Dual Block/Text Editing in Many Languages

D. Anthony Bau
Phillips Exeter Academy
Exeter, New Hampshire 03833
Email: dbau@exeter.edu

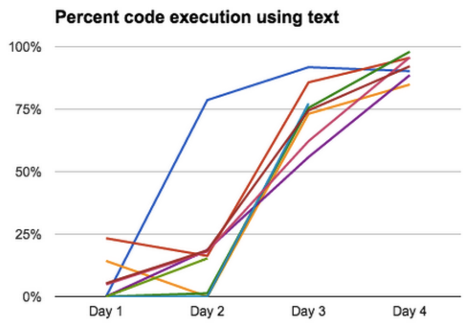


Fig. 1. Previous findings for the transition between blocks and text with Droplet

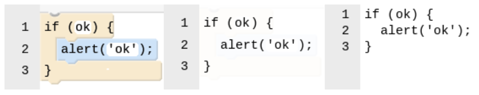
***Abstract*—Droplet is new programming editor that can edit arbitrary code in a blocks-based environment. This paper presents Droplet's techniques for parsing and editing a wide variety of languages, including Python, Java, and C. It presents a general framework for adapting Droplet to a new language, and discusses techniques for doing syntax-checking. It also presents a new Droplet input mode, allowing students to type expressions in sockets and turn them into blocks immediately.**

I. Introduction

In introductory courses, teachers often see a gap between block languages like Scratch and Blockly and text languages. Block languages give a visual representation of a program and help beginners and "at risk" students [3]. However, students encounter "syntax overload" [8] and a lack of confidence [9] when they attempt to transition to text.

Droplet is a new programming editor that edits text programs in a drag-and-drop blocks environment, built to span the gap between blocks to text [1]. Droplet was initially implemented with CoffeeScript and used with Pencilcode, a previously text-based environment, and has been found to help students transition smoothly from blocks to text in CoffeeScript [1]. In classroom experiment, all students started with blocks and the majority of students transitioned voluntarily to text within four days [2] (Fig. 1).

CoffeeScript, however, is a relatively uncommon teaching language. The CSTA 2013 National Secondary School Computer Science Surveys found that the most common text language for high school teaching was Java, followed by BASIC, C++, JavaScript, and Python [6]. A survey by Guo of university courses in 2014 found Python to be the most common teaching language, followed by Java, MATLAB, and C [7]. Extending Droplet to these languages would allow blocks to be easily used in courses that currently use text.



```
if (ok) {
  alert('ok');
}
```

```
if (ok) {
  alert('ok');
}
```

```
if (ok) {
  alert('ok');
}
```

Fig. 3. Frames of Droplet's Block/Text Animation

In this paper presents a framework for quickly extending Droplet to new languages, and applies this framework to create block/text editors for Java, Python, and C.

This paper also introduces Droplet's hybrid input mode, live reparsing, that allows the creation of block expressions in a manner similar to the new Greenfoot 3 frame-based editor [15].

II. Related Work

Tools have been created to help span the gap between blocks and text in individual text languages. BlockEditor is a dual block/text editor that can edit a subset of Java, and has been found to be effective assisting the transition from blocks to Java [10]. Tiled Grace is a block/text editor that can edit Grace programs, and has been found to smooth the transition from blocks to Grace [4]. Cheung, et. al. found that BrickLayer, a text-enhanced block language, helped high schoolers transition from blocks to C++ [11]. As far as the author is aware, Droplet is the first work creating a dual-mode block/text editor for Python.

