Exposé

ClojureBlocks: Ein visueller Editor für Clojure

Jakob Handke

April 2023

Inhaltsverzeichnis

Problemstellung

Programmieren lernen ist schwer. Dabei sind Probleme korrekt zu formalisieren und die Anforderungen an eine potentielle Lösung zu formulieren neben einer neuen Sprache mit ungewohnter Syntax und Semantik zu lernen nur Beispiele für Schwierigkeiten, die beim Einstieg ins Programmieren auftreten können.

Die meisten Studierenden lernen eine hauptsächlich imperative Programmiersprache wie Java oder Python im ersten Programmierkurs, die wenigsten lernen eine funktionale Sprache wie Haskell oder Scheme [Sie+21]. Wenn nach Lehrplan, Prüfungsordnung oder persönlichem Interesse in einem folgenden Kurs eine funktionale Programmiersprache bzw. das Paradigma der funktionalen Programmierung erlernt werden soll, ist der Lernaufwand vergleichbar mit dem erneuten lernen der Programmierung. So gibt es bei der funktionalen Programmierung grundlegende konzeptuelle Unterschiede zur imperativen Programmierung wie Funktionen als First-Class-Objekte, persistente Datenstrukturen, Funktionen höherer Ordnung und pure Funktionen. Bei einigen funktionalen Programmiersprachen wie Haskell gibt es dazu eine unübliche Syntax, die dem Lambda-Kalkül nachempfunden ist.

Visuelles Programmieren

„Visual programming is programming in which more than one dimension is used to convey semantics“ schreibt Burnett [Bur99]. Der Zeilenumbruch im klassischen, textbasierten Programmieren, also die zweite Dimension neben dem Text in einer Zeile, wird dabei als nicht bedeutungstragend betrachtet. Konkret lässt sich visuelle Programmierung dadurch beschreiben, dass Programme nicht durch Texteingabe, sondern durch eine visuelle Beschreibung wie Zeichnen von Datenzusammenhängen oder durch Zusammensetzen von Bausteinen oder ähnlichem erstellt werden.

Kaučič; Asič [KA11] und Poole [Poo15] zeigen, dass visuelle Programmierumgebungen zu Beginn des Lernprozesses die Probleme der Programmiersprache bzw. des Programmierens ausblenden und den Fokus von syntaktisch korrektem Code hin zu den grundlegenden Schwierigkeiten der Programmierung lenken können.

Das wohl prominenteste Beispiel von visuellen Programmierumgebungen ist Scratch. Scratch ist eine Browseranwendung, in der niederschwellig mit Hilfe von aneinanderreihbaren Bausteinen grundsätzliche Konzepte des Programmierens erlernt werden können. Es gibt einfache Scratch-Projekte, die einem Hello-World-Projekt der klassischen Programmierung entsprechen, aber auch komplexere, in denen als Übung beispielsweise eine Figur mit Codeanweisungen in ein Ziel bewegt werden muss. Projekte können von allen Menschen für den privaten Gebrauch oder für die Öffentlichkeit erstellt und auf der Plattform bereitgestellt werden [Mal+10].

Ein weiteres Beispiel für existierende visuelle Programmierumgebungen ist LabVIEW. Mit LabVIEW können mittels grafischem Interface Programme für Laborequipment wie Messgeräte oder Sensoren erstellt werden. Mit Hilfe sogenannter Virtual Instruments können gemessene oder berechnete Werte dargestellt werden und Automationen anhand

von Werten erstellt werden [Kod20]. Beispielsweise kann automatisch eine E-Mail verschickt werden, wenn ein angeschlossener Temperatursensor eine Temperatur misst, die einen Schwellwert überschreitet.

Clojure

Clojure ist ein funktionaler Lisp-Dialekt von Rich Hickey aus dem Jahr 2007. Die Sprache wird kompiliert und auf der JVM ausgeführt und ist dadurch für fast alle Plattformen verfügbar. Durch die beidseitige Interoperabilität mit Java können Komponenten in Clojure in bestehende Java-Anwendungen eingebunden werden [Hic20]. Clojure bringt unveränderbare, persistente Datenstrukturen und Unterstützung für nebenläufige Programmierung mittels transaktionalem Speicher und asynchronen Agenten mit.

ClojureScript ist ein Compiler für eine Sprache, die Clojure sehr ähnlich ist, nach JavaScript. Mit ClojureScript können auch Webanwendungen funktional erstellt werden und die meisten Funktionen und Vorteile von Clojure genutzt werden [McG11].

Motivation

Auch der Autor hat im Rahmen seines Informatikstudiums im Wintersemester 2022/2023 im Modul „Einführung in die funktionale Programmierung“ an der HHU Düsseldorf grundlegende Konzepte und Kompetenzen der funktionalen Programmierung anhand von Clojure erlernt. Die oben genannten Schwierigkeiten bei der Umstellung von imperativer zu funktionaler Programmierung sowie Probleme bei der korrekten Klammerung der Ausdrücke in Clojure traten bei ihm und Kommiliton:innen auf.

Zur Unterstützung der Lehre in diesem oder anderen Modulen soll ein visueller Editor für Clojure erstellt werden. Dieser soll die Verständnishürden bei der typischen Lisp-Syntax und den Konzepten der funktionalen Programmierung mindern. Weiter sollen Higher-Order-Funktionen visuell dargestellt werden, um einzelne Funktionsaufrufe greifbar zu machen.

