データ構造の視覚的カスタマイズとプログラム実行の視覚化

Visual Customization of Data Structures and Visualization of Program Execution

小川 徹　田中 二郎 *

Summary. This paper represents a mechanism to customize views of data structures. We assume that drag-and-drop operations are used for handling data structures. We also describe that both of program editing and execution can be visualized using the customized views.

1　はじめに

ビジュアルプログラミング [1] は，図形的表現を用いてプログラムを記述することでプログラムモデルの視覚的理解を補助するが，まだ不十分である．従来からのテキストベースのプログラミング環境では，色や形などの視覚的な情報も含めて全てをテキストで表現していた．これに対し，テキストプログラムで扱うもの全てを視覚化しようと試みたビジュアルプログラミングシステム (VPS) には， PP[2] や CafePie[3][4] などがある． CafePie は我々が開発している代数的仕様記述言語 CafeOBJ[5] の視覚的プログラミング環境である．これらのシステムでは，テキストプログラムで扱う要素を (ノードとエッジのように) 単純なオブジェクトで視覚化する方針をとる．この視覚化の特徴は，テキスト表現から図形的な表現への変換が比較的容易であること，ノード同士の繋がりを視覚的に捉えることでプログラム構造が把握しやすいことである．しかし，プログラムは全てノードとエッジのみで記述するため，プログラム全体やその一部分を一瞥することによるプログラムの判別が困難である．

本研究では，テキストプログラムを利用した VPS において，視覚的部分のカスタマイズをユーザが簡単に行なえるような枠組を提供するために，ドラッグ＆ドロップ (DND) 手法を用いた方法を提案する．また，カスタマイズされた表現を用いたプログラムの実行が可能であることを示す．

2　ビューのカスタマイズ表示とその効果

我々は，データ構造を表す項の視覚的部分 (ビュー) をカスタマイズの対象にしている．図 1はスタックに積木構造というビューを CafePie 上で作成した例の一部であり，これはカスタマイズ前後の対応関係を表示するための**ビュー対応画面**を示している．図 1左がシステムによるデフォルトのビュー，図 1右がユーザによってカスタマイズされたビューを示す． CafePie では，項の構成要素を演算と変数とし，項のデフォルトのビューは，演算や変数をラベル付ノードでその関係をエッジで表した木構造で表現する (図 2左端). 図 1左の中央に演算を示すラベル付楕円 push がある．引数 Nat と Stack をラベルの下

* Tohru Ogawa, 筑波大学 工学研究科 電子・情報工学専攻, Jiro Tanaka, 筑波大学 電子・情報工学系

方に， push の型を示す NeStack をラベルの上部に配置する．また図 1右は，変数 Stack の上に変数 Nat を配置することでを表し，何かスタックがあったらその上に要素を積むことを意味する．

2.1　カスタマイズ前後における表現とビューの切替え

木構造によるデフォルトのビューには，図 2左中のようなビューが定義されている．このビューからポップアップメニューを呼び出し，図 2右中に切り替えることによって，木構造というビュー全体を積木構造 (図 2右端) に変更することができる (この場合は，演算 empty に対するビューも同時に切り替えている)．また，逆に積木構造から木構造にビューを戻すことも可能である．

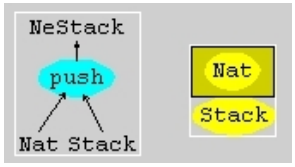
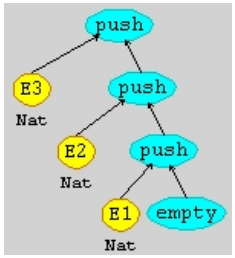


図 1. ビュー対応関係



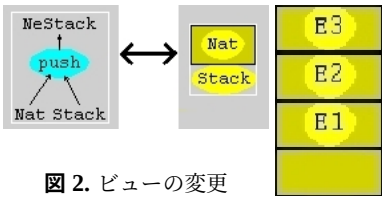


図 2. ビューの変更

2.2　カスタマイズ機能の適用例

このカスタマイズ機能によって，一つのプログラムモデルに対して異なるビューを定義することが可能となる．例えば，図 2右端の積木構造の代わりにスタックを人の並びと見なし，演算 empty を行き止まり，演算 push を列の最後に人が並ぶこと考える．スタックの要素を人の表情として他に定義することで，図 2右端の代わりに，図 3のような視覚化を行なうことも可能である (ただし，ここでは 7 個のスタック要素から成る)．

