5D-8

クラス記述と要求仕様記述を独立させた オブジェクト指向型C言語プリプロセッサの開発

増田 佳弘　　間野 浩太郎
(　青山学院大学　理工学部　経営工学科　)

1．はじめに

手続き型言語（C言語）にオブジェクト指向型言語の「クラス」とその継承機構の概念をプリプロセッサの形で組み込むことによりプログラムの部品化・再利用を実現しようという試みが行なわれている。[2][3]

しかし、従来のオブジェクト指向型言語の持つクラスの機構によりプログラムを共用・再利用しようとする場合には、要求仕様に基づいて作られる処理や私的な処理など、部品としての性格の薄い処理も何らかの「クラス」に属するメソッドとして定義する必要が生じる。したがって、部品として定義したメソッドと要求仕様から定義されるメソッドが同一クラス内に混在することになり、このことは不必要に個々のクラスの大きさを増大させるだけでなく、クラスを純粋な部品として再利用することを困難にしてしまう。　そこで、我々は、次のような考えに基づいたクラスの再利用を可能にするためのオブジェクト指向型C言語プリプロセッサ、およびその支援処理系をunix上で開発中である。

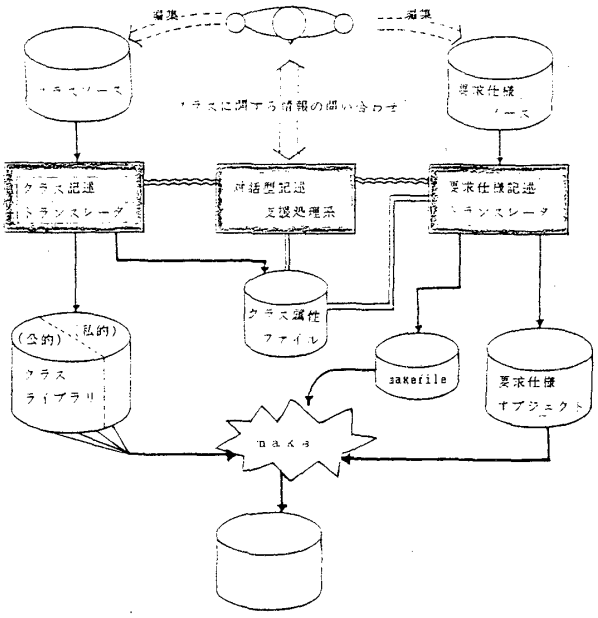
(1) 要求仕様に基づいた処理の記述は「要求仕様記述言語」というクラス定義とは別の言語で行なう。そして、共用クラスはユーザ共有のディレクトリ下に定義しておき、部品として共用できるようにする。
(2) 「要求仕様記述言語」を用いて行なう要求仕様記述は基本的に既定義のクラスのみで行なう。このことによって、要求仕様記述言語におけるクラスの定義は不要となり、基本的にはインスタンスの生成（「道具立て」）とそれへのメッセージング（「その操作」）で要求仕様を表現できる。
(3) (1) で定義するクラスとは別に処理固有のクラスや私的なクラスを定義する必要がある場合には各ユーザ所有のディレクトリ下で作成する。尚、インヘリタンスの対象となるクラスは(1) で定義されるクラスを含め、unix上で個々のユーザがアクセスできるファイル内で定義されたすべてのクラスである。
(4) クラスの増加や第三者による利用を考慮し、(1) 、(3) のように定義したクラスやメソッドについてのユーザインタラクティブな検索機構を設ける。

本稿では、本処理系のシステム構成、各言語の概要、クラス検索の方法、ならびに、開発現状について述べることにする。

2．システム構成

本システムは（図１）に示すような、３つの処理系で構成されている。

（図１）　システム構成



ここで、「対話型記述支援処理系」は本システムの中核となる処理系であり、次のような機能を備えている。

(1) 対話型のクラス検索（後述）
(2) クラスに関する情報（説明記述、型属性、カテゴリ、etc.）の参照
(3) unixコマンドの実行（viによる要求仕様記述の編集や catによるクラス記述の表示など。）
(4) 「クラス記述トランスレータ」の起動
(5) 「要求仕様記述トランスレータ」の起動

「クラス記述トランスレータ」は「クラス記述言語」によって記述された部品プログラムをクラス単位でC言語コードへ変換し、それと同時に部品プログラム内で記述されたクラスに関する属性情報を「クラス属性ファイル」に出力する。尚、この情報は上記 (1)、(2) で用いられる。また、「クラス記述言語」内でスーパクラスを指定する場合には、そのクラスの定義されているファイルをパス名によって指定することができるため、クラスをディレクトリによって管理することが可能である。

「要求仕様記述トランスレータ」は「要求仕様記述言語」をmain関数を含むC言語コードに変換すると同時に、部品として用いたクラスと結合するためのmakefileを出力する。

このようにして、最終的にはunixのmakeコマンドの実行により、「要求仕様記述言語」から変換されたプログラムとそこで用いられたクラスが結合され、目的とする実行形式のプログラムを得ることができる。

Object-Oriented C Language Preprocessor dealing with Class Description and Required Specification Description Separately

Yoshihiro MASUDA, Dr. Koutaro MANO
AOYAMA GAKUIN Univ.

