Vol. 29 No. 12 情報処理学会論文誌 Dec. 1988

ショートノート

Adaタスクに一意的名前を実行時に付ける方法†

程 京徳†† 牛島和夫††

Ada並列プログラム中の各タスクが実行時に自分自身を特定できるようにしたいという要求は，Adaを用いて並列プログラムを開発する際に生じた要求の一つである．本論文では，この要求に対して，従来提案されている解決方法の問題点を指摘した上で，我々がAda並列プログラムの事象駆動型実行モニタEDENを開発する際に考案し実現した新しい解決方法を述べる．本方法の主な特徴は，起動中のタスクも自分自身を特定することができることや，タスク間の起動開始あるいは実行文系列の実行開始の時間的順序にどんな制限も加えないことである．

1. はじめに

Adaは，大規模実時間ソフトウェアを記述する共通言語として設計されたプログラミング言語である[1),2)]．Adaでは，並列処理単位（タスクと呼ぶ）を型（タスク型と呼ぶ）として静的に宣言し，一つのタスク型に対して，実行時に多数のタスクを生成することができる．これによって，Adaを用いて並列プログラムを開発する際に，プログラム中の各タスクが自分自身を実行時に特定できるようにしたいという要求が生じる[3)~7)]．例えば，タスクを要素とする配列を宣言した際に，各タスク要素は，自分自身がその配列中の「どれ」であるかを実行時に特定したいことがある[3),4)]．同じことは，同一のタスク型を持つ幾つかのタスクが割り当て子の評価により動的に生成された時にも生じる．

もっと複雑な例としては，Ada並列プログラムの挙動を監視する実行モニタをAdaを用いて開発する際に生じる要求を挙げることができる．実行モニタは，被監視プログラムの実行中にどんな事象が発生したかを追跡し各タスクの挙動とタスク間の相互作用とをソース・テキストのレベルで利用者に報告する必要がある．そのために，被監視プログラム中の各タスクに，その実行中にどんな事象が発生したかを報告させなければならない[5)~7)]．この時にも被監視プログラム中の各タスクが自分自身を実行時に特定する必要が生じる．

しかしながら，Adaにはタスクが自分自身を実行時に特定できる手段について何も提供されていない．したがって，プログラマは何らかの手段でタスクにそれ自身が参照できる一意的名前を実行時に付ける方法を考案しなければならない[3)~7)]．タスクの一意的名前とは，タスクがその起動（すなわち，タスク本体の宣言部の確立）から終了までの生涯に持っておりかつ他のタスクと区別できる「名前」である．

本論文では，以上の要求に対して，従来提案されている解決方法の問題点を指摘した上で，我々がAda並列プログラムの事象駆動型実行モニタEDEN[8),9)]を開発する際に考案し実現した新しい解決方法を述べる．本方法の主な特徴は，起動中のタスクも自分自身を特定することができることや，タスク間の起動開始あるいは実行文系列の実行開始の時間的順序にどんな制限も加えないことである．

2. 従来提案されている方法とその問題点

Adaタスクに一意的名前を実行時に付けることについて幾つかの方法が既に考案されている[3),5)]．Boochの方法[3)]は，各タスク型に新しいエントリ（エントリ名SET_ID）を導入し，実行時に，各タスクを作り出すプログラム単位（マスタと呼ぶ）が各タスクのエントリSET_IDを呼び出してその一意的名前を与えるものである．しかし，この方法には，二つの重大な問題点があるので，Ada並列プログラムの実行モニタに適用することができない．

一つは，この方法を用いると，起動中のタスクは一意的名前を持ちえないので自分自身を特定することができない．なぜなら，タスクはその起動中に自分のエントリへの呼び出しを受け付けることができないから

† Naming Ada Tasks at Run-Time by JINGDE CHENG and KAZUO USHIJIMA (Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University).

