hong, Jason Landay, James. (2002). DENIM: An Informal Sketch-based Tool for Early Stage Web Design.
[17] Alex Robinson. 2019. Sketch2code: Generating a website from a paper mockup.

[18] AppScketcher. https://www.uxplaza.com/appsketcher. [Accessed 20 08 2022]
[19] JustInMind. https://www.justinmind.com. [Accessed 20 08 2022]
[20] Appery. https://appery.io. [Accessed 20 08 2022]
[21] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollar. 2020. Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis & Machine Intelligence 42, 02 (feb 2020), 318–327.
[22] Silva, Thiago Hak, Jean-Luc Winckler, Marco Nicolas, Olivier. (2017). A Comparative Study of Milestones for Featuring GUI Prototyping Tools. Journal of Software Engineering and Applications. 10. 564-589. 10.4236/jsea.2017.106031.
[23] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. 2018. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. In The 40th International Conference on Software Engineering, Gothenburg, Sweden. ACM.
[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition. IEEE, 248–255