Blockly

Blockly[1] ist eine JavaScript-Bibliothek von Google, die einen visuellen Editor für Codeblöcke in Browseranwendungen einbindet. Sie umfasst vorgefertigte Blöcke für imperative Sprachen und kann Code für Dart, JavaScript, Lua, PHP und Python generieren. Um Blockly mit weiteren Sprachen oder Konzepten zu verwenden, können Blöcke mit Eingabefeldern oder Verbindungsmöglichkeiten zu anderen Blöcken definiert werden. Außerdem können Codegeneratoren eingebunden werden, die anhand eines Blocks, den eingegebenen Werten und den Verbindungen zu anderen Blöcken Code generieren.

[1] `https://developers.google.com/blockly`

ClojureBlocks: Idee und MWE

Das Ziel dieser Arbeit ist ein visueller Editor für Clojure. Als Basis dazu soll Googles Blockly verwendet werden, da es leicht erweiterbar ist und den Aufwand der Erstellung eines eigenen Editors vermeidet. Für alle Datenstrukturen, Typen, special forms und einige Funktionen der Standardbibliothek von Clojure sollen Blöcke erstellt werden, die aneinandergereiht bzw. ineinander verschachtelt werden können. Blöcke sollen automatisch Code generieren, der an geeigneter Stelle angezeigt wird. Denkbar ist dafür ein Textfeld, in dem der gesamte erzeugte Code angezeigt wird. Außerdem kann die Anzeige des generierten Codes pro Block erfolgen. Beim Bau des Prototypen wurde allerdings festgestellt, dass dessen Anzeige nicht innerhalb des jeweiligen Blocks erfolgen kann. Der generierte Code soll mittels einem eingebundenen Clojure(Script)-Evaluator evaluiert werden und das Ergebnis jeder Evaluation im Format `(+ 3 4) => 7` angezeigt werden.

Auch soll die Funktionalität von Higher-Order-Funktionen wie zum Beispiel `map` visualisiert werden und beispielhaft ein oder mehrere Funktionsaufrufe auf Elemente aus der Collection gezeigt werden. Wie Higher-Order-Funktionen, die die Länge einer Collection ändern, z.B. `filter`, oder nur einen Wert zurückgeben, z.B. `reduce`, visualisiert werden sollen, ist noch offen.

Mit Hilfe der Tooltip- und Help-URL-Funtionen von Blockly sollen für vorgefertigte Funktionsblöcke Hilfestellungen gegeben werden. Als Help-URL kann beispielsweise immer der passende Link zu `clojuredocs.org` angegeben werden.

Ein Minimalbeispiel ist zu finden unter `codeberg.org/jhandke/ClojureBlocks-cljs`.

Vorläufige Gliederung

1. Einleitung
   - 1.1 Motivation
   - 1.2 Verwandte Arbeiten / Hintergrund
   - 1.3 Problemstellung
2. ClojureBlocks
   - 2.1 Plattform: JavaScript vs. ClojureScript
   - 2.2 Architektur
   - 2.3 Komponenten: Probleme und Entscheidungen
3. Evaluation
   - 3.1 Erreichte Ziele
   - 3.2 Großes Projekt in ClojureBlocks
   - 3.3 Ausblick

Variationen

Als zusätzliche Funktionalität kann die Formatierung des generierten Codes sinnvoll sein, da ClojureBlocks gerade zu Beginn des Erlernens von Clojure benutzt werden soll und die gängigen Entwicklungsumgebungen für Clojure eingegebenen Code ebenfalls automatisch formatieren.

Außerdem kann der Im- und Export des Blockly-Arbeitsbereichs implementiert werden, um Beispielprojekte oder Vorlagen zu sichern und wiederherzustellen. Darauf aufbauend kann auch die Generierung von Blöcken anhand von Clojure-Code eingebaut werden. Langfristig könnte die Anzeige des Codes in ClojureBlocks durch ein bearbeitbares Textfeld ersetzt werden, wobei automatisch die passenden Blöcke in Blockly generiert und anzeigt werden sollen. Dazu kann der Clojure-Parser parcera[2] hilfreich sein.

[2] `https://github.com/carocad/parcera`

Literatur

[Bur99] BURNETT, Margaret M. Visual Programming. In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, Ltd, 1999. ISBN 9780471346081. Abger. unter DOI: https://doi.org/10.1002/047134608X.W1707.

[Hic20] HICKEY, Rich. A History of Clojure. *Proc. ACM Program. Lang.* 2020, Jg. 4, Nr. HOPL. Abger. unter DOI: 10.1145/3386321.

[KA11] KAUČIČ, B.; ASIČ, T. Improving introductory programming with Scratch? In: *2011 Proceedings of the 34th International Convention MIPRO*. 2011, S. 1095–1100.

[Kod20] KODOSKY, Jeffrey. LabVIEW. 2020, Jg. 4, Nr. HOPL. Abger. unter DOI: 10.1145/3386328.

[Mal+10] MALONEY, John; RESNICK, Mitchel; RUSK, Natalie; SILVERMAN, Brian; EASTMOND, Evelyn. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 2010, Jg. 10, Nr. 4. Abger. unter DOI: 10.1145/1868358.1868363.

[McG11] MCGRANAGHAN, Mark. ClojureScript: Functional Programming for JavaScript Platforms. *IEEE Internet Computing*. 2011, Jg. 15, Nr. 6, S. 97–102. Abger. unter DOI: 10.1109/MIC.2011.148.

[Poo15] POOLE, Matthew. Design of a blocks-based environment for introductory programming in Python. In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. 2015, S. 31–34. Abger. unter DOI: 10.1109/BLOCKS.2015.7368996.

[Sie+21] SIEGFRIED, Robert M.; HERBERT-BERGER, Katherine G.; LEUNE, Kees; SIEGFRIED, Jason P. Trends Of Commonly Used Programming Languages in CS1 And CS2 Learning. In: *2021 16th International Conference on Computer Science & Education (ICCSE)*. 2021, S. 407–412. Abger. unter DOI: 10.1109/ICCSE51940.2021.9569444.