インターフェイスの街角 (10)
ビジュアル・プログラミング

増井俊之

最近は、計算機のユーザー・インターフェイスとして、シェルのような文字ベースの CUI (Character-base User Interface) より、GUI (Graphical User Interface) のほうが主流になりつつあります。プログラミングの世界でも、テキストを用いたプログラミングの代わりに 2 次元/3 次元のグラフィカルな表現を活用する**ビジュアル・プログラミング**の試みが古くから提案されています。ビジュアルなプログラミング環境では、テキストベースのものとくらべて、GUI と同様の利点を活かしたプログラム作成が可能になります。

ビジュアル・プログラミングとは

ビジュアル・プログラミングの定義は時代とともに変化しているようです。従来は、プログラムをすべて図表で表現するものをビジュアル・プログラミングと呼ぶ狭義の定義が一般的でした。しかし、ここではテキスト以外の画像や図表、グラフなどをプログラムの一部に使用しているものも含めることにします。Visual C++をはじめとする一連の Microsoft 製品では、プログラム本体はテキストで記述しなければなりません。しかし、GUI の部品配置などに画像を使えるので、ビジュアル・プログラミング環境の 1 つと考えてもいいでしょう[1]。

プリティプリンタやアルゴリズムアニメーション・システム[2]のように、プログラムの静的構造や動作などを視覚化する手法をこの範疇に含める考え方もあります。しかし、現在はこれらを〝プログラム・ビジュアリゼーション〟と呼び、6 月号で紹介した〝情報視覚化〟の手法の一種と捉えるのが一般的です[3]。

ビジュアル・プログラミングの利点

テキストベースのプログラミングと比較すると、ビジュアル・プログラミングには以下のような利点があります。

- 複雑な構造や関係を表現しやすい
  表やグラフを使用すると、テキストでは表現しにくい複雑な構造や関係を簡単に表現できる場合があります。配列で表現されたグラフ構造や状態遷移図は理解しにくいものですが、図として表現されていれば比較的容易に構造を把握できます。
- 具体性
  色や形をプログラムで扱う場合、〝Red〟〝Rectangle〟などのテキストよりも、赤色の〝赤〟という文字や四角形の図形（□）のほうがはるかに直感的です。GUI のツール部品のように 2 次元で表現されるものは、テキストに変換するより絵のままのほうが扱いやすいので、プログラミングに〝インターフェイス・ビルダ〟を利用するのが一般的になりつつあります。
  CUI にくらべて GUI が好まれる理由の 1 つは、抽象的な思考や操作が苦手な人が多いからでしょう。たとえば、ファイルシステムの構造やファイルの名前を念頭においてファイルを操作したり、コマンド名を思い出してタイプしたりするよりも、画面上のアイコンをドラッグするほうがはるかに簡単な場面が多いでしょう。

1 ビジュアル・プログラミングについて論議するニュースグループとして comp.lang.visual がありますが、ここに Visual C++などに関する質問を投稿して怒られる人が絶えないようです。

2 プログラムの実行状態をアニメーションとして表示するシステム。

3 テキストベースのプログラムをフローチャートに変換するのがプログラム・ビジュアリゼーションで、フローチャートを直接編集するのがビジュアル・プログラミングといえるかもしれません。たいした違いはないかもしれませんが……。

プログラミングには、さらに広範な抽象的思考が必要です。10 人、20kg などの具体的な数や量も、抽象的な値として扱わなければなりません。さらに、〝変数〟や〝関数〟〝プロセス〟などといった抽象的な概念に関する知識も必要なため、プログラミングの敷居が高くなっているといえます。ビジュアル・プログラミングでは、これらの抽象的なものもある程度具体的に表現できるので、プログラミングが容易になる可能性があります。

- 視認性

同じプログラムでも、表現方法を工夫すれば見やすく（恰好よく）なることがあります。テキストベースのプログラムでも、

if (strcmp(s, **"-help"**)==0) /*comment*/

のようにキーワードを太字にしたり、重要度に応じて文字サイズを変更するといった〝プリティ・プリンティング〟の手法により、プログラムの見栄えがよくなって読みやすくなる場合があります。

- 並列性の表現が容易

多くのテキストベースの手続き型言語では、一般に上から下へ順番に制御が移動します。したがって、同時に実行される複数の処理はうまく記述できません。2 次元表現が可能なビジュアル言語では、複数の処理を並べて書くことによって並列処理を分かりやすく表現できます。

