# 知的プログラム開発支援環境IPSEにおけるプログラム合成系[1]

3K－4

須河内寛　野上亜子　岩城明宏　廣田豊彦　橋本正明[2]
九州工業大学情報工学部[3]

## 1　はじめに

今日，ソフトウエアの開発や保守が重要な問題になっているが，ソフトウェア開発の生産性と質の向上のためには既存のソフトウエアやすでに存在する設計情報を再利用することによって必要な情報を得るのが有効であると考えられる．そこで筆者らは，プログラムの再利用を目的として，プログラムの処理内容に関する知識であるプログラムプラン[2]を用いた知的プログラム開発支援環境IPSEの開発をすすめている[1]．今回，ソースプログラムの合成を支援するシステムであるプログラム合成系を作成した．

## 2　知的プログラム開発支援環境IPSE

知的プログラム開発支援環境IPSEは，プログラムプランを蓄積しているプランライブラリを中心として以下のようなサブシステムから構成される．

**データフローモデル生成系** ソースプログラムの構文解析ならびにデータフロー解析を行い，その結果に基づいてソースプログラムのデータフローモデルを構成する．

**データフローモデル表示系** データフローモデルをディスプレイ上で図的に表示する．

**プログラム説明生成系** データフローモデルやプランライブラリに蓄積されているプログラムプランに基づいて，プログラムの意味や変数の使用状況などを説明する．

**プラン構成系** 既存プログラムから生成したデータフローモデルを元にプランを作成するための支援を行う．

**プログラム合成系** ユーザーが指定したプログラムプランを組み合わせ，ソースプログラムへと展開する．

プラン構成系をを除く各サブシステムはすでにプロトタイプを実現している．

## 3　プログラムプラン

プログラムプランとは，プログラムのデータフロー解析の結果をもとに作られたプログラムの断片と，それが表す処理内容の意味記述の対で表されたものである．

プログラムプランはプログラムの構造を表す構文木の葉に相当するターミナルプラン，部分木に相当する抽象化プランに分類される．

### 3.1　ターミナルプラン

ターミナルプランには，そのプランのプラン名，そのプランにおける定義変数名，使用変数名，使用定数名などが示されている．使用定数とはユーザーが定義する関数やデータの型名のように，ユーザーが任意に指定できるプログラムのパラメータのことである．これらはユーザーが定義することが出来る．

### 3.2　抽象化プラン

抽象化プランは，他の抽象化プランまたはターミナルプランを下位プランとして持つ．また，ターミナルプランと同様に，プラン名，定義変数名，使用変数名などを示す他，下位層のプラン名や下位層の定義変数名，下位層の使用変数名，下位層の使用定数も示す．これらには下位層の変数や定数の情報がすべて示される．

## 4　プログラム合成系

プログラム合成系はC++対応の，新たなプログラムを対話的に合成するシステムである．

ユーザーがプランライブラリから目的のプランをプラン名で選択し，それらを組み合わせてプログラムを合成する．プランの木構造はディスプレイ上に図示されるため，ユーザーはプログラムがどのような構造でできているかを容易に認識することができる．そして，関数名やクラス名などを入力してソースプログラムが出来上がる．できたソースプログラムは，ほぼそのままコンパイルできる．

このようにして，プログラム合成系はプログラミングの際に起こる煩雑さを軽減することによってプログラミングを支援し，知識の再利用を可能にする．

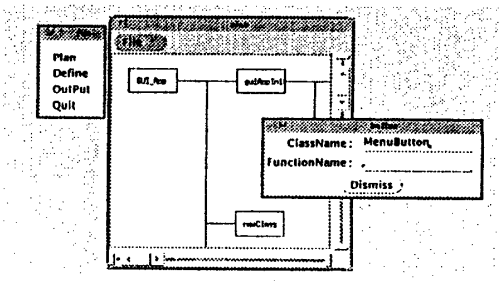


図 1: プログラム合成系

[1] Program Synthesizer of Inteligent Programming Support Environment

[2] Hirohi SUGAUCHI Ako NOGAMI Akihiro IWAKI Toyohiko HIROTA Masaaki HASHIMOTO

[3] Faculty of Computer Science and System Engineering Kyusyu Institute of Technology

### 4.1 プログラム合成の手順

次の手順でプログラム合成を行う.

