3N-5

Adaプログラム分離コンパイル支援ツール ADAMAKE

松野了二* 平原正樹** 荒木啓二郎** 牛島和夫**
(*有明工業高等専門学校 **九州大学工学部)

1. はじめに

プログラミング言語Ada[1]はモジュール化機能を持ち，また，分離コンパイル機能を備えている。これらの機能は、複数プログラマによる作業分担を可能とし、また、再コンパイルに関する費用を削減できるなど大規模なソフトウェア開発には欠かせない。

一方，分離コンパイル機能を生かすために、各コンパイル単位を別々のファイルに格納すると、次のような問題点が生じる。

・通常のエディタでは、一度に一つのファイルしか見えないので、全体を見通すことができず、編集作業がやりづらい。

・Adaプログラムのコンパイル順序には厳密な規則がある。プログラムが大規模になると、コンパイル順序の決定作業が煩雑となり、コンパイル順序の誤りが生じやすくなる。また、不必要なコンパイル単位を見いだすことが困難になり、無駄なコンパイル作業が増える。

我々は、これらの問題を解決するために、翻訳順序の決定から翻訳までを自動的に行う機能を備えた分離コンパイル支援ツール ADAMAKEの開発実現を進めている[2]。ADAMAKEはフロントエンドとして構築され現存のコンパイラシステムに対しプリプロセッサのように振舞う。利用者は、ADAMAKEと対話しながらプログラミング、及び、コンパイル作業を進める。ADAMAKEは利用者の要求に従い依存情報など必要な情報を提供する。本稿では、ADAMAKEの設計方針と実現について述べる。

2. ADAMAKEの設計方針

分離翻訳支援ツールADAMAKEの設計方針を以下に述べる。これは、文献［5］で示した手法を踏襲している。

（1）複数のファイルに分割されていることを感じさせないこと。

・利用者インタフェースとして、画面エディタEmacs[3]を使用する。なぜなら、Emacsは複数の窓を持ちプログラム可能であるからである。利用者は複数の窓を通して、いくつかのコンパイル単位、例えば、パッケージの仕様部、本体、あるいは、参照している他の仕様部などを一度に見たり、修正したりできる。

・コンパイルは自動的に行う。Make[4]が用いるような依存関係の記述はしなくてもよい。

（2）ファイル管理にはコンパイル単位名の構造を反映すること。

・ファイル名には仕様部，本体が識別できるような名前を用いる。また、副単位には separate 文に用いる親単位名の後ろにピリオド、副単位名をつけた名前（完全拡張名）を利用する。Adaプログラムの構造はツリーの形で表現でき、完全拡張名を用いることで利用者はファイル間の親子関係を把握する必要がなくなり、また、各コンパイル単位間をあたかもディレクトリ間を移動するように扱える。

（3）再コンパイルは必要最小限にとどめること。

・あるコンパイル単位の修正は、それに依存している他のコンパイル単位の再コンパイルも引き起こす。一つのファイルに一つのコンパイル単位を格納することによって再コンパイルの数を最小にできる。但し、利用者が新しいプログラムを開発する場合には一つのファイルに複数個のコンパイル単位が含まれてもよく、また、ファイル名は任意でよい。ADAMAKEを利用した時点でこれらのファイルは自動的に分割され、ファイル名には完全拡張名が付けられる。

（4）分割コンパイルを積極的に支援すること。

・編集直後に構文解析を行うので、利用者は、ADAMAKEに問い合わせることにより、識別子の相互参照関係や、コンパイル単位間の依存関係を得ることができる。また、構文上の誤りが検出された場合は自動的に誤りのある行にカーソルが置かれ修正作業を助ける。

3. ADAMAKEの概要

ADAMAKEの概略図を図1に示す。利用者がプログラムを作成、または、修正を行うと、ソース解析部が構文検査、依存情報の抽出を行う。さらに、利用者が作成したファイルに複数個のコンパイル単位が含まれているならばコンパイル単位毎に分割する。この段階で、構文上の誤りが検出されたならば再びエディタに戻される。構文的に正しければ抽出された依存情報はデータベースに蓄えられる。利用者がコンパイル指令を出すとソース管理部はデータベースに蓄えられた情報を基に、コンパイル可能であるかどうかの検査を行い、可能であればコンパイル順序を決定し、Adaのコンパイラに引き渡す。この段階で、プログ

ADAMAKE: A tool for supporting separate compilation of Ada programs.

Ryoji MATSUNO*, Masaki HIRABARU**, Keijiro ARAKI**, and Kazuo USHIJIMA**
* Ariake National College of Technology ** Kyushu University