- 宣言的記述

プログラムが 1 次元的に表現されるテキストベースの言語とは異なり、ビジュアル言語では 2 次元空間に言語要素をちりばめても、それらの配置が実行順序や優先度などを強く主張することはありません。ですから、規則や制約を列挙してプログラミングをおこなう宣言型言語のプログラムの表現に適しています。

- データの流れの表現が容易

テキストベースの一般的な手続き型言語では、プログラムの制御の流れは比較的容易に表現できますが、データの流れが把握しにくい場合があります。データの流れを矢印で表現し、データの処理をおこなうノードを矢印のあいだに挟むデータフロー型のビジュアルな表現をすれば、データの処理の流れが掴みやすくなります。

UNIX のシェルでコマンド列をパイプでつなぐ方法も、ごく単純なデータフロー表現といえます。

- 一覧性

テキストベースの巨大なプログラムの構造を把握するのは容易ではありません。しかし、プログラムをグラフィカルに表現してズーミング・インターフェイスのような手法を利用すれば、テキストベースのプログラムよりも、大きなプログラムの全体像を把握したり、一部分を詳細にみることが簡単になります。

- 複数の視点

もともと時間は 1 次元ですが、予定表をひと続きの大きなリストで表現するより、折り畳んでカレンダーのような 2 次元の表にしたほうが、曜日の関係が分かりやすくなります。

ビジュアル言語では、属性を X 軸、Y 軸にマッピングするといった多次元表現により、プログラムやデータを複数の視点から眺めることができるので、1 次元のリストより理解しやすくなる可能性があります。

- エンドユーザー・プログラミング

現在は、テキスト言語の開発環境のほうがビジュアル言語の環境よりもはるかに進んでいるので、プログラマーとしては、テキストベースの言語で十分な仕事にわざわざビジュアル言語を使う必要はありません。しかし、上述のようなビジュアル・プログラミングの各種の特徴をうまく活用すれば、テキストベースのプログラミングに慣れていない人でもちょっとしたプログラムを作れるかもしれません。

たとえば、アプリケーションのカスタマイズについて考えてみましょう。ユーザーの求める機能は千差万別です。したがって、ユーザーごとに特殊で複雑な作業を計算機に指示しなければならないケースが頻繁に起こる可能性があります。現在のところは、必要になるかもしれない機能をあらかじめすべて組み込む（高機能なワードプロセッサのような）アプローチが一般的です。しかし、すべての機能に対応するのは実質的に不可能ですし、システムの巨大化を招くという問題もあります。これに対し、ユーザーがある程度のプログラミングをおこなうことによってこのような状況に対処しようという、〝エンドユーザー・プログラミング〟[6] が提唱されています。

抽象的概念に関する知識を必要とせず、作成も実行も容易で、普通のユーザーでも手軽に使えるビジュアルなエ

ンドユーザー・プログラミング環境があれば、このようなアプローチが主流になるかもしれません。

ビジュアル・プログラミングの分類

Margaret M. Burnett は、ビジュアル・プログラミング関連の文献を下記のように分類する方法を提案しています [1][4]。

- 並列言語 (Concurrent languages)
- 制約型言語 (Constraint-based languages)
- データフロー言語 (Data-flow languages)
- フォーム/スプレッドシート言語 (Form-based and spreadsheet-based languages)
- 関数型言語 (Functional languages)
- 操作型言語 (Imperative languages)
- 論理型言語 (Logic languages)
- マルチパラダイム言語 (Multi-paradigm languages)
- オブジェクト指向言語 (Object-Oriented languages)
- 例示プログラミング言語 (Programming-by-demonstration languages)
- ルールベース言語 (Rule-based languages)

前述のように、ビジュアル言語は、順番に制御が移動していく手続き型言語よりも並列言語や宣言型言語の記述により適しているため、このような言語への適用が多くなっています。

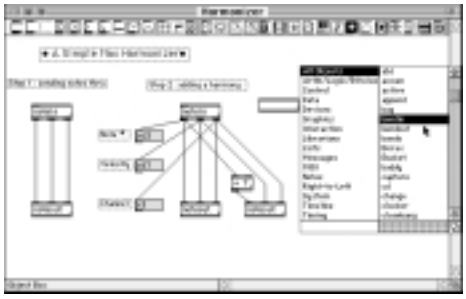
Burnett は、ビジュアル言語の表現形式についても次のように分類しています。

- ダイアグラム言語 (Diagrammatic languages)
- アイコン言語 (Iconic languages)
- 静的画像列（紙芝居型）言語 (Languages based on static pictorial sequences)

