3L－7

制御情報の埋め込みが可能な言語の作成とビジュアルプログラミングへの応用†

杉山 潤，越田 一郎†
東京工科大学†

1 概要

プログラムの構造に対応した制御情報を，プログラム内に埋め込むことができる言語を設計し，そのインタプリタを作成した．

今回作成したインタプリタ言語では，プログラムの表示形式やデバッグ時の制御情報などの他，関数呼出しを含む式を，プログラムの構造に結び付けることができる．結び付けられた情報は，プログラムの構造に対応した有効範囲を持っているので，コメント機能を利用した埋め込みとは異なる，より複雑な制御が可能である．

この機能の応用として，プログラムの表示方法をプログラム自身で変更する方法と，GUI を用いた開発環境を構築する方法について検討を行った．

2 インタプリタの特徴

2.1 構文

このインタプリタ言語では，C 言語に近い構文を用いており，数値演算などはC 言語と同様の演算子を使用することができる．

2.2 制御情報の埋め込み

一般に，プログラム内の制御情報は，特定の有効範囲を持っているため，その範囲を指定できることが望ましい．このインタプリタ言語では，それぞれの制御情報を，プログラムの構造に対応させることで，その有効範囲を指定することができる．そのための構文として，「プログラム情報指示子」があり，これによってif文やwhile文などに対する結び付けを行う．以下にプログラム情報指示子の書式を示す．

<ステートメント等># (*expression*)

ここで *expression* は任意の式であり，関数呼出しを含むことができる．

2.3 プリミティブ関数

アプリケーションを作成するための基本的な処理を，プリミティブ関数として用意する．プリミティブ関数の大まかなグループ分けを以下に示す．

- 基本的な処理
  C言語の標準関数と同様な処理を行う．ファイル処理や文字列処理などが含まれる．
- リフレクティブな処理
  プログラムの実行制御と構文木アクセスなどを行う．
- グラフィック
  2次元のグラフィック処理を行う．
- GUI 関連
  メニューやウィンドウ関連の処理を行う．

2.4 ライブラリ

ライブラリは，プリミティブ関数や他のライブラリ関数などを用いて作成した関数の集合であり，同じインタプリタ言語で記述されているので，プログラマが新たに追加することもできる．

3 ビジュアルプログラミングへの応用

3.1 プログラムの表示形式の変更

開発環境上でのプログラムの編集時の表示形式を，プログラムの記述によって変更できるようにする．また，プログラムの表示形式を部分的に変更するためには，表示形式の種類だけでなく，その範囲も指定する必要がある．このような範囲の指定は，プログラム情報指示子を用いて行う．

通常は，プログラムを表示する関数を定義しておき，*expression* の部分から起動するようにする．プログラムを表示する関数は，標準的なものをライブラリとして，あらかじめ用意しておき，その中から選んで使用する．

3.2 デバッグ情報の埋め込み

プログラムのデバッグを中断する場合，デバッグ時のブレークポイントや，インスペクトする変数などのデバッグ情報は，プログラムの中に埋め込んでおく必要がある．これはデバッグ情報とプログラム内容との食い違いを防ぐためであり，プログラムに埋め込まないと，デバッグを再開するまでの間，ソースプログラムを編集することができなくなる．

---

†A programming language with the function of imbued control infomation and the application to visual programming
†Jun Sugiyama , Ichiro Koshida
†Tokyo Engineering University,
1404-1, Katakura, Hachioji, Tokyo, Japan

プログラム情報指示子を用いれば，変数や関数・ステートメントなどのプログラムの構造ごとに，デバッグに必要な情報を結び付けることができる．これらの制御情報には，あらかじめ用意された制御をさせるための指令の他，プログラマが定義した関数を呼び出すための式も記述できるので，デバッグ時にのみ必要となるコードを実行させることが可能である．

3.3 GUI 部品の編集

ソースプログラムの表示形式の変更を容易なものとするために，ウィンドウやダイアログが手軽に利用できるようにする．これら GUI 部品の編集は，部品張り付けとリンク作成を基本とする編集によって行い，ウィンドウやボタンなどの各部品の配置だけでなく，プログラムの処理との結び付けが行えるようにする．

ダイアログの様に，プログラムの実行に伴って，ウィンドウ内の部品の表示が変化する場合，通常は表示の変化をもたらす部分に，再描画をするためのコードを記述する．この方法では，表示内容が変化するような部品を，部品張り付け型の編集で編集しても，実行時のイメージが把握できない (図 1)．

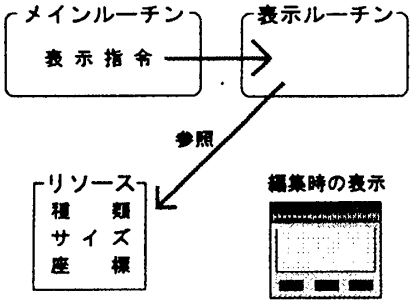

図 1. 単純な部品張り付け型編集

これを避けるため，表示内容を再描画するための関数を用意し，その関数を各部品に結び付けておく．表示の変化をもたらすステートメントに対しては，どの部品が変化するかという情報を付加する．この様にすると，部品張り付け型の編集時に，各部品の配置だけでなく再描画関数のテストも可能となる (図 2)．

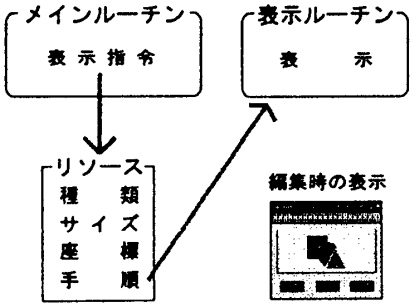

図 2. 表示ルーチンを含めた編集

4 プログラム例

編集時におけるプログラムの表現を変更するためのステートメントを含む，簡単なプログラム例を以下に示す．ここでは，while 文の実行回数を，デバッグ時にのみ，表示するようにしている．

```
1 func fibo(r,n)
2 #(Comment("(An) = (An-1) + (An-2)"))
3 {
4   #(Style(C, "initialize variables"));
5   put(i,n); put(p,0); put(q,1);
6
7   while(i > 1)
8   #(Style(PAD,"loop n/2 times"),
9     Debug(Print(LoopCount())))
10  {
11    put(p,p+q);
12    put(q,q+p);
13    put(i,i-2);
14  }
15
16  if(i == 0)
17  #(Style(DEFAULT,"check n is even"))
18    put(r,p);
19  else
20    put(r,q);
21 }
```

リスト 1. フィボナッチ数列を計算するプログラム

5 今後の展望

インタプリタ言語の基本的なコンセプトは，拡張性が高く，記述しやすい言語にすることであるが，プログラムの読み易さや構文の分かり易さなどに注意して研究を進めている．今後は，このインタプリタ言語のための開発環境を，インタプリタ言語のライブラリとして整備し，その際に必要と思われる言語機能やプリミティブ関数を追加していく．さらに，一般ユーザ向けの応用プログラムを実際に作成し，その拡張性やパフォーマンスなどについても，検証してみる必要がある．

参考文献

[1] Dylan Interrim Reference Manual, 1994

[2] John K. Ousterhout: Tcl and the Tk Toolkit, Addison-Wesley, 1994

[3] Smalltalk/V Mac Tutorial and Programming Handbook, digitalk inc.

[4] 所 真理雄, 松岡 聡, 垂水 浩幸: オブジェクト指向コンピューティング, 岩波書店, 1993