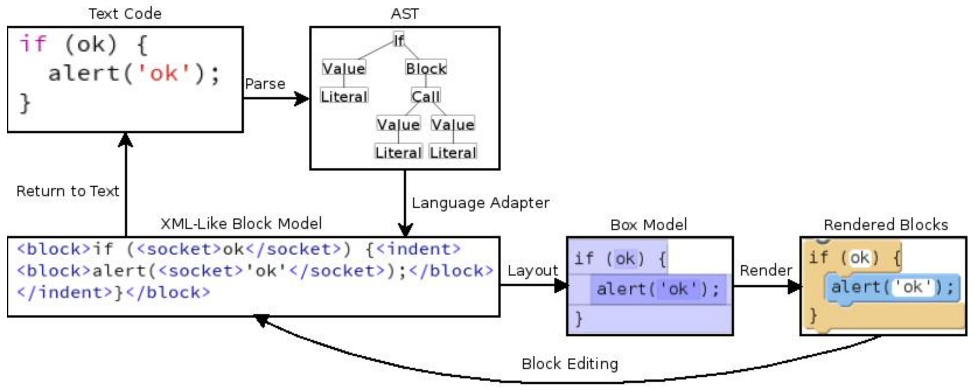
The successful CodeSpells game uses a block language that integrates closely with Java [13]. For embedded systems, Flowcode is a visual program editor that integrates closely with C++, and has a feature can help the user learn C++ alongside their Flowcode program [**?**]. Previous work has been done to extend Droplet to HTML [**?**].

Greenfoot 3 requires users to write text for single-line expressions, to make program composition faster and to give users the experience of writing text while interacting with blocks [15].

III. Background

A. *Droplet's Text-First Approach to Blocks*

Droplet was built to bridge the gap between blocks and text. Droplet's primary 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 program file, it runs the program text through a language adapter and inserts markup which indicates where blocks should go and how they should be rendered. The user interacts with this rendering of the program, performing splice operations on the markup



```
if (ok) {
  alert('ok');
}
```

```
<block>if (<socket>ok</socket>) {<indent>
<block>alert(<socket>'ok'</socket>);</block>
</indent>}</block>
```

Fig. 2. Lifecycle of a Droplet Program

stream. During editing, the language mode may be called back to preserve precedence or dictate droppability rules. 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 session in JavaScript.

Droplet will always lay out the text in the blocks as it appears in the source text. This is both to follow Droplet's text-first philosophy and to facilitate an animation between blocks and text (Fig. 3). This means that Droplet programs are always in a single linear order, unlike Scratch and Blockly. However, Droplet can still take advantage of block languages' two-dimensional composability with "floating stacks" [14].

B. *The Droplet Markup Scheme*

Droplet has three types of markup entities: Blocks, Sockets, and Indents. Blocks are solid-colored, draggable ranges; Sockets are white, typeable, droppable ranges that can contain single Blocks; Indents are ranges that include multiple Blocks and render their children with interlocking tabs. All three node types can have arbitrary "classes" attached to them, which is simply a list of strings that the language mode assigns at parse time and can re-examine later.

Indents consume whitespace at the beginning of each line; this whitespace is removed from the text stream while editing. This is done so that drags can be interpreted as simple splice operations. The whitespace is reinserted using the Indent metadata when the raw text is regenerated. Each Indent has as metadata a prefix to add to each line; if any lines are misformatted and do not line up with the indent, those lines are flagged as exceptions and handled appropriately when the text is regenerated.

IV. A Generalized Language Framework

In our work generalizing Droplet to new languages, we designed a framework for generating Droplet's block-language markup from an arbitrary AST. A new language mode uses a standard parser, like ANTLR (for Java and C++) or Skulpt (for Python), and convert the syntax tree to Droplet's format.

Generating Droplet markup from an abstract syntax tree usually involves a depth-first AST traversal. The Droplet treewalker framework facilitates this. Five types of events can happen when a Droplet treewalker traversal encounters a node:

1) indent
2) paren
3) merge
4) block/socket
5) socket terminal

A **merge** event skips the node and proceeds directly to its children without adding a block boundary. If a node has exactly one child, it is automatically skipped regardless of how the mode was configured. An automatically skipped node, however, is recorded, and if a Block is generated, it is flagged as having the node type of all the skipped nodes. This facilitates automatic rudimentary type-checking, as discussed in the ANTLR adapter section.

A **socket terminal** event makes a typable socket around a terminal token.