狭義のビジュアル言語はこのように単純な視覚的表現を用いるものが多いようです。部分的にビジュアル表現を利用する言語も含めると、さらに多くの表現方法が使われています。

4 http://www.cs.orst.edu/~burnett/vpl.html に、ビジュアル・プログラミングに関する膨大な文献が分類されています。

図 1　MAX



ビジュアル・プログラミングの例

膨大な数のビジュアル言語が提案されていますが、ここではその一部を紹介します。

MAX

MAX[5]は、フランスの IRCAM（Institut de Recherche et Coordination Acoustique/Musique：計算機音楽研究センター）で開発されたデータフロー型の計算機音楽記述用言語です。MIDI 信号や DSP 信号の流れを線で表現し、さまざまな処理を表現する矩形ノードをそれらの線で結ぶことによって音楽信号を処理します（図 1）。たとえば、ある MIDI 信号を表現する線を ``+ 1''というノードにつなぐと、そのノードからは半音高い MIDI 信号が出力されます。

計算機で音楽を処理するには、各種の連続的な音信号を並列に扱う必要があります。データフロー型のビジュアル・プログラミングを利用すれば、音の流れや処理を視覚化することで直感的な表現ができるので、テキスト言語によるプログラミングに不慣れな音楽研究者や作曲家、演奏家のあいだでも MAX はひろく使われています。

MAX はもともと NeXT で開発され、SGI のマシンにも移植されていますが、現在は Macintosh 用の製品がよく使われているようです。また、MAX の後継の FTS、PD[6]といったシステムも開発されています。

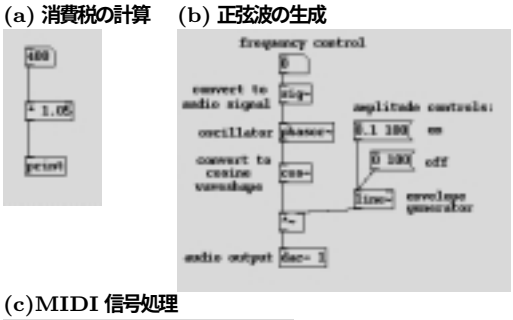
図 2 は、PD によるプログラムの例です。図 2-a では、数を表す一番上のノードの出力が乗算を表す中間のノード

5 コンピュータ音楽研究の先駆者である Max Mathews 氏にちなんだ名称です。

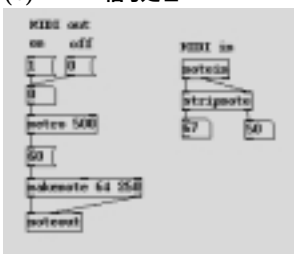
6 IRIX 版 PD は ftp://crca-ftp.ucsd.edu/pub/msp/、Linux 版は http://iem.mhsg.ac.at/~geiger/から入手できます。

図 2　PD のプログラム例

(a) 消費税の計算　(b) 正弦波の生成



(c)MIDI 信号処理



に入力され、その結果が印刷を表現する下のノードに入力されています。マウスを使って対話的に数値が変更でき、変更するたびに計算結果が出力されます。

図 2-b は、PD でデジタル信号処理をおこなっているところです。周波数を示す数字から正弦波信号を生成し、出力レベル信号との乗算をおこなって ON/OFF を切り替えます。

図 2-c は、MIDI 信号を処理する PD プログラムです。noteout は MIDI 信号を MIDI 機器に出力し、notein は MIDI 信号を入力します。

Triggers

例示プログラミング (Programming by Example) を支援するビジュアル言語も、いくつか提案されています。

メリーランド大学の Richard Potter が開発した Triggers[7] (図 3)は、ビットマップ画面上のパターン・マッチングにもとづくビジュアルな例示プログラミング・システムです。Triggers のプログラムは、画面に出現するビットマップ・パターンと、それに対応するアクションの集合として表現されます。Triggers はデーモンとしてつねに画面を監視しており、ユーザーが定義したビットマップ・パターンが出現すると定義したアクションが実行されます。たとえば〝OK〟という文字列のビットマップ・パターンとマウスクリックというアクションを登録しておけば、画面上に〝OK〟が表示されると、自動的にマウスがクリックされるようになります。

