A Blocks-Based Editor for HTML Code

Saksham Aggarwal
International Institue of Information Technology
Hyderabad, India
saksham.aggarwal@students.iiit.ac.in

David Anthony Bau
Phillips Exeter Academy
Exeter, NH, USA
dbau@exeter.edu

David Bau
Google
Cambridge, MA, USA
davidbau@google.com

Abstract—**This paper presents a block-programming editor for HTML code. The editor provides a block visualization of HTML syntax, allowing students to work in either blocks or text and switch freely. Our editor was created as an extension of Droplet, a dual-mode programming block editing framework that was previously used for JavaScript and CoffeeScript. We describe the process of extending Droplet to apply to HTML. We also discuss an analysis of real-world HTML tags and attributes and propose a palette based on this analysis.**

I. Introduction

Teaching HTML has long been an early step in a programming curriculum. For example Budny, et al [1] in Four Steps to Teaching C Programming, suggest "The layout of a web page allowed us to begin to teach the basic concepts of program layout... We are teaching web page design ... not for the purpose of teaching HTML, but to teach students the concept of writing code." Mahmoud, et al [2] suggest that starting with HTML is a way of teaching "programming for fun" and is a strategy for motivating students.

Nonetheless, for first-time-coder, HTML can be difficult to learn. In a workshop with English students, Mauriello, Pagnucci, and Winner [3] observed "Students are generally not careful and experienced enough in their reading of the codes to find mistakes." HTML guides for non-coding students such as Taylor and Gitsaki [4] suggest simplifying the problem by starting with a small set of about 30 HTML tags to create a basic web page.

Therefore we are interested in finding an alternative to WYSIWYG HTML tools that expose the code, while still simplifying the process of learning to use HTML tags for the first time. In recent years, block programming languages such as Scratch [5] have introduced many students to coding through a visual representation of commands and control flow. Here we investigate whether a similar approach can be effective when used with HTML code.

We aim to meet three goals simultaneously (borrowing terminology from Nielsen's usability heuristics[6]):

- Allow students to create HTML using *recognition rather than recall*, assembling blocks to make pages.

- Permit editing of real-world wepages with blocks or text: *the system must match the real world*.

- Provide students with a *minimalist design* with a clear, useful, realistic, yet minimal set of choices.
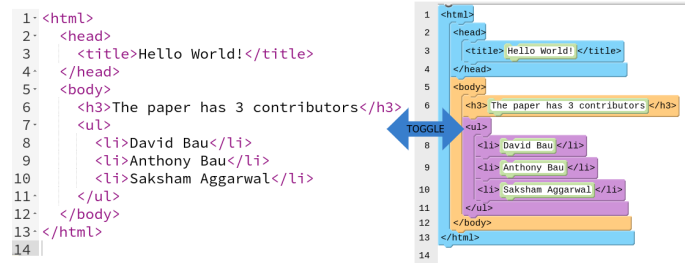

Fig. 1. Our HTML Block editor user experience
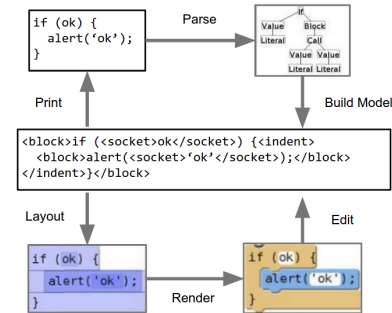Users can switch between blocks and text freely.

Fig. 2. Lifecycle of a Droplet Editing Session

II. Background

A. Droplet's Text-First Approach to Blocks

We built our HTML editor as an extension of Droplet. Droplet [7] is a dual-mode blocks and text editor framework that allows students to work with traditional text code syntax using either drag-and-drop blocks or by typing the text. Students can switch modes at any time.

Droplet's guiding philosophy is that the text, not the blocks, are the primary data. Thus, Droplet programs begin and end their life as text. When Droplet opens a file, the text is parsed into blocks, preserving the original text within block markup. Block editing operations perform splice operations on the text markup stream. At the end of the editing session, the markup is simply discarded and a raw text program is generated again. Figure 2 shows a typical Droplet editing for a JavaScript program, and Figure 1 shows a typical droplet user experience for our HTML editor.

B. Adding A New Language to Droplet

Droplet is designed to be extended to any text language by creating a language adapter.

A Droplet language adapter must parse text to delinate blocks, and it can also provide rules that determine whether blocks are allowed to be dropped into specific locations. Typically the language parser relies on a standard language parser, inserting blocks based on positions of specific types of nodes in an AST. Callback functions are used to determine drag-and-drop permissibility.

New languages in Droplet also need a block palette. The Droplet palette is a reflection of useful choices, not a complete catalog of all possible blocks for a language. Similarly, dropdown choices for for sockets within blocks are similarly curated. Users are always free to enter tags and attributes as free-form text, but a good palette design will allow tall he common cases to be handled with drag-and-drop.