A **block/socket** event is the basic Droplet event, and is applicable to most AST nodes. Most nodes should become blocks in the output, so a Block is created. However, a Block cannot immediately have a Block parent (when the Block is removed, it must leave behind a Socket), so a Socket must be inserted directly around the Block.

An **indent** event generates an Indent node in the Droplet markup. Most languages do not have strict whitespace rules, so the indent depth of an Indent can sometimes be ambiguous. Unless other information is available, the indent depth is determined by majority count over all the lines in the range that are not within other Indent children. Indents exclude leading and trailing free text, as this is usually a non-operational marker like "{" or "}" that should properly belong to the indent's parent.

The Droplet language framework configuration for Python, which is only 99 lines, can be found on Github [16].

A. *ANTLR*

ANTLR (ANother Tool for Language Recognition) is a parser generator that can generate parsers in JavaScript. Grammars for ANTLR have been written for Java and C++, along with 63 other langauges including BAISC, C#, Scala, and Haskell [17]. The author created a transformer from ANTLR output to Droplet ASTs, allowing languages to be created directly from ANTLR grammars. An example configuration for Java, which is only 81 lines, can be found on Github [18].

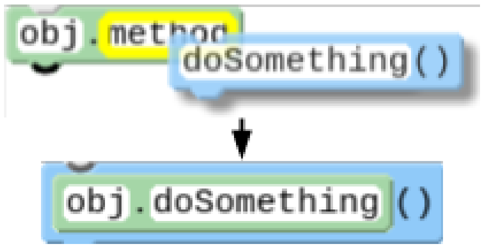


Fig. 5. Example of Droplet's automatic parenthesis insertion

V. Droppability Rules, Precedence, and Parentheses

A. Droppability

One of the primary advantages of block languages is that blocks can only be inserted where they are syntactically correct. In CoffeeScript and JavaScript mode, these rules were implemented by hand. To facilitate easy extension to new languages, Droplet's framework now automatically computes droppability rules from AST node types.

Blocks usually have more than one node type due to the merge treewalk event, and have rendering data associated with the lowest node type. Sockets take on only the highest node type. A Block can be dropped into a Socket if the Socket's node type is any of the Block's types. This strategy ensures that the result will be legal in the grammar. However, it also forbids some combinations that should be legal. For instance, in Figure 9, the ”x” socket expects a node with an ”expression” type. However, because of the way the C grammar is written, the ”2 * 3” block only has types ”additiveExpression” and ”multiplicativeExpression”; Droplet does not know that ”2 * 3” is an ”expression”, so the drop is forbidden.

These problems can usually be resolved by modifying the source grammar; how to generalize droppability rules for an arbitrary grammar is still an open problem.

B. Precedence and Parentheses

In Droplet's hand-coded modes (CoffeeScript and JavaScript), precedence is handled with metadata on Blocks and Sockets. Each Block is assigned a precedence number based on its node type, and each Socket is assigned a precedence number based on its parent's node type. Whenever a drop occurs, Droplet notifies the language mode, which adds or removes parentheses if necessary. The same callback is also used for semicolons in block statements and commas in array literals; the language mode receives the classes that it assigned at parse time, preceding, and succeeding sibling nodes, and adds or removes delimiters as necessary.

In the modes generated from the new Droplet framework, precedence is handled automatically by reparsing. When a block is placed, the parent block is reparsed. If this reparse changes the block structure, it is assumed that order of operations was broken, and parentheses are inserted (Fig. 4). This strategy works correctly in all cases except where the user may want the block structure to deliberately change – for instance, the CoffeeScript drop described in Figure 5.