3　ドラッグ＆ドロップを用いたビューの変更

このようなデータ構造のビューのカスタマイズは，例えば，自分の作成したプログラムを他者に説明する場合に，テキストだけでは表現しにくい概略イメージを直観的に相手に伝えることができる．この場合の概略イメージの作成は，時間を掛けて詳細なパラメタを設定するのではなく，なるべく簡単な操作で手早く編集することが望まれる．そこで我々は，**図形オブジェクトに対するの直接操作のみを用いて編集すること**を目標にし， DND 手法に注目した．

ビューのカスタマイズは，図 1右のカスタマイズ後を提示する枠内に，ユーザが図形オブジェクトを組み合わせることで行う．図形オブジェクトは，変数以外に，矩形，丸矩形，楕円とユーザ定義オブジェクトがある．図形オブジェクトの操作は，基本的に DND 手法を用いた**図形の移動**と**拡大／縮小**から成る．

3.1　図形間の状態の視覚的表示

図 2右端に示すようにスタックの push を積木構造で表現する場合，中心がズレたり，幅の長さが違うとうまく揃わない．これを解決するために，現在の図形間の状態を**視覚的に表示する**ことで支援する．変数以外の図形オブジェクトはビューの構成要素であ

り，図形のサイズはその場で決まる．しかし変数オブジェクトの場合，項の編集時には変数の代わりに任意のオブジェクトに置き換わる可能性があり，図形のサイズを決定できない．この場合には項の編集時に有効になる図形的な制約が付く．

例として，図 1の push の編集を挙げる．矩形の内側に引数 Nat を収める場合，矩形の中ほどに変数 Nat をドラッグする．矩形と Nat の高さの中心が揃うと，システムは横線「－」を表示する (図 4左端)． Nat をそのまま横にスライドさせ，横幅の中心が揃うと，今度は十字線「＋」を表示する．このときにマウスを放し Nat をドロップすることで，システムは中心を揃える (図 4左中)． Nat の下に変数 Stack を配置する場合は，矩形の下側の方に Stack をドラッグする．横幅の中心が揃ったときにドロップすることでその中心が揃う (図 4右中)．更に，横幅のサイズを揃えたい場合には， Stack を拡大／縮小して調整する．左右の端が揃ったことを示すために，システムは図形の両脇に縦線「｜」を表示する (図 4右端).

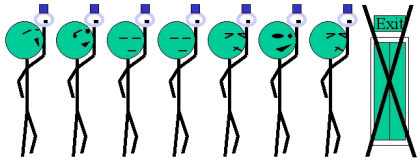


図 3. 視覚化 (人の並び)

図 4. ドラッグ＆ドロップによる編集

3.2 配置の調整

項の編集は，まず，各部品に対してビューを定義し，それらを組み合わせることで項全体のビューを作成するという，ボトムアップ方式で行う．このボトムアップ方式は，**図形を組み合わせて編集してみないと全体像がわかりにくい**という問題がある．これを解決するために，編集中の図形からいつもでビューの修正ができるようにした．例えば，図 5左端のようにスタックの要素間にある隙間を除去したいとする．この場合には，まず，ユーザがスタックの push に対応する部分を指定し，そのビュー対応画面 (図 5左中) を呼び出す．そして，図 5右中のように DND 操作で Stack を移動させ，矩形の下と Stack の上を隣接することで，図 5右端のように隙間のない形に修正できる．

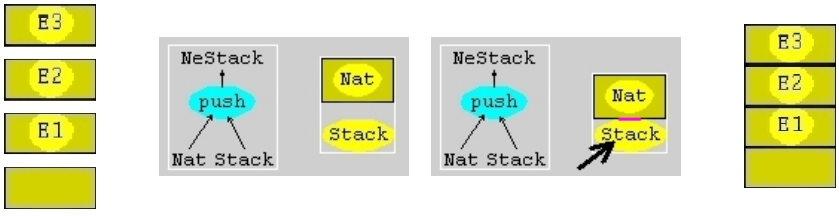


図 5. カスタマイズされた配置の調整

4　制御構造の編集とプログラム実行

データ構造 (項) のビューを変更することで視認性が向上するが，そのモデルの判断が困難ば場合がある．図 3の人の並びを一瞥した場合，他のユーザはスタックではなくキューだと想像する恐れがある．この区別を行うために制御構造の理解が必要になる．プログラムを実行することで制御構造を判断できる．制御構造 (等式) は，**図形書換えルール**を用いて表現する．この表現は，シミュレーションのためのプログラミングシステム KidSim[6] などで用いられており，動作前と動作後とを図示することで図形の書換えを示す手法である．例えば，スタックから一番上の要素を取り除いた残りを返す pop の図形書換えルールは，図 6右のようになる．この等式は，スタック S に要素 N が積まれているとき，要素 N を取り除いたスタック S を得ることを意味する．