III. Process

A. Adapting A Parser

One of the goals of our HTML editor is to be able to visualize real-world webpages as blocks. This poses a difficulty because browsers are tolerant and many existing webpages are not striclt1y standards-compliant, with mismatched tags, and missing, implicit, or redundant elements.

Droplet's HTML mode adapts the parse5 HTML parser [8], which tolerates syntactically "incorrect" HTML code in the same way browsers do. We modified the parse5 parser for Droplet's purposes to add more detailed location data, so that precise source code locations for tags can be extracted.

The block parsing process for HTML proceeds as the following psuedocode:

1) Parse the text into an DOM tree using parse5.
2) Process the root of the DOM.
3) For each node, check if it is a text node, comment, empty element or an element with content.
4) Mark the range of a node based on its type
   - If a text node, make it editable.
   - If a comment, make the comment editable.
   - If an empty element, mark it as a block and make its attributes editable.
   - If an element with content, mark it as a block, make its attributes editable and add an indent to make space for its children.
   - If an implicit or error tag, do not mark it.
5) Recurse from step (3) for every child.

B. Enforcing Droppability Rules

One major advantage of a block language is that it can guide authoring so that students create standards-compliant code. Droplet's HTML mode therefore enforces droppability rules adapted from the WHATWG HTML specifications [9]. For example, a `<head>` element may only contain metadata elements such as `<title>` and `<meta>`, and a `<table>` element may only contain appropriate table elements such as `<tr>` and `<colgroup>` and `<caption>`.
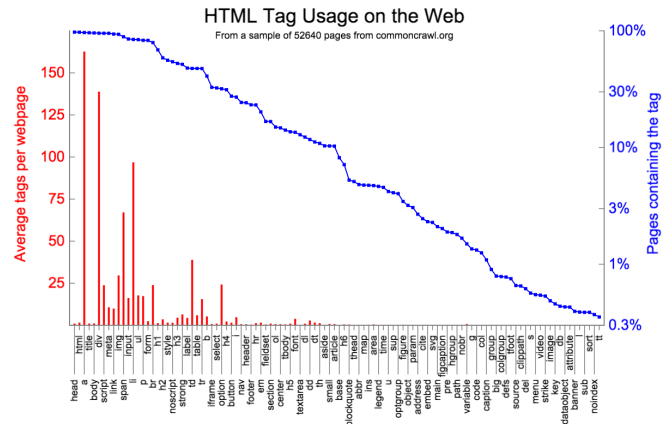

Fig. 3.   HTML Tag Usage on the Web

We have implemented standard the containment rules for 92 tags. When a block is dragged into the document, only legal locations according to these rules are highlighted, and when a drop occurs, only those locations will receive a block.

C. Choosing A Palette

The palette in a block language is important to discovery and self-directed learning, because students can try new commands without having to read documentation. Having a palette that contains useful and rewarding tags in an HTML mode is therefore important.

The WHATWG HTML specifications define over 100 tags. However, most them are not used on a typical webpage. A number of developers online have informally posted HTML cheat sheets with the "most important tags," [13] [14] [15] but these are subjective and often conflict with each other. Because Droplet's philosophy is to be able to interact with real-world code on the Internet, we here determine and recommend a palette based on real-world tag frequencies.

We are not aware of any published statistics on the real-world frequency of HTML tag usage on the web, so we crawled a selection of 52460 webpages using Common Crawl [11] to create real-world frequency data. Our analysis of the top 108 tags is summarized in Figure 3. (Full data from our crawl is available on github [10].) Red bars in that graph represent a count of the number of times each tag was used per page on average in the crawled data set. The blue line, on a logarithmic scale, represents the percentage of documents in the data set that used the tag. In addition, we analyzed the most common tag-attribute pairs. These statistics are summarized in Figure 5. Red bars on that graph represent the percentage of webpages that include a given attribute on a specific tag.

We created the final palette (Figure 4) by choosing the top 40 tags from the above 2 analysis results, adding a few elements to align with courses such as [16] [17] [18], and removing some elements which have a similar behavior as another commonly-used alternative such as `<i>` versus `<em>`.