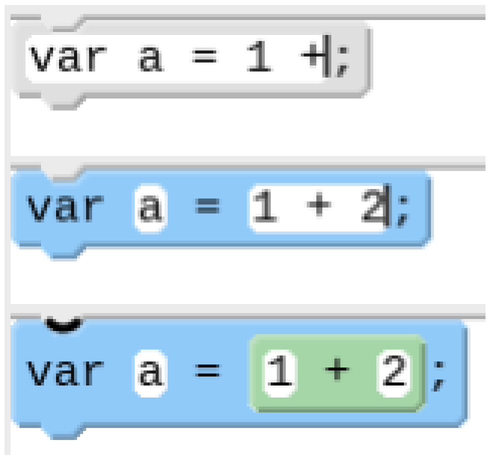


Fig. 6. Example of Droplet's Live Reparsing

VI. Live Reparsing

Droplet's contextual parsing also allows for a dual block/text input mode. When users enter reparseable expressions into a socket, Droplet can reparse these expressions into blocks on-the-fly (Fig. 6). This is similar to Greenfoot 3's frame-based editing, which allows for quicker input of expressions [15].

If a parser, like ANTLR-generated parsers, exposes arbitrary parsing contexts, live reparsing reparses the closest parent using its node type, which will have been stored in the ”classes” flag at parse time.

For parsers that do not expose arbitrary parsing contexts, such as the CoffeeScript or Python parsers, Droplet simulates non-root parsing contexts by assigning a piece of metadata to each Block. This metadata is three strings: a prefix, a suffix, and an indent prefix. When parsing, Droplet concatenates the prefix, input string, and suffix, and parses (fully, so the output is a Droplet markup stream) it at the root context. It then scans forward in the generated markup stream to the end of the prefix. If there is a single Droplet markup entity (Block, Socket, or Indent) that starts at the end of the prefix and ends at the beginning of the suffix, it splices this entity out and uses it as the parse. This process is illustrated in figure 7.

If the language mode fails to assign an appropriate parsing context, Droplet instead attempts to bubble the reparse up the parents of the nodes until parsing succeeds, ultimately attempting to reparse the entire document. If this fails, Droplet informs the user that they have made a syntax error by highlighting the socket in red.

VII. Results

Online demos for Droplet's new Java, Python, and C modes can be found online at [19], [20] and [21] respectively. Droppability rules for these languages occasionally prevent a drop where a drop should be legal, but the language modes are otherwise fully functional.

VIII. Conclusion

Because Droplet edits text files, Droplet can be used in existing text curricula. This has been demonstrated by

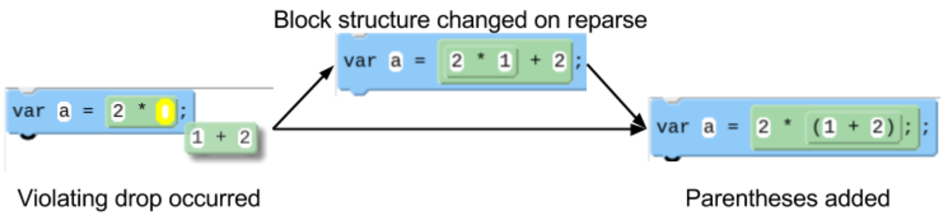
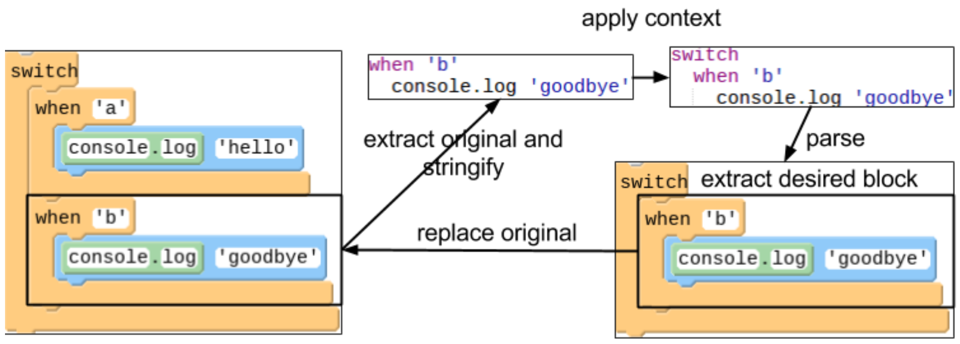


Fig. 4. Example of Droplet's automatic parenthesis insertion



```
when 'b'
  console.log 'goodbye'
```

```
switch
  when 'b'
    console.log 'goodbye'
```

Fig. 7. Example of Droplet's Three-String Contexts

Droplet's CoffeeScript mode, which was used in Pencil Code-with a previously text-based curriculum. By supporting more common teaching languages, Droplet can bring the benefits of block languages to more curricula. In the future, Droplet could be extended using the presented framework to support BASIC and MATLAB.