カスタマイズされたビューを切り替えることで， 2.1節の項と同様にビューの変更を等式へ反映させることが容易にできる．例えば， push 上にカスタマイズされたビューを図 2左から図 2右に切り替えることで，システムは図 6左のから図 6右のように等式上のビューを変更する (このとき等式のモデルは変更していないことに注意されたい)．またビューを元に戻す場合も，その逆操作で簡単に実現できる．

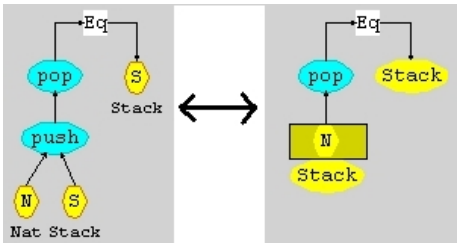


図 6. 制御構造とビューの変更

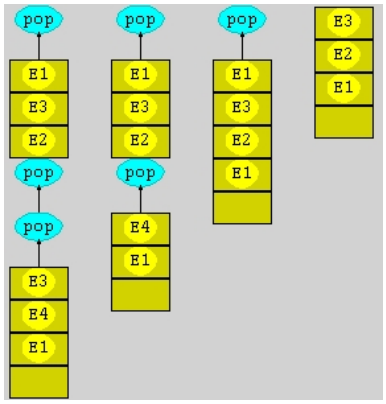


図 7. プログラムの実行表示

4.1　プログラムの実行

CafePie 上の実行処理は既存の CafeOBJ インタプリタを利用する．図 7は STACK の項，

```
pop(push(E1,push(E3,push(E2,pop(pop(push(E3,push(E4,push(E1,empty))))))))
```

を実行の対象である項 (ゴール) として与えた場合の実行結果を示している．ゴールはカスタマイズ済みビューを用いてユーザが与える (図 7左端)．まず， CafePie はインタプリタで解釈可能なテキスト形式に変換する．ビュー変更に関わらず一意に項のモデルが決定できる． CafePie はビュー変更済みの項からインタプリタ形式に変換し，インタプリタに処理を依頼する．インタプリタから実行結果を受け取ると， CafePie はそれを解釈し内部表現に変換する．その後で実行の表示を開始する．図 7の左から右のように実行が進み，結果として図 7右端に示す項を得る．このテキスト表現は次のようになる．

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

4.2　アニメーションの設定と表示

図 7のような静的に表示することで実行過程全体を把握できる．また，より効果的に実行表示するために動的な実行表示もサポートしている．実行を動的に表示した場合，図

7左端のユーザが与えたゴールはその場で変化する．そして，変化した項を一定の間隔で次々と表示し，図 7右端の項まで書換えるとアニメーションは終了する．実行の動的表示はスナップショット形式である．例えば図 7左端において， pop を重ねて表示していた場合にどちらの pop を評価して書換えたのかを判別しにくい．図形の書換えを滑らかに見せる仕組みが必要である．我々は，図形書換えルールである等式の間に，**スナップショット間の状態を追加する**ことでアニメーションを定義できるようにした (図 8). 図 8は， pop における書換えの間の図形を追加した等式である．横軸は表示するタイミングを表し，左側から右側に書換える途中で 50 %のタイミングで図形を表示することを意味している．追加した図形をドラッグして左に移動すれば，その表示は少し速いタイミングになる．また，ユーザが等式の左側 (または右) の図形を等式の中央にドラッグすることでそのコピーが作成される．ユーザはこの図形の一部を移動するなどして追加する図形を編集する．追加する数を 2 個， 3 個と増加すれば，より滑らかなアニメーションを作成できる (補完数を増やし過ぎると処理が間に合わず全体として遅くなる). 追加した図形の削除は，等式以外の所に移動することで行なう．プログラムの実行は図 9のようにスナップショットの間が追加した形でより滑らかなアニメーションとして表示される．本来アニメーションの作成は手間のかかる作業であるが，既存の図形を利用し DND 手法を用いることで容易に定義することができる．