We also created dropdown lists for each element (Figure 6) to allow stduents to choose between the most common attributes. The student can choose an attribute and enter the
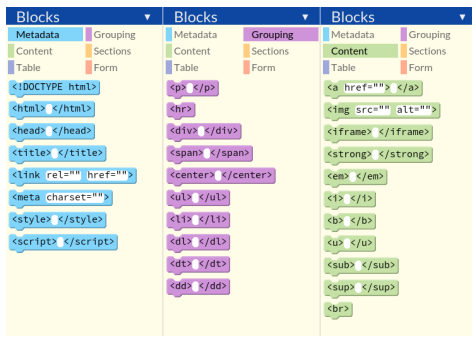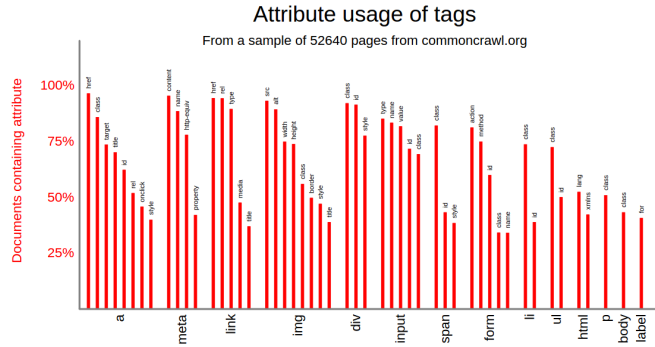
Fig. 4. Three panels from the palette

Fig. 5. Top attributes used with tags

attribute value as text, or just enter the entire attribute/value pair as text.

IV. FUTURE WORK

A. Buttons for adding and removing attributes

In HTML, every tag can accept a variable number of attributes, and the HTML mode would benefit from improved block editor support for variable argument lists. We are building improvements to make it easy to use "add/remove" buttons on a block for adding new attributes.

B. Polymorphic elements

Some elements are used in several distinct ways, and we plan to do further analysis on common tag usage to decide on a few tags to represent in multiple ways on the palette. For example, two ways of using `<a>` are with `href` or with

name; two ways of using `<script>` are with `src` or with inline script.

C. Transparent content model

Some tags such as 'a', 'ins', 'del', 'map' are "transparent" elements, which means, according to the standard [24], their content model is derived from the content model of its parent element. This type of contextual content model is not currently implemented but provided as "flow content" [19]. We plan to add support for a transparent content model.

REFERENCES

[1] Budny, D.; Lund, L.; Vipperman, J.; Patzer, J.L.I.I., "Four steps to teaching C programming," Frontiers in Education, 2002. FIE 2002. 32nd Annual , vol.2, no., pp.F1G-18,F1G-22 vol.2, 2002

[2] Qusay H. Mahmoud, Wlodek Dobosiewicz, and David Swayne. 2004. Redesigning introductory computer programming with HTML, JavaScript, and Java. SIGCSE Bull. 36, 1 (March 2004), 120-124. DOI=10.1145/1028174.971344 http://doi.acm.org/10.1145/1028174.971344

[3] Mauriello, N. Pagnucci, G. and Winner, T. Reading between the Code: The Teaching of HTML and the Displacement of Writing Instruction. Computers and Composition 16, 409-19 (1999)

[4] Taylor, R. and Gitaski, C. Teaching WELL and loving IT. New Perspectives on CALL for Second Language Classrooms, 131-147.

[5] Scratch. https://scratch.mit.edu/

[6] Nielsen, Jakob. Usability engineering. Elsevier, 1994.

[7] Bau, D. A. Droplet, A Blocks-Based Editor for Text Code. Journal of Computer Science in Colleges. 30, 6 (June 2015).

[8] Parse5. https://github.com/inikulin/parse5

[9] HTML living standard. https://html.spec.whatwg.org/

[10] https://github.com/sakagg/HTMLtagsFrequencyAnalysis

[11] Common Crawl. https://commoncrawl.org/

[12] HTML Palette in use on Pencil Code. pencilcode.net/edit/example.html

[13] Webmonkey. HTML Cheat Sheet. http://www.webmonkey.com/2010/02/html_cheatsheet/

[14] A Simple Guide to HTML. HTML Cheat Sheet. http://www.simplehtmlguide.com/cheatsheet.php

[15] Usabilla. An HTML Cheat Sheet That Never Fails. http://blog.usabilla.com/an-html-cheat-sheet-that-never-fails/

[16] Exploring computer Science - pages 105-110 http://www.exploringcs.org/wp-content/uploads/2014/02/ExploringComputerScience-v5.0.pdf

[17] W3Schools HTML starters guide http://www.w3schools.com/html/default.asp

[18] Htmldog beginner tutorial http://htmldog.com/guides/html/beginner/conclusion/

[19] WHATWG flow content list https://developers.whatwg.org/content-models.html#flow-content

[20] WHATWG metadata content list https://developers.whatwg.org/content-models.html#metadata-content

[21] WHATWG phrasing content list https://developers.whatwg.org/content-models.html#phrasing-content

[22] WHATWG interactive content list https://developers.whatwg.org/content-models.html#interactive-content

[23] WHATWG script supporting elements https://developers.whatwg.org/content-models.html#script-supporting-elements

[24] https://html.spec.whatwg.org/multipage/dom.html#transparent-content-models

Fig. 6. Using commonly used attributes in a dropdown