変換規則を外部に持つビジュアルプログラミングのためのトランスレーター *

5N－3

岩田 竜一　田中 義一　上原 稔　森 秀樹 †
東洋大学工学部情報工学科 ‡

1　はじめに

図形によってプログラムの構造や振舞いを記述するビジュアルプログラミング [1, 2]（以下VP）では、通常、作成されたビジュアルプログラムがテキストプログラムへ変換された後、実行される。変換はトランスレーターにより、視覚化の方法を定めた**表現規則**と、プログラミング言語の文法に沿った**変換規則**に基づいて行う。

従来のVPの問題点の1つは、表現規則の範囲でプログラミングをするため、対象となる問題領域が拡大するに従って表現規則が不十分となり、プログラミングが困難になる事である。よってVPが様々な問題に対処するために、問題に適した表現規則を適用することが望ましい。しかし、トランスレーターは内部に表現規則や変換規則に依存した機能が埋め込まれているため、汎用的に使うことができない。

そこで表現規則と変換規則を外部に持つトランスレーターを提案する。このトランスレーターでは、まず図形の間の空間的な関係を調べる。これはすべてのビジュアルプログラムに対して共通して行われる。次に外部の表現規則の情報に基づいて中間的なモデルを生成する。最後に変換規則を適用しプログラムを生成する。

トランスレーターはTclインタープリターに図形データを格納するデータベースと、それを操作する機能を追加して実現されているため（図1参照）、規則は簡単なTclの命令と拡張命令によって記述する。よって規則の変更が容易に行える。

2　ルールの記述方法

表現規則や変換規則の記述を**ルール**と呼ぶ。トランスレーターは内部に**辞書**と呼ばれるデータベースを持ち、ルールで書かれた命令によって操作される。辞書

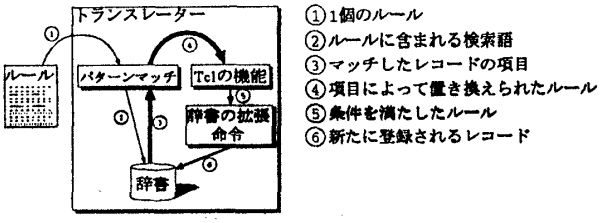

図 1: トランスレーターの内部

はレコードの集合で構成され、改行までを1つのレコードとする。レコードは空白で区切られる**項目**の集合により構成される。項目は以下の表記法を含む。

$$Item = Value$$

これは項目に値を与える表記法である。この表記法によりビジュアルプログラムを構成する図形は座標値、色、形などを項目とした1つのレコードで表される。

図2にデータフロー表現のビジュアルプログラムの例を示す。このプログラムはキーボードからの入力と他の文字列を連結し、表示をする。図2の破線で囲まれた図形を登録した辞書の内容を示す。(x1,y1) と (x2,y2) は図形を完全に囲む最小の四角形の左上、右下の頂点を表す座標である。

```
id=1 type=rectangle x1=150.0 y1=30.0 \
     x2=270.0 y2=62.0 width=2
id=2 type=text x1=158 y1=37 x2=263 y2=55 \
     -text {"My name is "}
}
```

項目の値を取り出すための表記法を次に示す。

$$@KeyWord\ (Item)$$

ルールの中にこの表記法が含まれると、*KeyWord* にマッチする全てのレコードの *Item* で指定された項目の値によってルールが置き換えられる。よっていくつかのレコードが見つかると、1個のルールが値の異なる複数のルールに置換される。

その他の辞書操作の命令には、レコードの登録をする `set@`、削除をする `delete@`、項目の一部が共通する複数のレコードを合成する `join@`、外部ファイルを読み込む `read@` などがある。

---

*A Translator with External Conversion Rules for Visual Programmings

†Ryuichi IWATA, Yoshikazu TANAKA, Minoru UEHARA, Hideki MORI
‡Department of Information and Computer Sciences, Toyo University