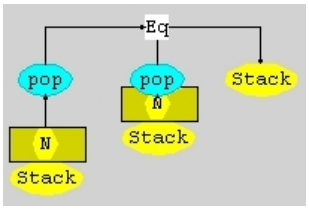


図 8. アニメーションの設定

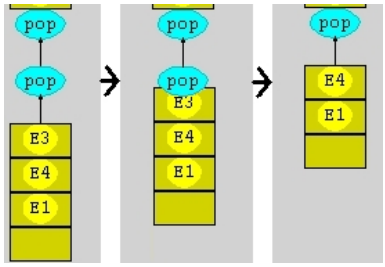


図 9. アニメーションの表示

5　実装方式

CafePie において， DND 手法を用いたデータ構造 (項) の編集を簡潔に表現するために，「代入」という概念を用いてモデル化している [3]．代入対象を変数とし，他の項をその上に DND 手法を用いて追加／削除することで項の編集を行なう．項におけるビューのカスタマイズは変数の位置を変更するだけであり，ビューの変更後でも代入方式をそのまま使用できる．これは，等式上の項にも言えることである．

1 つの項に対して同時に複数のビューを定義することが可能であるが，例えばスタックに対して対象的な 2 つのビュー，上に積むというビューと下に積むというビュー，が定義されている場合などに問題が起きる．これらのビューをスタックの各要素に交互に適用していくと中心から外側へ上下交互にデータを積むことになり，項を見ただけでは積まれた順番が判定できなくなる．ビューの設定を制限するなどの工夫が必要になる．

CafePie のベース言語となる CafeOBJ は，等式のある一階の代数系を表現できるため，スタックを編集／拡張することで double-ended queue なども表現できる． CafeOBJ はモジュール構造から構成される銀行の ATM システムなどの記述例もあり [5]，より複雑なシステムの記述にも将来に対応可能であると考えられる．

6 関連研究

データ構造の変化を効果的に見せる手法としてアルゴリズムアニメーション (AA) がある． Pavane[7], Zeus[8] などがあり，１つのアルゴリズムに対し詳細なアニメーションを指定できるなどの特徴を持つ．我々は，単純なアニメーションに限定する代わりに，変更可能なプログラムを対象としており，視覚的言語のような言語的な立場と AA の立場との中間的に位置する． Visulan[9] 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． Visulan がビットマップを対象としているのに対して，我々は通常の図形を対象としている．また，ビットマップそのものがプログラムモデルを示しているため，ビューだけの変更はできない． GELO[10] は，データ構造の視覚化においてユーザがカスタマイズ可能なシステムである．これは直接操作によるビューの編集方法は述べていない． DND 手法によるビューカスタマイズでは，例示により図形間の制約を予め明示的に与えている．図形エディタ Chimera[11] のように履歴からの高機能な予測を用いているのではなく．予め決められた制約に対する表示を行なっている．

7 まとめ

視覚的部分のカスタマイズをユーザが簡単に行なえるような枠組を提供するために，DND 手法を用いた方法を提案した．また，本手法を CafePie に実装することで，カスタマイズされた表現を用いたプログラムの実行が可能であることを示した．

参考文献

[1] B. A. Myers. Taxonomies of Visual Programming and Programming Visualization. *Journal of Visual Languages and Computing*, 1(1):97–123, 1990.

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

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

[4] T. Ogawa and J. Tanaka. Realistic Program Visualization in CafePie. In *Proceedings of the fifth World Conference on Integrated Design and Process Technology (IDPT'99)*, Dallas, Texas, June 4-8 2000. (CD-ROM), Copyright (c) 2000 by the Society for Design and Process Science, (SDPS), 8 pages.

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

[6] Alan Cypher and D.C. Smith. KidSim: End User Programming of Simulations. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, pages 27–34, 1995. Denver, CO.

[7] K. C. Cox and G.-C. Roman. Visualizing Concurrent Computations. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pages 18–24, 1991.

[8] M. H. Brown. Zeus: A System for Algorithm Animation and Multi-View Editing. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pages 4–9, October 1991.

[9] 山本 格也. ビットマップに基づくプログラミング言語 visulan. In 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*, pages 151–160. 近代科学社, 1995.

[10] S. P. Reiss, S. Meyers, and C. Duby. Using gelo to visualize software systems. In *Proc. of the 2nd Annual Symposium on User Interface Software and Technology (UIST'89)*, pages 149–157, Williamsburg, VA, 1989.

[11] David Kurlander and Steven Feiner. Inferring constraints from multiple snapshots. *ACM Transactions on Graphics (TOG'93)*, 12(4):277–304, October 1993.