Forth Projectional Editing

Ulrich Hoffmann <uh@fh-wedel.de>

Abstract

Projectional editing is an alternative way to handle programs and data. Instead of starting with text based source code it is centered around internal program/data structures and so called projections create editable representations that allow to modify the internal structures. In the Forth context memory seems to be the appropriate internal data structure. Different editors interpret memory content in specific ways and allow the user to modify it in an appropriate fashion. A hex and a stack editor are described and other editors are proposed. The idea of Forth projectional editing gives a general view to program and data handling that allows to classify techniques used in different Forth systems.

1 Introduction

Projectional editing [1] is an approach to edit programs in a programming language that does not rely on manipulating source code and then scan, parse, translate it to object code. Instead a projectional editor presents a pleasantly *editable representation* of some internal structure (often an abstract syntax tree). Editing this representation results in appropriate modification of the internal data structure, see figure 1. Instead of source code, the internal *abstract representation* is the original artifact and all other representations are produced from it: To persist a storage representation, to run an executable representation is generated. For editing appropriate editable representations are created.

The process of building editable representations from internal data is called *projection* as it might only extract some important aspects of the internal data structure while leaving others untouched. Different projections for the same structure can exist, allowing to edit it in different ways. As an example, a decision table could be edited in a textual representation as an array initialization or it could be presented to the user in tabular form visualizing a decision table similar to those used in a design document.

Projectional editing can be used with great benefit when using domain specific languages

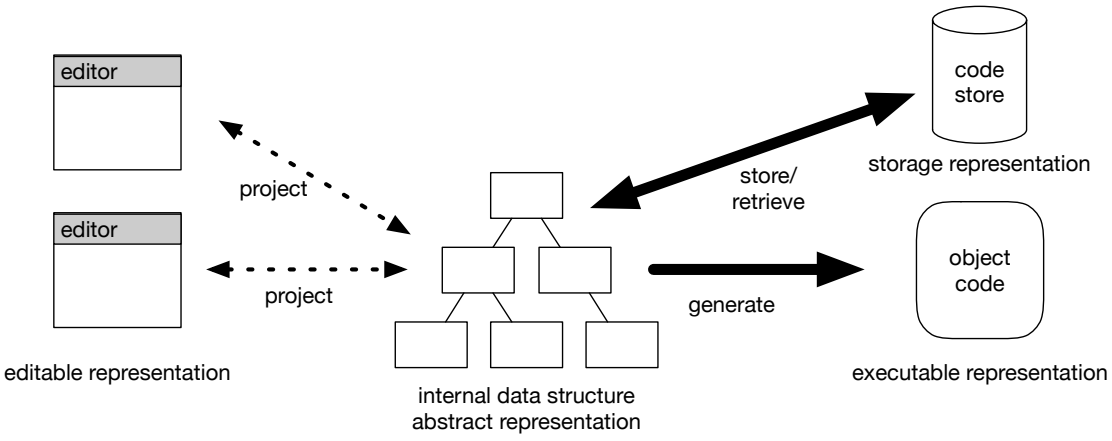

Figure 1: Projectional Editing

(DSLs). They often deal with specific aspects of a problem domain that might have special visual representations. Projectional editing can provide familiar visualization for (parts of) the DSL.

2 Forth Projectional Editing

Forth is especially strong when creating DSLs. Its approach however is not the traditional one building an abstract syntax tree for the internal representation of programs and generate code from this. Instead code generation — even native code generation — typically takes place in a single pass directly starting from Forth source code. This is to support the Forth's interactive nature.

In order to apply projectional editing ideas to Forth a suitable internal representation has to be identified. As Forth is strongly memory oriented (`@ ! , ALLOT MOVE` ...) it seems to be reasonable to use memory as the internal data structure and start different projections from there. Forth programmers are used to take some memory area, represent their data structures in this memory and later on interpret (project) the memory area in this specific way by using only suitable operators on it.[1]

The classical Forth block editor already interprets memory in a specific way: `BLOCK` returns a memory address of a typically 1 KB large memory area that is interpreted as 16 lines of 64 characters without line breaks. Forth programmers edit their source code in these blocks using the block editor and the system interprets and compiles that program by means of the `LOAD` operator. (The block/buffer subsystem transparently handles storing and retrieving blocks.)