図 3　Triggers

上の例にはほとんど実用性がありませんが、このようなビットマップ画面上のパターン・マッチングだけでもけっこう複雑な処理をプログラムすることができます。たとえば、表計算プログラムを実行している場合、〝−〟(マイナス記号) がみつかったら周囲に赤い円を描くといった処理をビットマップ・パターンだけで定義できます。これと同様のことを、表計算プログラムのマクロ言語などを使って実現するのはかなり大変です。Triggers は画面上の表示しか使わないので、どんなアプリケーションでも同じようにプログラミングができるという利点もあります。また、アクション実行後に次のトリガーを探す領域をユーザーが指定できるので、さらに複雑な処理の設定も可能です。

AutoMouse

京都大学の山本格也氏は、Triggers と似た処理を Windows 95 上で実行するビジュアル例示システム AutoMouse を作成して公開しています[7]。

図 4 は、Windows の〝スタート〟メニューを認識するとそこでマウスをクリックし、表示されたメニュー項目から〝プログラム〟を選ぶという処理をビットマップ画像を用いて指定しているところです。Triggers にくらべると機能は限られていますが、ちょっとした Windows 操作マクロを作るのに便利です。

Cocoa

Allen Cypher は、子ども向けのビジュアルな例示プログラミング・システムとして KidSim[3] を考案しました。

7 http://www.yuasa.kuis.kyoto-u.ac.jp/ylab/yamakaku/AutoMouse/で公開されています。

図 4　AutoMouse

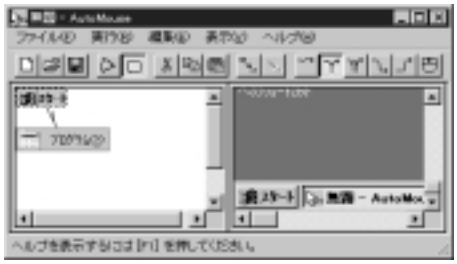


図 5　KidSim による魚のアニメーション

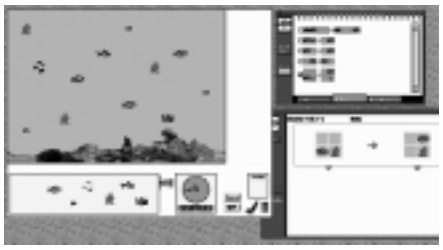


図 6　Cocoa

(a) 虫の移動規則

(b) 虫が階段を越えていくアニメーション

図 5 は、KidSim で魚のアニメーションをプログラムしているところです。画面右下の〝rule editor〟で、2 × 2 のマス領域の左下だけに魚がいる場合には魚が右上に移動するという書換え規則を定義しています。この規則を何度も適用することにより、最初は画面の左下にいた魚がだんだん右上に泳いでいくアニメーションが表示されます。このように、KidSim のプログラムは具体的で分かりやすいので、子どもでもいろいろな規則を設定して複雑なアニメーションが作れます。

Cocoa[2] は KidSim を商品化したものです。図 6-a のような規則を定義していくことにより、虫が階段を越えていくアニメーションが作成できます。

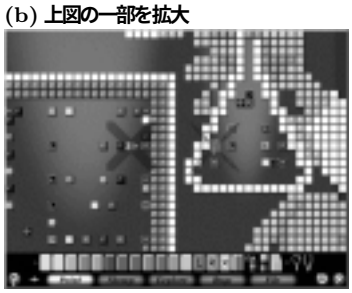
SimTune

SimTune は、メディア・アーチストの岩井俊雄氏が開発した音楽ゲームですが、ビジュアル・プログラミングのための簡便なシステムとも考えられるので紹介します。

図 7　SimTune の〝プログラム〟

(a) バグ研究所

(b) 上図の一部を拡大



SimTune では、画面上を何匹かの〝虫〟が一定速度で動きまわり、点に虫が当たると音が出たり虫の向きが変わったりします。たとえば、虫が当たると〝ド〟〝レ〟〝ミ〟などの音が出る点を一定間隔で並べておけば、その上を通る虫に音階を演奏させることができます。異なる種類の虫を同時に動かして和音の列を鳴らすことも可能です。

図 7-a は、SimTune の〝プログラム〟の例です。人間の絵などを描いている点の大半は演奏とは無関係ですが、フラスコのなかなどに音を出す点が巧妙にちりばめられており、数匹の虫が音を出しながら動きまわっておもしろい音楽を奏でるようになっています。

