CafePie: 図形の組合せを用いたCafeOBJの視覚化

CafePie: CafeOBJ Visualization by using a Combination of Diagrams

小川 徹[†] 田中 二郎[††]
Tohru OGAWA Jiro TANAKA

†筑波大学 博士課程 工学研究科
Doctoral Program in Engineering, University of Tsukuba
††筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

概 要

CafePieは、代数的仕様記述言語CafeOBJのためのビジュアルプログラミングシステムである。この実行系である項書換えは、システムにより与えられた表現を使って視覚化される。その１つ１つの図形は単純で何を意味しているのか分かり難い。我々は、図・絵・イメージなどの図形要素を組合せて、新たに視覚化表現を編集する枠組を提供した。ユーザにとってより分かりやすい表現で項を視覚化されるだけでなく、その表現を使用してプログラム編集・実行も従来と同じように行うことが可能である。

1 はじめに

ビジュアルプログラミング(VP) [1]とは、従来のような文字列や記号を中心としたインタフェースに代わって、アイコンやアニメーションという人間がより理解しやすいビジュアルな表現を用いて人間とコンピュータとの相互作用を行う技術のことであり、例えばPP [2]などがそのシステム例である。

VPが従来のテキストベースによるプログラミングと大きく異なることは、プログラムを表現する際に視覚的な特徴を用いる点にある。視覚的特徴をうまくプログラム表現に反映させることで、プログラムの可視性が向上し、理解がしやすくなる。しかし、ビジュアルプログラミングシステム(VPS)により視覚化されたプログラムが、それ以上表現を変更できなかったり、できたとしてもある制限の元で変更することしかできないのは問題である。視覚的な意味を与えられるというVPの特徴を十分に生かしきれていないためである。

そこで、我々は、システムにより与えられた視覚化表現をユーザが自由に変更可能にするために、図形の組合せを用いた視覚化変換方法を提案する。これにより、システムの視覚化表現が柔軟に変更できるようになり、ユーザによるプログラム表現の幅が広がる。

2 システム CafePie

我々はVPSとしてCafePie [3] [4] [5]を作成している。特に直接操作 [6]に注目し、マウスによるDrag-and-Drop操作により単純な枠組みで全てのプログラムの編集・実行を行うことを目指して研究を行ってきた。

CafePieでは、代数的仕様記述言語であり、その処理系でもあるCafeOBJ [7]をベース言語とする。ターゲットとしてこのような宣言型言語を選んだ理由は、VPが宣言的であり相性が良いことが挙げられる。また、そのプログラムの実行は項書換え系、即ち、プログラム構造のリダクションで与えられるため、静的なプログラムの編集と動的なプログラムの実行との間とのギャップが少なくなる。