†† 九州大学工学部情報工学科

である[1]．Adaでは，タスクは，その宣言部の演算対象の初期値設定中に関数副プログラムを呼び出し，さらにこの関数副プログラムの実行中にエントリを呼び出すことができる[1]．すなわち，タスクはその起動中にも他のタスクと通信することができる．けれども，Ada並列プログラムの実行モニタは，上記の方法を用いると，対象プログラム中のタスクの起動やタスク起動中の通信などの挙動を監視することができない[5),10),11)]．

もう一つは，この方法を用いると，タスク間の相互作用の時間的半順序関係を破壊する恐れがある．Adaの規定によれば，あるマスタに依存する各タスクの起動は並列的になされる[1]．したがって，タスク間の起動開始の時間的順序は半順序（すなわち，各タスクの起動開始はマスタの起動完了以後に限りどんな順序でもよい[1]）である．これによって，各タスク本体中の実行文系列の実行開始の時間的順序も半順序である．しかし，上記方法を用いてタスクに一意的名前を実行時に付けると，あるマスタに依存する各タスクの本体中の実行文系列の実行開始の時間的順序は，半順序になりえず，全順序になってしまう．なぜなら，そのマスタが各タスクに一意的名前を付けるために各タスクのエントリ SET_ID を順次に呼び出さなければならないからである．結局，Ada並列プログラムの実行モニタは，上記の方法を用いると対象プログラムの本来のタスキング挙動を正確に監視することができない[5),10),11)]．

Germanは，上記の一番目の問題を解決するためにBoochの方法の改良案を提示した[5]．それは，元のタスク型の本体を新しいタスク型の本体中のブロック文に変換するものである（図1参照）．しかし，Adaではタスク本体に関する例外処理とブロック文に関する例外処理とが異なるので，この方法には，変換前のプログラムと変換後のプログラムとが例外処理に対して等価ではないという新しい問題点がある[5]．また，タスク間の起動開始の時間的順序を半順序から全順序に変えてしまうのでBoochの方法と同じ問題点がある．

pattern :

```
task type T is
  [entry_declaration]
end;
:
task body T is
  [declarative_part]
begin
  [sequence_of_statements]
end;
```

→

replacement :

```
task type T is
  entry SET_ID (N : in INTEGER);
  [entry_declaration]
end;
:
task body T is
  ID : INTEGER;
begin
  accept SET_ID (N : in INTEGER) do
    ID := N
  end;
  declare
    [declarative_part]
  begin
    [sequence_of_statements]
  end;
end;
```

図1 Germanの名前付け方法における変換
Fig. 1 Transformation of German's naming strategy.

3. Adaタスクに一意的名前を実行時に付ける新方法

我々が開発したEDENシステムの主な機能は，Ada並列プログラムの実行を監視しそのタスキング挙動に関する情報を収集し保存してから利用者の要求に応じて解析し報告することである[8),9)]．また，収集した情報に基づいて被監視プログラムの実行中にタスク間でランデブーにより通信する際に起こるデッドロック（タスキングにおける通信デッドロックという）を自動的に検出することもできる[8),9)]．EDENは，対象プログラムの実行を監視するためにその各タスクに一意的名前を実行時に付けなければならない．

第2章で指摘した従来提案されている方法の問題点をさらに深く調べると，それらの原因は，各タスクの一意的名前がそのタスクのマスタからSET_IDエントリへの呼び出しによって付けられるためであることが分かる．なぜなら，呼び出しを受けるaccept文は実行文だからタスクの本体の実行文系列中にしかおくことができないのでタスク起動中に一意的名前を与えることができないからである．そこで，我々は，逆方向の名前付け方法，すなわち，名前付けサービスをする特定のタスクを設けそのエントリを各タスクから呼び出すことによって呼び出しタスクに一意的名前を付けるという方法を考え出した．

以下では我々の方法の概略を示す．Ada並列プログラム中のタスクに一意的名前を付けることをサービスするタスク名前サーバをライブラリ・パッ

```
package TASK_NAME_SERVER is
  MAXIMAL_NUMBER_OF_TASKS : constant POSITIVE := 100;
  type INTERNAL_ID_OF_TASK is private;
  function GET_TASK_ID return INTERNAL_ID_OF_TASK;
private
  type INTERNAL_ID_OF_TASK is range 0..MAXIMAL_NUMBER_OF_TASKS;
end TASK_NAME_SERVER;

package body TASK_NAME_SERVER is
  task NAME_SERVER is
    entry GET_A_NEW_TASK_ID (ID : out INTERNAL_ID_OF_TASK);
  end NAME_SERVER;
  function GET_TASK_ID return INTERNAL_ID_OF_TASK is
    ID : INTERNAL_ID_OF_TASK;
  begin
    NAME_SERVER . GET_A_NEW_TASK_ID (ID);
    return ID;
  end GET_TASK_ID;
  task body NAME_SERVER is
    NUMBER_OF_TASKS_OVERFLOW : exception;
    TASK_ID_COUNTER : INTERNAL_ID_OF_TASK := 0;
  begin
    loop
      select
        accept GET_A_NEW_TASK_ID (ID : out INTERNAL_ID_OF_TASK) do
          if TASK_ID_COUNTER = MAXIMAL_NUMBER_OF_TASKS then
            raise NUMBER_OF_TASKS_OVERFLOW;
          end if;
          TASK_ID_COUNTER := TASK_ID_COUNTER + 1;
          ID := TASK_ID_COUNTER;
        end GET_A_NEW_TASK_ID;
      or
        terminate;
      end select;
    end loop;
  end NAME_SERVER;
end TASK_NAME_SERVER;
```