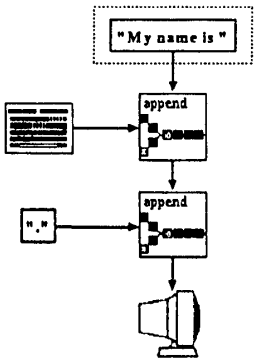

図 2: データフローの例

3 空間的な関係の認識

空間的な関係とは 2 つの図形の間の関係であり、Contain (内包する)、Enclosed (囲まれる)、Overlap (重なる)、Connect (接する)、Near (近い)、Faraway (遠い) などが調べられる。例えば前述の辞書の内容の Contain の関係を調べるルールは、次のように書ける。

```
if {@type=rectangle(x1) <= @type=text(x1) && \
    @type=text(x2) <= @type=rectangle(x2) && \
    @type=rectangle(y1) <= @type=text(y1) && \
    @type=text(y2) <= @type=rectangle(y2)} {
       set@ "ID=@type=rectangle(id) \
           Contain=@type=text(id)"
       set@ "ID=@type=text(id) \
           Enclosed=@type=rectangle(id)"
}
```

この結果から次のレコードが新たに登録される。

```
ID=1 Contain=2
ID=2 Enclosed=1
```

4 表現規則の記述

個々の図形間の関係から表現規則に基づいた中間モデルを構成する。例えば図 2では「rectangle に囲まれる」という条件を満たす text は、文字列を表すノードである。また「bitmap か rectangle に接続している」という条件が成り立つ line はデータの流れを表すエッジである。

よって図の"My name is "と、それに接続する append のモデルは次のようなレコードで登録される。

```
ID=2 String={"My name is "}
ID=4 Procedure=append
ID=5 Flow From=2 To=4
```

5 変換規則の記述

プログラミング言語の文法や命令に基づいて変換規則は記述される。例えば図 2を C 言語のプログラムにする場合、キーボードのアイコンは標準入力からのデータを文字列として返す Keyboard() 関数で置き換えられる。返されるデータ型をプログラムに応じて変えるためには、矢印の付近に型を書くか、型ごとに異なるアイコンを使用する必要がある。

キーボードのアイコンに対するプログラムは、変数の宣言と手続きの呼び出しが必要であるため、以下のように生成される。_に続く数字はアイコンの ID であり、変数名の重複を避けるために付けられている。

```
char Keyboard_3[80];
Keyboard_3 = Keyboard();
```

6 まとめ

本研究では外部の規則により変換を行うトランスレーターを作成した。従来、専用のソフトウェアを作成しなければならなかったが、本研究の方式ではルールという簡単な記述で行える。

現在、表現規則としてデータフロー、フローチャート、変換規則として C 言語、Tcl、私達が開発中のデータフローモデルの並行プログラミング言語 NET/C[3] の各ルールが記述されている。表 1 にそれぞれの規則の組合せで変換が可能かどうかを示している。○は可能であり、△は制限付きで可能である。

	C 言語	Tcl	NET/C
データフロー	○	○	○
フローチャート	○	○	△

表 1: 表現規則と変換規則の対応

今後の課題として、全てのビジュアルプログラムの解析に十分な空間的な関係を調べるルールの作成と、そのアルゴリズムの見直しが必要である。

参考文献

[1] Dan Ingalls, Scott Wallace, et al. Fabric (A Visual Programming Environment). In *OOPSLA '88 Conference Proceedings*, pp. 176–189. ACM SIGPLAN, September 1988.

[2] P.T.Cox, F.R.Giles, and T.Pietrzykowski. Prograph. In Adele Goldberg, Margaret Burnett, and Ted Lewis, editors, *VISUAL OBJECT-ORIENTED PROGRAMMING*. Manning Publications, 1995.

[3] Minoru Uehara. NET/C: Toward the Fine Grained UNIX-like OS. In *OOIS'94*, pp. 453–456. Springer-Verlag, December 1994.