2.1 Hex editor

The first projection to look at is a very general one that interprets data in memory as just bytes. This leads to a hex editor. This editor is similar to the `DUMP` utility that can be found in many systems but in addition memory is not just displayed but the hex dump becomes editable. The hex editor and dump utility have the same interface `( c-addr u -- )` and the editor is just a Forth word that can be invoked both interactively and also from within a running program. It returns to the calling word when the editor is exited. Figure 2 shows a sample hex editor session: The editor view is split into two main parts: the hex dump and the character dump. The TAB key switches between the two. The cursor is initially placed in the hex dump and can move by means of the cursor keys. Appropriate data can be entered and directly modifies the represented memory.

A binary editor for Forth blocks can simply be defined as

```
: hedit ( u -- )
    block 1024 hex-edit update ;
```

[1] Type safety is assured by the programmer not the compiler.

```
$ sf hexedit.fs

( hex-editor loaded. Usage: c-addr u HEX-EDIT  ) ok

Create conference  'E' c,  'u' c,  'r' c,  'o' c,  'F' c,  'o' c,  'r' c,  't' c,  'h' c,
conference 30 hex-edit

00003CB44  45 75 72 6F 46 6F 72 74 68 08 68 65 78 2D 65 64   EuroForth.hex-ed
00003CB54  69 74 63 65 2A 00 0F 00 4F 14 00 00 2A 00         itce*...O...*.
```

Figure 2: A sample hex-editor session

If a Forth system implements its stacks in main memory then the stacks also become editable using the hex editor, e.g.:

```
10 20 30 40  sp@ 4 cells hex-edit
```

will start the hex editor on the top most 4 stack items:

```
0BFFFFA80 28 00 00 00 1E 00 00 00 14
00 00 00 0A 00 00 00 (.............
```

The exact layout of the stack in memory is of course system specific.

For this however a different projection of memory might be more appropriate. This leads to the stack editor.

2.2 Stack Editor

The stack editor (sample session in figure 3) displays an editable stack representation with each stack item on a line of its own and allows to interactively modify the stack. Each item ist shown in its character, unsigned hexadecimal, unsigned decimal and signed decimal representation. The stack editor items can be cut/copied and pasted (Ctrl-X, -C, -V) or replaced by items that are the result of a Forth fragment entered on the items line.

Like the hex editor also the stack editor is just a word that can be invoked when appropriate and resumes execution of the caller when exited. So it can be inserted in source where appropriate and be used as an interactive alternative for `.S` debugging.

2.3 Other editors

Adopting the idea of Forth Projectional Editing (i.e. projections from memory to editable representations) in other areas would enable a large selection of possible editors. Examples might be:

- **A Variable Editor**
  This could represent a `VARIABLE` defined word similar to the stack editor in different ways and interactively allow for appropriate changes. Depending on the programmer intended variable type different representations might be reasonable. For example a flag might be shown as a toggle that can be flipped or enumeration data could show and allow to select one of the possible values.
- **A User Area Editor**
  In a multi tasking system the user area is a collection of task specific variables. The idea of the variable editor could be extended to editing the entire user area along with the user variable name and its content.
- **A Structure Editor**
  While a variable editor would allow for editing just a single cell, a structure editor could project an editable representation of an entire structure defined by `BEGIN-STRUCTURE`, `FIELD:`, etc. By introspection it could display the field names and provide appropriate variable editors for each of the fields in the structure.

```
> gforth stackedit.fs
10 20 30 40 50 -1 stack-edit

  0: '?' $FFFFFFFFFFFFFFFF  #18446744073709551615                  -1
  1: '2'               $32                    #50                  50
  2: '('               $28                    #40                  40  42
  3: '.'              $1E                     #30                  30
  4: '.'              $14                     #20                  20
  5: '.'               $A                     #10                  10

up/down: select line   DEL   Ctrl-X, -C , -V    Forth words leaving one item
```