(1) 入力されたプラン名をプランライブラリから検索する.

(2) プランが存在するならばそのプランがもつ下位プランも検索し, それらで構成される木構造を図示する.

(3) 必要に応じてプランを組み合わせる.

(4) 関数名, クラス名, ファイル名が入力されるとそのファイルにソースコードを出力する. ファイルは, 関数を出力する.cc ファイルと, クラスを定義する.h ファイルの2つを作る.

### 4.2 関数の作成

プランの組合せが終了し関数名が入力されるとプランを組み合わせた結果できるひとつの木構造を一関数とみなし, その名前の関数としてソースプログラムを作る.

ソースプログラムは, プランのスケルトンを展開することにより作成する. 抽象化プランとターミナルプランはそれぞれプログラム合成用のソースコードスケルトンを持っていて, これには各々のプラン中で行なうべき処理のコードが書かれている. プログラム合成系では, 出来上がったプランの木構造を最上位の抽象化プランから最下位のターミナルプランへと縦型探索をしてゆく. よってその探索順に, スケルトンに書かれているコードを指定されたファイルに出力する.

ほぼ完成した形式で出力できるが, インクルードファイルや変数の型, 引数などはユーザーが指定する必要がある. また, 下位プランが指定されていないようなプランに対してもユーザーが目的のコードを埋め込む.

本報告では, 合成されるソースプログラムの例としてウインドウプログラミング用のツールキットである OLIT を用いた GUI アプリケーションプログラムを使用した. 図2に, メニューボタンを作成する関数の出力例を示す.

```
MenuButton::MenuButton( Widget parent ){
  Arg wargs[3];

  menubutton = XtCreateManagedWidget( "File",
     menuButtonWidgetClass, parent, NULL, 0 );
  int n=0;
  XtSetArg( wargs[n], XtNmenuPane, &menu_pane );
  n++;
  XtGetValues( menubutton, wargs, n );
  for( int i=0; i<XtNumber(button_names); i++ ){
    oblong[i] = XtCreateManagedWidget(
                        button_names[i],
                        oblongButtonWidgetClass,
                        menu_pane, NULL, 0 );
    XtAddCallback( oblong[i], XtNselect,
                (XtCallbackProc)&button_callback,
                button_names[i] );
  }
}
```

図2 関数の出力例

### 4.3 クラスの定義

ソースプログラムは C++ で表現されているので, プログラムをコンパイルするためにはクラスの定義が必要である. 作成するクラスは, 先に作った関数をメソッドとして持ち, クラス名はユーザーが指定する.

プランを組み合わせた結果, 最上位のプランにはそれ以下で用いられる変数や関数が伝搬されているので, 最上位のプランからプログラムで使用または定義されている変数名と関数名とを区別して読みとり, クラス定義の構文に従って出力する.

変数はすべてクラス内に定義されるが, ある関数内だけのローカルな変数名も存在し得ると考えられる. この様な場合はユーザーが手直しを行う. 変数の型などもユーザーが指定する.

図2で示した関数をメソッドとしてもつクラスの出力例を図3に示す.

```
class MenuButton {
 private:
  Widget menubutton,
         menu_pane,
         oblong[ XtNumber(button_names) ];
  static void button_callback( Widget,
                    XtPointer, XtPointer );
 public:
  MenuButton( Widget );
  ~MenuButton();
};
```

図3 クラスの出力例

## 5 おわりに

本報告では IPSE におけるプログラム合成系について述べた. プランを用いてプログラム合成を行なうことにより, プログラム開発の効率化が期待できる. 今後はプランライブラリの充実に伴い, 様々なプログラムの合成の研究に努める予定である.

## 参考文献

[1] 廣田豊彦, 橋本正明. データフローモデルに基づく知的プログラム開発支援環境. 電子情報通信学会技術研究報告 (知能ソフトウェア工学), Vol. 93, No. 30, pp. 17–24, 5 1993.

[2] W. L. Johnson and E. Soloway. PROUST: Knowledge-based program understanding. *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 3, pp. 267–275, March 1985.