図 1　スタックにおける項の視覚化

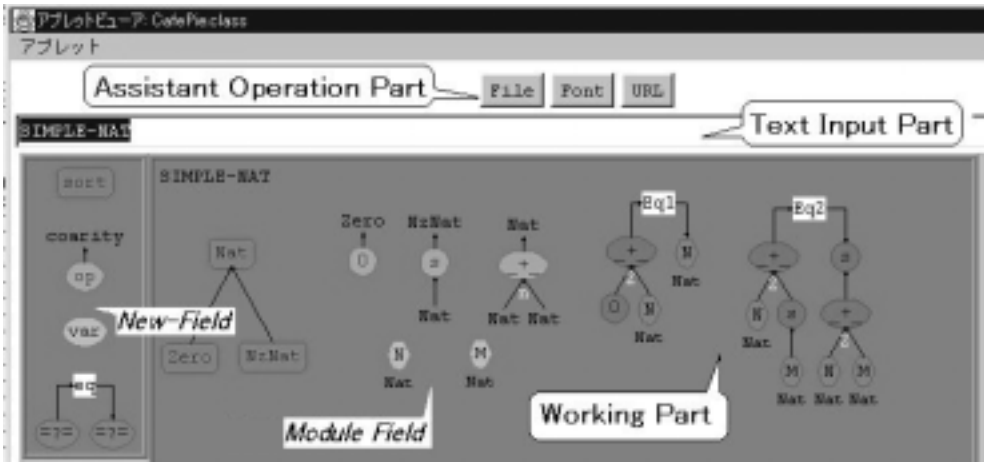


図 2　CafePie の実行画面

CafeOBJ はモジュール単位でプログラムを記述する。CafePie はこのモジュールを視覚化し編集・実行を行うことができる。そのモジュールに含まれる基本的な要素は、ソート・演算・変数・等式であり、視覚化の際には、特に、データ構造を表す項の基本構成要素である演算と、項の書換え規則を表す等式が重要となる。

演算は図 3に示すようにラベル付の水色の楕円で表される。演算には引数がその下方に並び (第 1 引数が最左端)、返却値がその上に配置される。また、それらを結ぶように引数から演算の中心へ、演算の中心から返却値へという具合いに有向線が引かれる。等式は図 4に示すようにラベルを中央に配置し、左辺と右辺をその両端にぶら下げる形で配置する。　左辺と右辺とは、即ち項である。項は演算 (と

図 3　演算

図 4　等式

変数) の組合せで表される。例えば、スタック構造を示す項

```
push( E3, push( E2, push( E1, push( E0, empty ))))
```

の場合には図 1のように視覚化することができる。

CafePie の実行画面を図 2に示す。画面中央がモジュールを作成する部分である (Module　Field)。プログラムはこの中で視覚化・編集され、実行される。その左側上部にあるのが各要素の基本図形を納めたもので、この部品を使用してプログラムを記述する (New-Field)。これらをまとめて　Working Part と呼ぶ。この上部にテキスト入力とファイル入出力用ボタンなどの補助的な作業用のものが配置される。

3　視覚化変換規則

CafePie 上に表示するプログラムは、システムから与えられた規則に従い視覚化される (図 1)。このようにシステムにより与えられた視覚化では、必ずしもその表現が適切であるとは言えない。そこで我々は、ここからさらに視覚化の方式を与えるために、視覚化変換規則を導入してシステムによって与えられた視覚化の再定義を行う。

3.1　視覚化変換規則の編集

我々は、ユーザに完璧な作業を要求し完全なプログラムを作成するを行うことよりも、大雑把な操作でも完全でないにしろ大体の作業が行えるようにすることを目指す。

マウスを使って図形の編集を考えた場合、その全てを直接操作 (Drag and Drop 手法など) で扱うことが可能である [3] [4]。この操作手法は、細い線を選択する、図形をちょっと拡大する、など微妙な動きを必要とする作業には向かない。そのため、操作を単純にする、対象物にある程度の大きさを持たせる、というような工夫が必要となる。変換規則を複数の図形の集まりとみなした。つまり、これらの規則はドローツールのように描くのではなく、図形を組立てるという操作で編集することで行う。このため、以前の編集における特色 [3] [4] (プログラムの編

集要素がそれぞれ図形の集まりであり、それを組立てることでプログラミングしていく手法) を生かした形、マウスを使った直接操作を用いた実装が実現できる。

まず、長方形、円などの良く使われる基本的図形は用意しておき、必用に応じてイメージなどをファイルから呼出すことにする。ユーザが行う作業は、用意された図形や既に作成した図形を再利用して、視覚化変換規則を各演算上に定義することである。以下で具体的な例を使ってこの作業を見てみる。