Figure 3: A sample stack-editor session

- **A Wordlist Editor or Dictionary Editor**
  Forth systems typically use a system specific way to organize their dictionary. A projectional editor for the dictionary or a single word list would allow to manipulate the dictionary, i.e. change the order of definitions, adapt spelling or word names, change immediacy of words, possible removing definitions, and others. This way the dictionary becomes the central internal data structure. Also the search-order could be subject of a Search Order Editor.

- **A Word Definition Editor**
  A Forth decompiler typically recreates Forth source code from the memory representation of a definition in the Forth dictionary. A Word Definition Editor could based on a decompiler project a definition in dictionary to an editable representation in source code (tokens or text) and so allow for changing definitions directly in the dictionary.

- **A Screen Editor with line and screen terminators**
  As mentioned before the traditional way to represent Forth source code in memory is 16 by 64 characters in blocks. Source code could be represented in different ways with handling of line terminators (and possible screen terminators) and a screen editor would perform the appropriate projection to user editable source code.

Editors for other data structures seem very well to be possible. The main idea here is to see all of theses editors as a projection from their memory representation to an appropriate editable representation.

3 Related work

Since Martin Fowler's blog article [1] in 2008 some systems have been developed around projectional editing for DSLs. Most prominent is Jetbrain's Meta Programming Systems (MPS) [2] that allows for defining domain specific languages along with appropriate projectional editors. It also supports projectional editing for (parts of) contemporary languages (Java, Javascript, XML, C, ...).

In Forth context the Jupiter Ace [3] home computer includes a Word Definition Editor and follows the *code is the source* paradigm.

The ForthOS [4] system uses 4 KB screens with a source representation of 80 x 25. The Enth [5] system has a screen editor (CodeEd) that handles line terminated source code in fixed sized 1 KB blocks.

HolonForth [6] stores word definitions along with meta data in a database and includes an integrated editing environment that structures projects in a hierarchical way. For editing source code is projected to a full screen editor. Machine code is generated from the internal data base representation.

ColorForth tokenizes word names on entry to 32 bit items and stores these in screens. The colorForth editor generates an editable representation of this token sequence and allow for interactive manipulation of the tokenized source code. The colorForth compiler used the token sequence to generate machine code.

4 Conclusion and future work

Projectional editing gives a different view of programming language editing based on internal abstract representations. It opens additional possibilities for handling programs.

Although Forth systems have never explicitly used projectional editing its ideas are well present in several Forth systems and the idea of projecting memory to specific editors has many interesting applications that complement Forths interactive nature.

Hex- and Stack editors have been implemented as Forth-200x standard programs. Other edi-

tors as mentioned in section 2.3 seem not to be more difficult to implement but some of them probably need system specific details. Among the proposed editors a Structure Editor seems to be the most useful. One can also envision mixed editors that use different editors for different parts of memory.
Is the map the territory? You decide.

> Forth is stacks, words, and blocks; start there.
>
> Jeff Fox [8]

References

[1] *ProjectionalEditing*, M. Fowler, `martinfowler.com/bliki/ProjectionalEditing.html`, 2008
[2] *Meta Programming System*, Jetbrains, `jetbrains.com/mps`
[3] *Jupiter Ace*, Wikipedia, `en.wikipedia.org/wiki/Jupiter_Ace`
[4] *ForthOS*, Wikipedia, `sources.vsta.org/forthos/`
[5] *Enth Flux aha colorForth*, Sean Pringle, `www.ultratechnology.com/enthflux.htm`, 2000
[6] *Holonforth*, Wolf Wejgaard, `holonforth.com`
[7] *ColorForth*, Charles Moore, `colorforth.github.io`
[8] *Forth is stacks, words, blocks*, Jeff Fox, `www.ultratechnology.com/forth2.htm`