図 7-b は、フラスコ付近を拡大表示したところです。2 匹の虫が音を鳴らしながら同期して移動していきます。

今後の課題

テキストベースのプログラミング言語にくらべると、ビジュアル言語はまだまだひろく使われているとはいいがたいのが実状です。さきほど述べたように、ビジュアル言語は並列/論理/制約言語などに向いていますが、現在ほとんどのプログラムが手続き型言語で書かれているため、ビジュアル・プログラミングによるメリットが少ないのがもっとも大きな理由でしょう。そもそも、ビジュアル言

語はプログラミングに有効ではないという批判もあるようです。たとえば、次のような意見をよく耳にします。

- 大規模プログラムに対応できない
  画面上にプログラム要素を配置していくとすぐに画面が一杯になってしまうので、大規模なプログラムの開発には向かない。
- 自動レイアウトが難しい
  プログラム要素を追加したり削除したりするたびに、画面のレイアウトを変更しなければならない。これは、ユーザーにとってひどく面倒な操作である。一般に、自動的にレイアウトするのは困難である。
- ポータビリティが低い
  グラフィック処理はシステムにより大きく異なるので、テキストベースの言語にくらべると処理系にポータビリティをもたせるのが難しい。
- 形式的仕様
  ビジュアル言語では、テキストベース言語のような形式的文法や意味を定義するのが難しい。

おわりに

電気通信大学の竹内郁雄氏によれば、インターフェイス・ビルダのように「最初から目に見えているものをビジュアルにするのは、言ってみれば当たり前に簡単」であり、抽象的な「本当に難しいもの」をビジュアルに扱えないから「なっとらん」そうです [9]。まったくそのとおりですが、昔ながらのテキスト言語では、具体的で当り前に簡単に図示できるものすらビジュアルに表現して扱うことができなかったことを考えれば、一応の進歩は遂げているといえるのではないでしょうか。音楽やアニメーションのようなマルチメディア処理をプログラムするためには、どうしてもテキストだけでは不十分ですから、今後はビジュアルなプログラミング環境が増えていくことでしょう。

ビジュアル・プログラミングは、かなり古くから研究されているわりには実際に活用されているものが多くありません。計算機で絵や図を扱うことが従来にくらべて格段に簡単になり、エンドユーザー・プログラミングの重要性が以前にもまして高まってきた現在、ビジュアル・プログラミングはさらに普及する余地が残っていると思います。

IEEE の主催で毎年開催されるビジュアル言語に関するシンポジウム (IEEE Symposium on Visual Languages) では、例年日本からの発表が多く、米国からの発表の割合は（ほかの会議にくらべると）少なくなっています。漢字という視覚的言語を使っているからかどうかは分かりませんが、日本人は欧米人よりも視覚表現に関する意識が高いのかもしれません。世界に通用する日本発のビジュアル言語に期待したいものです。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] M. M. Burnett and M. J. Baker, “A classification system for visual programming languages”, *Journal of Visual Languages and Computing*, Vol.5, No.3, pp.287–300, 1994

[2] Apple Computer, *Welcome to Cocoa — Internet Authoring For Kids*, February 1997 (http://cocoa.apple.com/cocoa/index.html)

[3] Allen Cypher and D. C. Smith, “KIDSIM: End User Programming of Simulations”, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI’95)*, pp.27–34, Addison-Wesley, May 1995 (http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/ac1bdy.htm)

[4] Ephraim P. Glinert, Meera M. Blattner and Christopher J. Frerking, “Visual tools and languages: Directions for the ’90s”, *Proceedings of 1991 IEEE Workshop on Visual Languages*, pp.89–95, IEEE Computer Society, October 1991

[5] Brad A. Myers, “Visual programming, programming by example and program visualization: A taxonomy”, *Proceedings of the CHI’86 Conference on Human Factors in Computing Systems and Graphic Interfaces*, pp.59–66, Addison-Wesley, May 1986

[6] Bonnie A. Nardi, *A Small Matter of Programming*, The MIT Press, 1993

[7] Allen Cypher (ed), “Triggers - Guiding Automation with Pixels to Achieve Data Access”, *Watch What I Do - Programming by Demonstration*, pp.361–380, The MIT Press, 1993

[8] 田中二郎「ビジュアルプログラミング」第 2.3 章、『bit 別冊 ビジュアルインタフェース』pp.65–78、共立出版、1996 年 2 月

[9] 竹内郁雄「ビジュアル言語はまだまだなっとらん」、bit、1998 年 1 月号、pp.20–22

[10] 増井俊之「ビジュアルプログラミングのすすめ」、bit、1998 年 1 月号、pp.17–19