多彩なメッセージ表現が可能なオブジェクト指向言語Koola

3P-1

渡守武 和記[1] 西山 保[2]
1松下電子工業(株) 開発推進センター 2松下電器産業(株) 半導体研究センター

1. はじめに

半導体CADソフトウェアの開発基盤となるオブジェクト指向プログラミング言語Koola(Kernel Object Oriented LAnguage)を開発し、その処理系を作成した。本稿ではKoolaの特長と、一般的な表現から自然言語に近い表現まで多彩な表現が可能である柔軟なメッセージシンタックスについて報告する。

2. 開発方針

大規模なCADソフトウェアの開発言語として、
①開発効率をあげる高い記述性
②モジュールの高い再利用性
③実行スピードの高速性
を条件に、C言語をベースとしたオブジェクト指向言語という方針で開発を行った。

3. 特長

3.1 C言語のスーパーセット

Koolaはクラス定義をC言語のソースプログラムに変換するプリプロセッサ方式のオブジェクト指向言語である。そのため既存のプログラムと簡単にリンクでき、C言語との相性が非常によい。文法的にはファンクションコールの代わりにメッセージの記述を可能にしたC言語のスーパーセットであり、C言語のもつ豊富な制御構造・オペレータ等がそのまま使用できる。したがって記述性が高く、また導入の教育が殆ど要らない。

3.2 完全スタティックバインディング

Koolaにおけるメッセージとメソッドの結合は全てプリプロセッサが解決し、メッセージはメソッドに対応するファンクションの呼び出しに変換される。つまり実行時にメソッドを探すダイナミックバインディングは一切使用していない。このため実行スピードが速く、同等のプログラムをC言語で直接書いたものと比べ何ら遜色がない。

3.3 Smalltalk-80[1]準拠のクラス定義

KoolaはSmalltalk-80準拠のクラス定義を主体としており、クラスのライブラリ化・再利用性を考慮したフォーマットが厳密に決められている。このためプログラミングスタイルの統一が自然に行え、可読性が非常に高く、保守も容易である。

3.4 簡単なコンパイル

Koolaでは1クラスを1ファイルに記述する。これをプリプロセッサでコンパイルすると、プログラムファイルとヘッダファイルとが得られる。このプログラムファイルをCコンパイラでコンパイルすればオブジェクトファイルが得られる。この時のエラーメッセージは元のクラス定義の行番号で示されるようになっており、エラーの修正が行いやすくなっている。またコンパイルは上位クラスから行う必要があるが、UNIXではKoolaのユーティリティを用いてmakefileを作成することにより、makeコマンドで全てのコンパイルを簡単に行える。

4. シンタックス

4.1 クラス定義

図1にクラス定義の例を示す。ClassNameなどのキーワード以外は必要な部分だけを記述する。

4.2 変数の宣言

インスタンスを格納する変数は、そのインスタンスの属するクラスへのポインタで宣言する。

4.3 メソッド定義

Koolaでは、オブジェクトの後にセレクタ・引数

```
ClassName
  NodeWithLevel
SuperClass        (継承するときのみ1クラスだけ指定)
  Node
ReferenceClass    (他に参照するクラスの宣言)
  StackOfInteger
Definition        (#include文・#define文・構造体の宣言等の記述)
ClassVariable
InstanceVariable
  StackOfInteger    *levelStack;
ClassMethod
  NodeWithLevel    *[ new `NodeWithLevel ]
  {  return [ initialized [ new super ] ];                    }
InstanceMethod
  void    [ Destroy `NodeWithLevel ]
  {  Destroy levelStack;
     Destroy super;                                           }

  void    [ Push `level onto `NodeWithLevel ]
  int     level;
  {  Push level onto levelStack;                              }

  int     [ last level of `NodeWithLevel ]
  {  return [ last item from levelStack ];                    }
PrivateMethod    (他のクラスからは見えないメソッドの定義)
  NodeWithLevel    *[ initialized `NodeWithLevel ]
  {  levelStack = new StackOfInteger;
     return self;                                             }
PrivateFunction  (他のクラスからは見えないファンクションの定義)
EndOfClass
```

図1 クラス定義例

Koola: An Object Oriented Language and its Flexible Message Syntax
Kazunori TOMOTAKE[1], Tamotsu NISHIYAMA[2]
1 Matsushita Electronics Corporation, 2 Matsushita Electric Industrial Co.,Ltd.