図 2 パッケージ TASE_NAME_SERVER
Fig. 2 Package TASK_NAME_SERVER.

```
with TASK_NAME_SERVER;
use TASK_NAME_SERVER;
procedure MAIN_PROGRAM is
    :
  task body T is
    TASK_ID : INTERNAL_ID_OF_TASK := GET_TASK_ID;
    [declarative part]
  begin
    :
  end T;
    :
end MAIN_PROGRAM;
```

図 3 タスクに一意的名前を実行時に付ける新方法
Fig. 3 A new strategy for naming tasks at run-time.

ケージ（パッケージ名 TASK_NAME_SERVER, 図 2 参照）の一つとしてプログラム・ライブラリに導入する．対象プログラムは，with 節によってタスク名前サーバを参照し，各タスク型の本体の宣言部の先頭にタスク名を格納する変数 TASK_ID を宣言しその変数の初期値設定として一意的名前を与える関数副プログラム GET_TASK_ID を呼び出す（図 3 参照）．これによって，各タスクは実行時にいったん起動すると TASK_ID の確立によって自分の名前を持つ．それからタスクの終了までに，TASK_ID に格納されているタスク名をタスク自身が参照することができる．タスク名の一意性は，一意的名前を与える関数副プログラム GET_TASK_ID がタスク名前サーバ中の名前管理タスクの同一のエントリを呼び出してタスク名をもらうことによって保証される．ここで提案する方法の要点は，Ada の演算対象の初期値設定機能と関数副プログラム中からのエントリ呼び出し機能とを活用していることである．

以上のように付けたタスク名をタスクの起動中にも参照することができるので，起動中のタスクも自分自身を特定できる．さらに，タスクに一意的名前を付けることはタスクの起動中に行われるのでタスクの起動開始に対してどんな影響も与えない．対象プログラム中の各タスクの起動が本来のとおりに並列的になされ，起動開始の時間的順序が本来の半順序になりうる．したがって，我々は従来提

案されている方法の二つの問題点をすべて解決した.

我々が開発した事象駆動型実行モニタ EDEN では,対象プログラムのソース・テキスト中に各タスク型の本体の宣言部の先頭にタスクの一意的名前を格納する変数 TASK_ID の宣言とその初期値設定と(図3参照)を EDEN の前処理部によって挿入する[8),9)].

4. 考　察

Ada には,静的に宣言した一つのサンプルから多数のインスタンスを生成できる演算対象がタスクのほかにもある.例えば,同じ副プログラムを多数のタスクが同時に呼び出す時に多数の副プログラムインスタンスが生成される.多数のタスクが同じ汎用体単位を具体化すると,同時に多数の汎用体インスタンスが存在することが起こる.本論文で提示した名前付け方法は,これらの演算対象に一意的名前を実行時に付ける際にも適用することができる.Ada ではタスクしかエントリをもちえないので,従来提案されている方法はこれらの演算対象に適用することができない.

本論文で提示した方法をタスク以外の演算対象へ適用する例として,タスクのマスタに一意的名前を実行時に付けることを挙げることができる.Ada 並列プログラムの実行モニタは,タスクとそのマスタとの依存関係を把握できるために,各タスクの各マスタに自分自身を実行時に特定できるようにさせなければならない.我々は本論文で提示した方法を EDEN に適用し対象プログラム中のタスクのマスタに一意的名前を実行時に付けることにしている.

