Efficiency of Projectional Editing (Extended Abstract)

Thorsten Berger[1] Markus Voelter[2] Hans Peter Jensen[3] Taweesap Dangprasert[3] Janet Siegmund[4]

Abstract: Projectional editors are editors where a user's editing actions directly change the abstract syntax tree without using a parser. For programming, they promise essentially unrestricted language composition and flexible notations. For instance, graphical and textual domain-specific languages can be easily embedded into source code, avoiding intricate parser integration. Yet, despite these benefits, programming still mainly relies on editing textual code, where projectional editors imply a very different experience, often seen as the main adoption challenge. We describe an experiment [Be16] on the efficiency of code editing in a projectional editor conducted with industrial and student developers.

Keywords: projectional editing; language workbench; experiment

Projectional editor describes a type of editor where users work on a projection of a program's abstract syntax tree (AST) and directly change it with their editing gestures. This concept is different from parser-based editing, where users change the concrete syntax, and a parser then constructs the AST. Projectional editing, also known as structured editing or syntax-directed editing, is not a new idea; early references go back to the 1980s and include the Incremental Programming Environment, GANDALF, and the Synthesizer Generator. More contemporary incarnations are Intentional Programming, the Whole Platform, Más, Onion, and Jetbrains Meta Programming System (MPS). Most projectional editors are used in language workbenches—tools for developing and composing languages.

Projectional editors have two main advantages resulting from the absence of parsing. First, they support notations that cannot easily be parsed, such as tables, diagrams or mathematical formulas—each of which can be mixed with the others and with textual notations. Second, they support various ways of language composition, typically including modular language extension as well as embedding unrelated languages into a host language. Projectional editors can deal with mixed-language code while retaining awareness of the code structure (avoiding syntactic ambiguities), which is much harder to achieve with parser-based tools.

These benefits come at a cost. Even though, projectional editors support a wide range of non-textual notations, a significant share of any program, such as expressions and statements, will be expressed textually. For textual notations, however, projectional editors imply a very different—typically perceived as worse—editing experience compared to

[1] Chalmers | University of Gothenburg, Sweden, thorsten.berger@chalmers.se
[2] independent / itemis, Germany, voelter@acm.org
[3] ITU Copenhagen, Denmark, {hpgurre|taweesap}@gmail.com
[4] University of Passau, Germany, janet.siegmund@uni-passau.de

textual (parser-based) editors, often seen as the main challenge prohibiting their widespread adoption. The early projectional editors from the 1980s did very little to address this issue, ultimately limiting their adoption. Contemporary tools, such as MPS, have significantly improved usability, but inherited the bad reputation.

In our main publication [Be16], we present an experiment of code-editing activities in a projectional editor, conducted with 19 graduate computer-science students and industrial developers. We investigate the effects of projectional editing on editing efficiency, editing strategies, and types and frequencies of errors made—each of which we also compare to conventional, parser-based editing. We design the experiment based on a survey [Vo14] we conducted with industrial developers familiar with projectional editing. The instrument for our experiment is JetBrains MPS, since (i) it is the most widely used projectional editor today, (ii) it improved significantly over the tools from the 1980s, and (iii) it is open-source software, which fosters the replicability of our results.

Our results show that efficiency with projectional editing can be quickly achieved for basic code-editing activities. More experience does not lead to significantly better results. In contrast, advanced editing (e.g., larger code modifications or refactorings) requires significantly more experience and understanding of the underlying concepts (in particular, the AST structure). Then, however, experienced developers can outperform beginners with a projectional editor and even the participants using the parser-based editor. The projectional editor also fosters fewer errors (mistakes) and different editing strategies (e.g., increased use of operations that work well on ASTs).

Based on our results, we conceived techniques to further improve the editing experience. We presented Grammar Cells [Vo16], which support creating consistent language-editing experiences, with a focus on supporting the editing of very hierarchical program constructs, such as expressions. Grammar cells create a consistent editing experience that increasingly resembles linear (textual) editing, further reducing the need for understanding the underlying AST. In future work we plan to systematically study the benefits of arbitrary language composition (i.e., language embedding) and flexible notations (graphical and textual).

Literatur

[Be16] Berger, T.; Voelter, M.; Jensen, H. P.; Dangprasert, T.; Siegmund, J.: Efficiency of Projectional Editing: A Controlled Experiment. In: 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE). 2016.

[Vo14] Voelter, M.; Siegmund, J.; Berger, T.; Kolb, B.: Towards User-Friendly Projectional Editors. In: 7th International Conference on Software Language Engineering (SLE). 2014.

[Vo16] Voelter, M.; Szabo, T.; Lisson, S.; Kolb, B.; Erdweg, S.; Berger, T.: Efficient Development of Consistent Projectional Editors using Grammar Cells. In: 9th International Conference on Software Language Engineering (SLE). 2016.