3．クラス記述言語

「クラス記述言語」の構文の基本構成要素は“クラス定義”であり、これは通常のＣ言語の構文にオブジェクト指向型言語のクラス、インヘリタンス、メッセージングなどの機構を付加したような形をしている。継承には多重継承を許しており、多重定義の場合に継承を上位クラスから開始させるための疑似変数super は次のメッセージングのようにクラスを明示して行なう。

```
[ super.class_name  <==  selector( key1:arg1 ... ) ] ;
```

また、クラスの定義は次のような形式をしている。尚、ここで下線で示した部分は必須のものであり、これらはクラス属性データとして「クラス属性ファイル」に保存され、クラス検索の際に「対話型記述支援処理系」から参照される。

```
class  class_name  <path> super_class1, ...   [ class_category1, ... ]
        (1) クラス名      (2) スーパクラス名          (3) クラスカテゴリ
 %%   ～   %%
(4) クラスに関する仕様記述
{
  char  * s ;
  instance  class_1  inst1 = { key1:120 } , ... ;
  int   a[ a_size:$length ]   ;
          (5) インスタンス生成のためのキーワード
    ...

    method  ( char * )  method_name( key1:arg1, ... ) [ category1, ... ]
            (6) 返す値の型  (7) メソッド名  (8) 引数キーワード      (9) メソッドカテゴリ
    %%  ～  %%
    (10)メソッドに関する仕様記述
    char*  arg1 ;          %%   ～   %%
    (11)引数の型           (12)引数に関する説明記述
        ...
    {

        メソッド本体

    }

}
```

4．要求仕様記述言語

「要求仕様記述言語」は、基本的には、「インスタンスの宣言」と「メッセージングの記述」で記述される。このうち、「インスタンスの宣言」はいわば“道具立て”であり、次のように宣言することによってインスタンスが自動的に生成される。

```
instance   <path> class_name  instance_name = { key1:size1, ... }
```

ここで、class_nameはクラス名、<path>はそのクラスのソースファイルのパス名、instance_name はインスタンス名、key1:size1, ... はクラス宣言中で“key1:$～”の形で与えられたインスタンス生成に関する疑似パラメータ（前述のクラス定義における $length）に対する実際の値の指定である。（key はクラス定義内で与えられたキーワード。size1 は定数あるいは定数シンボル。）　また、ここではインスタンスとそれを用いたメッセージングをまとめて「シナリオ」と呼ぶＣ言語の関数呼び出しに似た形で記述し、インスタンスのスコープを限定してその生成・解放を行なうことによって記憶域の管理を行なっている。このことにより、自動的なガベジコレクションを行なわずにある程度の記憶域の節約を可能にすることができる。

5．対話型クラス検索の方法

「クラス記述言語」でクラスを定義する場合、クラスやメソッドには「カテゴリ」という範疇化のための概念属性を複数個持たせることが可能であり、この情報は「クラス属性ファイル」を通して「対話型記述支援処理系」から参照される。そして、プログラマは「対話型記述支援処理系」上で次のような手順に従ってクラスおよびメソッドの検索を行なう。

(1) 「クラスカテゴリ一覧」を参照する。（ルートノードからアクセス可能なクラス全部について表示される。）

(2) キーワードと思われる複数のカテゴリについてクラスの集合和および集合積を求める。ここで、集合積とは“候補クラスを絞る”ことを意味し、集合和は“候補クラスの範疇を広げる”ことを意味する。例を以下に示す。

例：「通信デバイスを扱うクラス、または、通信プロトコル（OSI参照モデル）のデータリンク層を扱うクラスを列挙せよ。」

```
> CLIST  comm-device-obj | ( comm-protocol & data-link-entity )
```

(3) (2) で得られたいくつかの候補クラスに対してその詳しい属性（説明記述、スーパクラス、etc.）を参照し、要求を満足するクラスを見つけ出す。

メソッド検索についても同様に行なう。指定の例を以下に示す。

例：「単純、かつ高速にファイルをソートするメソッドを列挙せよ。」

```
> MLIST  file  sorting & simple & high-speed
```

6．おわりに

以上、オブジェクト指向言語のプログラムをクラス記述と要求仕様記述に分けることによってクラスを純粋な部品として作成・保守・再利用する方法について述べてきた。

開発に関しては、これまでに、各言語の構文則の設計と変換イメージでのシミュレーション、および、「対話型記述支援処理系」の「クラス記述トランスレータ」「要求仕様記述トランスレータ」を除いた部分の開発が完了しており、現在、各言語処理系をyaccとＣ言語によって開発中である。

尚、将来は「対話型記述支援処理系」に知識ベースを組み込み、非常に高い抽象度の仕様から対話式に詳細化を行なってプログラムを生成することを可能にする処理系を開発することも考えている。

【参考文献】

[1] A. Goldberg and D. Robson ,
Smalltalk-80 The Language and Its Implementation,
Addison Wesley, 1983

[2] Brad J. Cox, "The Object-Oriented Precompiler
- Programming Smalltalk-80 Methods in C Languages,"
ACM Sigplan Notices, Vol.18, No.1, Jan.1983, pp.15-22

[3] Brad J. Cox, "Message/Object Programming :
An Evolutionary Change in Programming Technology,"
IEEE SOFTWARE, Jan. 1984, pp.50-61

[4] B. Stroustrup, The C++ Programming Language,
Addison Wesley, 1986

[4] Objective-C Reference Manual Ver.3.0, PPI, 1984