3.2 視覚化例

例としてスタックの再視覚化を考える。スタックの要素となる演算には主に empty と push がある。empty はスタックが空であることを示す。これは長方形で表すことにする。また、 push はスタックに 1 つの要素を積んだという状態を表す。これを何かスタックがあるときにその上に要素を積むという表現に変換する。 この演算の編集は図 7に示すように、 Drag-and-Drop 操作を繰り返すことで作成できる。これはスタックの push 演算に対し変換規則を定義の手順を示している。 その手順は次のようになる。まず、長方形の図形を用意する。次に、その中に push の引数である Elt を利用してそれを中に配置する。このとき Elt は文字ではなくそれをラベルとして持つ楕円に変換される。最後に今度は push のもう一つの引数である STACK を利用してその図形の下方に配置させる。

図 5 empty の視覚化

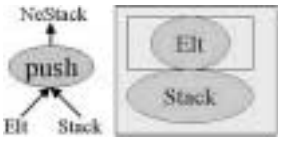


図 6 push の視覚化

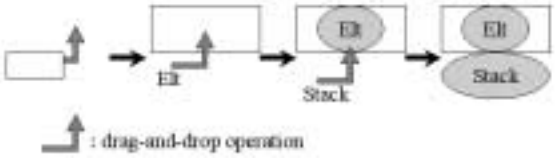
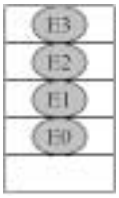


図 7 push における視覚化変換規則の編集

これらの作業で、前に示したスタックの項 (図 1) は図 8のようにコンパクトで分かりやすい図で再視覚化できる。単に表示方法を変更するだけではなく、さらにこれらの変換規則を利用して等式を記述することができる。各等式にはそれを引起す演算があり、これを対応付けて表記する。例えば、スタックからデータを一つとる作業に対しては演算 pop がそれにあたり、図 9のように視覚化できる。このように等式でも再視覚化できることは、実行表示にもこの手法がそのまま使えることを意味している。



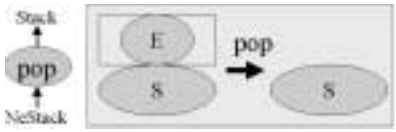


図 9 pop と等式の視覚化

図 8 再視覚化

さらに言えば、視覚化は 1 通りである必要はない。視覚化変換規則を用いる利点はプログラム表現を自由に変更できるという点にある。例えば、他の視点から見たスタックの視覚化を考えてみる。

一般に待ち行列はキュー (FIFO) と呼ばれスタック (LIFO) と対峙するデータ構造を意味する。これらの違いはデータを取り出す順番にあり、視覚的表現に見るだけならば実は大差はない (ユーザのもつイメージよって決まる)。

人の順番待ちはキューであると言われているが、列の先頭からではなく後尾から取り出すようにした場合、それはたちまちスタックとなり得る。例えば、図 5や図 6の代りに、各演算に対し以下のような視覚化を試みる (図 10,11)。

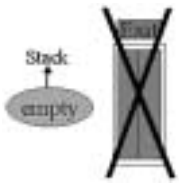


図 10 empty その 2

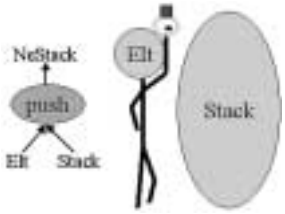


図 11 push のその 2

この視覚化方法におけるスタックの要素に人の顔を用いることで以下のような視覚化が可能である (図 12)。

これは先頭に出口があったのだが、何らかの原因で出口が故障してしまい、列の末尾からしか出られなくなったというスタック構造を示している。

このように 1 つのプログラムに対し、違った視点からプログラムの意味を考察することが可能とな

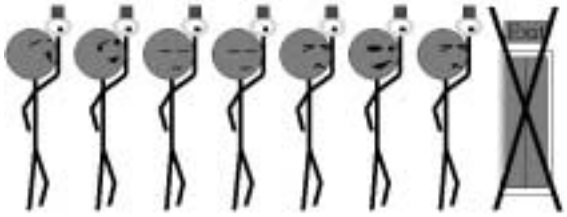


図 12　スタックの視覚化その 2

り、プログラムの直感的な学習などに役立つ。

4　関連研究

Visulan [8]は、ビットマップ書換えに基づくビジュアルプログラミング言語である。プログラム表現として扱うビットマップをパターン置換ルールの順序集合で記述する。それらのビットマップをどのように描くかという点については何も明記していない。VISTATCH [9]は図的なルールをもとに図形を書換えるビジュアル言語である。例として図形エディタなどがあり、ユーザやシステムによって発生されるイベント (マウスボタンを押すなど) によって書換えが制御できる。