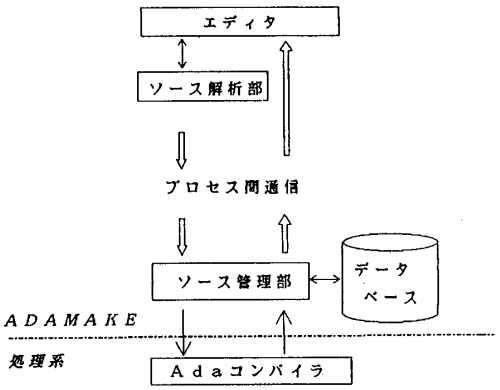

図１　ＡＤＡＭＡＫＥの概略図

ラムに誤りが発見されたならば、それに依存しているコンパイル単位はコンパイル対象から取り除き、コンパイル可能な残りの部分の処理を続行する。全ての部分の処理を終えた後、誤りが指摘されたコンパイル単位は、再びエディタに戻される。

```
ADAMAKE V1.03  9-Dec-1988  08:43:37

test.tss.x1.ada: NO change
t1.ada: Processing
  Unit: t1.Body.ada
  Unit: t2.Body.ada
  Unit: t3.Body.ada
sss.ada: Processing
  * Warning: [ t1.Body.ada ] already exists.
    Do you want to reset(y/n)? y
-----  test.ada : Syntax Error  -----
"./test.ada", line 7:  at or near .


Library Unit : CHAR_TEST
Library Unit : SIEVE
** Warning] File not found ( SIEVE.BEGIN_SIEVING )
Library Unit : SIEVE_DISPLAY
** Warning] File not found ( SIEVE_DISPLAY.DRAW_BOX )
Library Unit : T1
Library Unit : T2
Library Unit : T3
Library Unit : TEST
** Warning] File not found ( Body: TEST )
** Warning] The library unit  not found  or illegal( SIEVE_DISPLAY
** Warning] The library unit  not found  or illegal( SCREEN_IO )
** Warning] A circularity exists in the dependence relations.
t1.Body.ada        t3.Body.ada        t2.Body.ada

Compile  sieve_display.Spec.ada  ... OK
Compile  sieve_display.Body.ada  ... OK
Compile  sieve_display.write_header.ada  ... OK
Compile  sieve_display.write_trailer.ada  ... OK
Compile  sieve_display.new_box.ada  ... OK
Compile  sieve_display.write_value.ada  ... OK
Compile  sieve_display.erase_value.ada  ... OK
Compile  sieve_display.write_exception.ada  ... OK
Compile  char_test.Body.ada  ... OK

**ERR] Command not found
AdaMake(DebugMode):
```

図２　処理例

４．ＡＤＡＭＡＫＥの実現

現在までに、依存関係の抽出から、コンパイルまでの自動コンパイル機能はすでに実現済みであり、ＡＤＡＭＡＫＥ自身のデバッグや拡張作業にも利用している。（但し、Ｕｎｉｘのシェル環境下で動作し、依存関係の抽出はプログラムの編集直後ではなく、利用者がコンパイル指令を出したときに行う）

これらの機能を実現するに当たって、ソース解析部はコンパイラの自動生成系を用いて作成された既存のＡｄａ構文解析プログラム*を利用して作成した（プログラムサイズは注釈部を含め約２１００行）。　また、その他の部分はＡｄａを用いて作成した（同約２４００行）。

利用者インタフェースの部分がまだ未完成であり、現在、Ｅｍａｃｓ上でＥｍａｃｓ　Ｌｉｓｐを用いて作業中である。

なお、ＡＤＡＭＡＫＥの自動コンパイル機能を用いてＡＤＡＭＡＫＥ自身のＡｄａで記述した部分（上に述べた通り、約２４００行、ライブラリ単位数　６個、コンパイル単位数２１個）を処理した場合の処理時間は実時間で、依存情報抽出部が約１６秒、依存関係グラフ作成からコンパイル順序の決定、及びコンパイル開始までが約８秒である（Ｓｕｎ－３／２６０；ＯＳ３．４で測定）。

プロトタイプということから字句解析部には既存のＬＥＸで書かれたプログラムを利用しているのでこれを書き直すことにより、処理時間については、さらに改善できると思われる。

５．処理例

図２に、ＡＤＡＭＡＫＥのｃｏｍｐｉｌｅ命令を用いて処理した例を示す。この処理例に含まれる情報の主なものを以下に示す。

・ｔ１．ａｄａには”ｔ１”，”ｔ２”，”ｔ３”という名前の３個のコンパイル単位（いずれも本体）が含まれていて分割された。

・ライブラリ単位は７個あり、”ＳＩＥＶＥ”、”ＳＩＥＶＥ＿ＤＩＳＰＬＡＹ”，”ＴＥＳＴ”には未作成のコンパイル単位がある。

・”ｔ１”，”ｔ２”，”ｔ３”間の依存関係には循環性がある。

６．おわりに

ＡＤＡＭＡＫＥの設計方針と実現について述べた。利用者インタフェースの部分を早急に完成し、実際に使用経験を積み不備な点の発見、機能の追加を行っていきたい。

参考文献

１）U.S.Department of Defence:Reference Manual for the Ada Programming Language, ANSI /MIL - STD - 1815A(1983).

２）松野、平原、荒木、牛島：”Ada プログラム分離翻訳支援ツール ADAMAKE の実現”、昭和６３年度電気関係学会九州支部連合大会講演論文集、pp.571(1988).

３）Stallman,R.M.:"GNU Emacs Manual," Free Software Foundation,1987.

４）Feldman,S.:"Make- A Program for Maintaining Computer Programs," Softw.Pract.Expr., vol.9,No.3,pp.255-265,1979.

５）平原、荒木：”モジュラプログラミングを支援するＮａｎｏ－２コンパイラ”、情報処理学会第３４回全国大会予稿集、pp.947-948,1987.

* Litton Data Systems の Herman Fischer によって書かれた。