Droplet's hybrid input mode can also help in the transition between blocks and text, allowing students to type some text while still looking at and working with blocks. In future work, this mode could be extended to allow text editing of blocks that have already been parsed.

References

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

[2] David Bau, D. Anthony Bau, Mathew Dawson, and C. Sydney Pickens. 2015. Pencil code: block code for a text world. In Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15). ACM, New York, NY, USA, 445-448. DOI=10.1145/2771839.2771875 http://doi.acm.org/10.1145/2771839.2771875

[3] Moskal B, Lurie D, and Cooper S. Evaluating the Effectiveness of a New Instructional Approach. Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education.

[4] Homer M. Combining Tiled and Textual Views of Code. in 2nd IEEE Working Conference on Software Visualization (Victoria, CA, 2014), IEEE, 1-4.

[5] Bau D, Bau D. A. A Preview of Pencil Code. Proceedings of the 2014 ACM Conference on Systems, Programming and Applications. (In press.)

[6] Computer Science Teachers Association. CSTA National Secondary School Computer Science Survey (2013). http://csta.acm.org/Research/sub/Projects/ResearchFiles/CSTASurvey13Results.pdf. Accessed Mar 14, 2015.

[7] Guo, P. Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. Communications of the ACM blog. http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext. Accessed Mar 14, 2015.

[8] Powers K, Ecott S, Hirshfield L M. 2007. Through the looking glass: teaching CS0 with Alice.SIGCSE Bull. 39, 1 (March 2007), 213-217.

[9] Lewis C M. 2010. How programming environment shapes perception, learning and goals: logo vs. scratch. In Proceedings of the 41st ACM technical symposium on Computer science education.

[10] Yoshiaki Matsuzawa, Takashi Ohata, Manabu Sugiura, and Sanshiro Sakai. 2015. Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). ACM, New York, NY, USA, 185-190. DOI=10.1145/2676723.2677230 http://doi.acm.org/10.1145/2676723.2677230

[11] Joey C.Y. Cheung, Grace Ngai, Stephen C.F. Chan, and Winnie W.Y. Lau. 2009. Filling the gap in programming instruction: a text-enhanced graphical programming environment for junior high students. In Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE '09). ACM, New York, NY, USA, 276-280. DOI=10.1145/1508865.1508968 http://doi.acm.org/10.1145/1508865.1508968

[12] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated transfer: Alice 3 to Java. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12). ACM, New York, NY, USA, 141-146. DOI=10.1145/2157136.2157180 http://doi.acm.org/10.1145/2157136.2157180

[13] Sarah Esper, Stephen R. Foster, William G. Griswold, Carlos Herrera, and Wyatt Snyder. 2014. CodeSpells: bridging educational language features with industry-standard languages. In Proceedings of the 14th Koli Calling International Conference on Computing Education Research (Koli Calling '14). ACM, New York, NY, USA, 05-14. DOI=10.1145/2674683.2674684 http://doi.acm.org/10.1145/2674683.2674684

[14] Bau, D.A. Integrating Droplet into Applab: Improving the Usability of a Blocks/Text Editor. Proceedings of the 2015 VLHCC (to appear).

[15] Frame-Based Editing. http://www.greenfoot.org/frames/.

[16] http://github.com/dabbler0/droplet/tree/master/src/languages/python.coffee

[17] https://github.com/antlr/grammars-v4

[18] http://github.com/dabbler0/droplet/tree/master/src/languages/java.coffee

[19] http://experiment.pencilcode.net/anthony/droplet/example/example-java.html

[20] http://experiment.pencilcode.net/anthony/droplet/example/example-python.html

[21] http://experiment.pencilcode.net/anthony/droplet/example/example-c.html