さらに,もし本論文で提示した方法を用いる際に,タスクの一意的名前を実引数の値として,そのタスクが呼び出す副プログラムへ伝達すれば,副プログラムの実行中にも該当副プログラムを呼び出すタスクを特定することができる.これによって,対象プログラム中の各タスクの起動から終了までの生涯中の挙動をはっきり区別し監視することができる.これは,Ada 並列プログラムを処理対象とする動的解析ツールにとって,非常に望ましい性質である.例えば,従来の方法を用いているテスト・デバッグ支援ツールは,起動中のタスクの挙動を監視することや,タスクの起動中に起こる通信デッドロック[12)]とタスク依存性に関連する通信デッドロック[12)]とを検出することができなかった[5),10),11)].それに対して,本論文で提示した方法を用いている EDEN システムは,これらのいずれも可能である[8),9),13)].また,Ada 並列プログラム中の実行文の実行回数を計数する動的解析ツールを開発する際に,本論文で提示した方法を用いれば,同一のタスク型の異なるタスクの実行特性を区別することができる.タスク本体中の実行文はもちろん,副プログラムおよび汎用体プログラム単位中の実行文も,どのタスクによって何回実行されたかを調べることができる[14)].

謝辞 九州大学情報工学科荒木啓二郎助教授から本研究について貴重な御意見をいただいた.同大学院情報工学専攻白木原敏雄氏が EDEN システムの前処理部の実現において御協力いただいた.ここに記して両氏に謝意を表する.

参考文献

1) United States Department of Defense: Reference Manual for the Ada Programming Language (ANSI/MIL-STD-1815 A) (Jan. 1983).
2) United States Department of Defense: "STONEMAN", Requirements for Ada Programming Support Environment (Feb. 1980).
3) Booch, G.: Dear Ada, *ACM Ada Letters*, Vol. 2, No. 3, pp. 10-13 (1982).
4) Gehani, N.: *Ada Concurrent Programming*, Prentice-Hall, Englewood Cliffs (1984).
5) German, S. M.: Monitoring for Deadlock and Blocking in Ada Tasking, *IEEE Trans. Softw. Eng.*, Vol. SE-10, No. 6, pp. 764-777 (1984).
6) Helmbold, D. and Luckham, D.: TSL: Task Sequencing Language, Ada in Use, *Proceedings of the Ada International Conference*, pp. 255-274 (1985).
7) LeDoux, C. H. and Parker, D. S., Jr.: Saving Traces for Ada Debugging, Ada in Use, *Proceedings of the Ada International Conference*, pp. 97-108 (1985).
8) Cheng, J., Araki, K. and Ushijima, K.: Event-Driven Execution Monitor for Ada Tasking Programs, *IEEE Proc. of COMPSAC 87*, pp. 381-388 (1987).
9) 程　京徳,荒木啓二郎,牛島和夫:Ada 並列プログラムの事象駆動型実行モニタ EDEN の開発と応用,情報処理学会「コンピュータ・システム」シンポジウム論文集, pp. 1-10 (1987).
10) Helmbold, D. and Luckham, D.: Debugging Ada Tasking Programs, *IEEE Software*, Vol. 2, No. 2, pp. 47-57 (1985).
11) Helmbold, D. and Luckham, D.: Runtime Detection and Description of Deadness Errors in Ada Tasking, *ACM Ada Letters*, Vol. 4, No. 6, pp. 60-72 (1984).
12) Cheng, J., Araki, K. and Ushijima, K.: Task-

ing Communication Deadlocks in Concurrent Ada Programs, *ACM Ada Letters*, Vol. 8, No. 5, pp. 61-70 (1988).
13) Cheng, J. and Usijima, K.: Detecting Tasking Communication Deadlocks in Concurrent Ada Programs, *IEEE Proc. of ICSC '88*, to appear (1988).
14) Kausae, R.: ADAPT: An Ada Programs Validation Tool, Master's Thesis, Department of Computer Science and Communication Engineering, Kyushu University (1988).

(昭和63年4月27日受付)
(昭和63年10月7日採録)

程　京徳（正会員）

1952年生．1982年中国清華大学計算機科学・技術系卒業．同年同大学助手．1986年九州大学大学院工学研究科修士課程情報工学専攻修了．現在同博士後期課程在学中．ソフトウェア仕様記述，プログラミング方法論，ソフトウェア開発支援環境などに興味を持つ．1986年情報処理学会学術奨励賞．日本ソフトウェア科学会会員．

牛島　和夫（正会員）

1937年生．1961年東京大学工学部応用物理学科（数理工学）卒業．1963年同大学院修士課程修了．同年九州大学中央計数施設勤務．1977年九州大学工学部情報工学科教授（計算機ソフトウェア講座担当），現在に至る．1986年4月から九州大学情報処理教育センター長を兼務．工学博士．著書「Fortran プログラミングツール」（産業図書）ほか，日本ソフトウェア科学会，電子情報通信学会，ACM 各会員．