を並べた形式だけではなく、オブジェクト･セレクタ･引数が混在した形式をメッセージと呼んでいる。つまり、オブジェクト･セレクタ･引数を自由な順番で並べたメソッドを定義できる。

メソッド定義は図1に示したようにC言語のファンクション定義に似た方法で記述する。[ ]の中にオブジェクト･セレクタ･仮引数を自由な順序で並べる(これをメソッド名と言う)。オブジェクトとして自分のクラス名を使用する。セレクタにはクラス名以外ならどんな単語を使用してもよい。またオブジェクトと仮引数の前には「`」を付ける。

4.4 メッセージ

メッセージはメソッド名におけるオブジェクトと仮引数の所に式を書いたものである。セレクタを明示するために、セレクタの後ろに「:」を付けてもよい。メッセージは普通[ ]で囲むが、入れ子の最も外側にあり、他の式と紛らわしくないときは[ ]を省略できる。

5. 多彩なメッセージ表現

Koolaは、4.で説明した柔軟なメッセージシンタックスにより、様々な表現形式のメッセージが記述可能である。その一例を図2に示す。

①はオブジェクトの後にセレクタ･引数を並べた記述例である(inputStream, inputToken, wordSetがオブジェクトである)。これはオブジェクトが先頭にあるので、プログラム構造の解析等が行いやすい形式と言える。

②はオブジェクトをメッセージの中に埋め込んだ記述例である。これはwhileやifなども含めてすらすらと読め、プログラムの動作やアルゴリズム等が理解しやすい形式である。この場合、オブジェクトがどれかなどと気にしながら読むのではなく、文全体の意味を考えながら(つまり普通に)読んで行くのがよい。

この形式はKoolaでの標準的なプログラミングスタイルに決められている。例えば値を返すメッセージは小文字で始まり、返さないものは大文字で始まるメソッド名で定義する。図1で示したクラス定義例もこの形式で記述されている。

③は②をさらに自然な英語に近づけた記述例である。メッセージの途中に大文字や[ ]が出てくることを除けば、普通の英語として読める。このようにKoolaでは英文による疑似コード的プログラミングが可能である。

①Smalltalk-80的記述

```
while( ![inputStream isEmpty] ){
    inputToken = inputStream next;
    if ( ![wordSet includes:[inputToken string]] )
        wordSet put:[inputToken string];
    inputToken free;
}
```

②簡略化英語的記述

```
while( ![inputStream is empty] ){
    inputToken = next token from inputStream;
    if ( ![[string of inputToken] is in wordSet] )
        Put[saved string of inputToken] into wordSet;
    Free inputToken;
}
```

③英語的記述

```
while(theInputStream is not empty){
    theInputToken = a next token from theInputStream;
    if ([the string of theInputToken] is not in theWordSet)
        Put[the saved string of theInputToken] into theWordSet;
    Free theInputToken;
}
```

④日本語記述(参考)

```
while(入力ストリームが空でない){
    入力トークン＝入力ストリームにおける次のトークン;
    if(単語集に[入力トークンの文字列]が無い)
        単語集に[入力トークンの保存済文字列]を加える;
    入力トークンを解放する;
}
```

図2 メッセージ記述例

6. 日本語への対応

5.で述べたように、Koolaでは自然な英語によるプログラミングが可能である。これは記述性および可読性においては非常にすぐれている。しかし一般のオブジェクト指向言語のようには、オブジェクトがメッセージの先頭に来ないことが多いので、プログラムを解析･修正しにくい嫌いがある。また日本人にとっては英語よりも日本語で記述した方がプログラミングしやすい。

この観点から現在Koolaの日本語化を行っており、これによると図2④のような表現が可能である。これはオブジェクトが先頭に来る自然言語的記述であり、記述性･可読性･保守性に優れたプログラミング言語になり得る。

7. 評価

C言語で書かれたプリプロセッサ自体をKoolaに書き換えたところ、記述量としてはほぼ同等であった。デバッグでは、変換結果のC言語のプログラムが表示されるものの、元のクラス定義の行と1対1に対応しているので何ら支障はなかった。しかし一般のユーザのためには、クラス定義のソースのままでデバッグできるデバッガの開発が必要である。今後Koolaを多くのアプリケーションの開発に使用し、定量的な評価及び他言語との比較等を行ってゆく。

8. おわりに

Koolaは半導体CADに限らず一般のアプリケーションで使用できる。日本語化とクラスライブラリの充実により、広範な方面への展開を図って行きたい。

参考文献

1) Adele Goldberg and David Robson: Smalltalk-80 The Language and its Implementation.