これらの研究はそれ自身がオリジナル言語でありベース言語を持たない。我々は、システム実装する際、既存のテキストベースの処理系に寄生させて実現するアプローチを取った。こうすることで、既存の資産を有効に活用することができ、システムの中心部の開発に重点を置くことができる。

また、書換えルールはプログラムの実行部分に利用される (CafePie での等式を視覚的表現で記述する点がこれに近い)。CafePie では、ベースとなる言語があり、その視覚化の形態を静的に変更することに図形の書換えを利用するという点で異なる。

5　まとめ

図や写真などのイメージを使うことで、プログラムをより理解しやすい表現に視覚化できる。我々は、一旦システムによって視覚化されたプログラムの表現をユーザにより自由に変更可能にするために、図形の組合せによる視覚化方法を提案した。これを実現するために視覚化変換規則を用いることでこれが実現できることを我々の開発している CafePie において試験的に実装することで示した。実装の際には、以前の操作手法 [3] [4]をそのまま適用できるように、直接操作を用いた編集が行えるように留意した。

本研究はリアリスティックな視覚化の第一歩である。ここでいうリアリスティックとは、プログラム表現をユーザが現実世界に対応付けて持つプログラムの概念イメージに近づけるように視覚化することを目指すことを意味する。基本的な視覚化手法の考え方は出来上がったと思われるが、これから必要なことは、それをどのように構築し、その手法を体系化することである。また、単なる視覚化だけの表現ではまだ不十分であり、他に動きを取入れた視覚化法などが考える必要がある。

参考文献

[1] B. A. Myers: Taxonomies of Visual Programming and Programming Visualization, *Journal of Visual Languages and Computing*, Vol. 1, No. 1, pp. 97–123, 1990.

[2] J. Tanaka: PP : Visual Programming System For Parallel Logic Programming Language GHC, *Parallel and Distributied Computing and Networks '97*, pp. 188–193, August 11-13 1997.

[3] 小川 徹 and 田中二郎: Drag and drop 手法を用いた代数的仕様記述言語における視覚的プログラミング環境, 日本ソフトウェア科学会 第 *15* 回大会論文集, pp. 165–168, 1998.

[4] T. Ogawa and J. Tanaka: Double-Click and Drag-and-Drop in Visual Programming Environment for CafeOBJ, *Proceedings of International Symposium on Future Software Technology (ISFST'98)*, pp. 155–160, Hangzhou, October 28-30 1998.

[5] T. Ogawa and J. Tanaka: Realistic Program Visualization in CafePie, *Proceedings of IDPT-99*, 1999 (to be appear).

[6] B. Shneiderman: Direct Manipulation: A Step Beyond Programming Languages, *IEEE Computer*, Vol. 16, No. 8, pp. 57–69, 1983.

[7] R. Diaconescu and K. Futatsugi: *CafeOBJ Report*, World Scientific, 1998.

[8] 山本 格也: ビットマップに基づくプログラミング言語 Visulan, インタラクティブシステムとソフトウェア *III*, 日本ソフトウェア科学会 *WISS'95*, pp. 151–160, 近代科学社, 1995.

[9] Yasunori Harada and Kenji Miyamoto: VISPATCH: Graphical rule-based language controlled by user event, *Proceedings of the 1997 IEEE Symposium on Visual Languages*, pp. 162–163, 1997.