

分散並行オブジェクト指向言語 ACOOL の実装

丸 山 勝 己†

実時間システムの分野では、実時間多重処理と分散処理の能力を持ち、並行して発生しうる多数のイベントを遅延なく処理できるオブジェクト指向言語が望まれる。例えば、パーソナル番号により相手が何処にいても通信できるパーソナル移動通信網や Intelligent Network では、通信網ノード（交換接続を行う交換機や高度のサービス分析や制御を行うサービス制御ノードなど）が協調してサービスを行う。各交換機は同時に数千の呼を遅延無く処理する必要がある。本論文では、分散並行オブジェクト指向言語 ACOOL のコンパイラと実行支援システムの実装技法を述べている。動的・静的効率と並行処理能力は、能動オブジェクトと受動オブジェクトの簡潔で直交的な設計およびコピー無メッセージ通信により実現された。分散処理の効率化は、グローバルオブジェクト ID によるグローバル通信と簡潔なプロトコルスタックにより実現された。目的に適合した簡潔な言語設計は、コンパイラや実行支援システムの効率化と高信頼化に有効である。

Implementation of a Distributed Concurrent Object-Oriented Language "ACOOL"

KATSUMI MARUYAMA†

An efficient object-oriented language with multi-processing and distributed processing capabilities are required in enhanced real-time system programming. For example, in nation-wide personal mobile telecommunication networks, where persons are called using personal-IDs wherever they are, each switching node system is required to handle thousands of calls simultaneously without delay, cooperating with service control nodes distributed in the network. This paper explains the implementation method of a compiler and a run-time execution support for the distributed concurrent object-oriented language "ACOOL". Dynamic and static efficiency is attained by simple and orthogonal implementation of passive objects and active objects, and by copyless message passing scheme. Distributed processing efficiency is attained by global message passing based on global-object-IDs, and by a simple protocol stack. The purpose-tuned and simple language design enables the efficient and reliable implementation of a compiler and a run-time execution support.

1. はじめに

大規模実時間処理システムの拡張性・保守性の向上にはオブジェクト指向が効果的であるが¹⁾、並行多重処理と分散処理の高効率な実現が必要である。例えば、パーソナル番号により相手が何処にいても通信できるパーソナル移動通信網や Intelligent Network では、通信網ノード（交換接続を行う交換機や高度のサービス分析や制御を行うサービス制御ノードなど）が協調してサービスを行う。各ノードは同時に数千の呼を遅延無く処理する必要がある。

これらの要求に対応するために、分散処理並びに実時間多重処理の能力を持ったオブジェクト指向言語 ACOOL (asynchronous-message-base concurrent

object-oriented language)^{2)*}の処理系（コンパイラと実行支援環境）を実装した。本実装法は、以下の特色を持つ。

- (a) 数万の能動および受動オブジェクトを含む実時間処理プログラムに適した動的にも静的にも効率的なオブジェクトを実現。
- (b) データコピー削減などによる非同期メッセージ転送の効率的な実現。
- (c) ノード間通信階層での通信制御による効率的な遠隔メッセージ転送。
- (d) 通信やプロトコルを意識させない分散処理。
- (e) 要求性能に応じて、専用の実行支援システム上でも汎用 OS 上でも走行可能。
- (f) コンパイラと実行支援システムの簡潔な実現。

† NTT ネットワークサービスシステム研究所
NTT Network Service System Laboratories

* 本言語は従来 "COOL" と呼んでいたが、同一名が他で使われていたので "ACOOL" に変更した。

本手法は、C++等の既存言語に並行処理能力や分散処理能力を追加する手法としても効果的である。

2. 言語 ACOOL の概要

ACool は、以下の特徴を持つ簡潔な言語である。

(1) 高効率の並行オブジェクト指向言語

オブジェクトに“スレッド”（本稿では動的静的に軽量の並行プロセスのことをスレッドと呼ぶ）を持たせることで、並列処理機能を持ったオブジェクトを実現できる。このオブジェクトは能動的・並行的に動作できるので**能動オブジェクト**と呼ぶ。モデル的には能動オブジェクトのみでシステムを構築できるが、能動オブジェクトは動的にはコンテキスト切替え、静的にはスタック域が必要であり、並列処理が不要なものに対してはオーバヘッドとなる。またすべてが並列処理能力を必要とするわけではない。従って、ACool はスレッドを持たずに受動的に動作する**受動オブジェクト**も備えている。能動・受動の両オブジェクトの記述は、属性 ACTIVE の有無とメッセージ転送記号以外は同一なので、両者を適切に使い分けることにより、少ないメモリで処理能力の高いプログラムを容易に記述できる。

(2) 受動オブジェクトの内容

受動オブジェクトは、インスタンス変数とメソッドのカプセルである。受動オブジェクトは自己のスレッドを持たないので自らメソッドを実行することができず、メッセージ送信側能動オブジェクトのスレッドによって実行される。メッセージを受信した受動オブジェクトは送信側オブジェクトと直列的に実行されるので、このメッセージを**直列メッセージ**と呼ぶ。“ $\alpha < \mu$ ”は、オブジェクト α に直列メッセージ μ を送ることを意味する。

図1に中心座標と半径で規定される円オブジェクトの定義例を示す。このようにオブジェクトの定義は、オブジェクトの仕様を記述した**クラス定義**とオブジェクトの内部論理を記述した**メソッド定義**からなる。読解性向上のために、メソッド内でインスタンス変数にアクセスするには“MY.”を前置して“MY.x”のように記述する。メソッドは、一つの値を返す“関数形式”（図1 enlarge）と、任意個の値を返す“出力パラメータ形式”（同 move）が可能である。前者はC言語同様に return 式の値が返り、後者はメソッドの実行が終わった時に出力パラメータの値が実パラメータ変数に代入される。

```

TYPE circle =
  CLASS
    init(x,y,r:int);
    move(x,y:int)=>(newx,newy:int);
    enlarge(n:int)=> int;
  VAR
    x,y,r: int; /* instance variables */
  END;

METHOD circle<<init(x,y,r:int);
  MY.x := x; MY.y := y; MY.r := r;
END init;

---- Output-parameter-style method
METHOD circle<<move(x,y:int)=>(x,y:int);
  MY.x += x; MY.y += y;
  x := MY.x; Y := MY.y;
END move;

---- Function-style method
METHOD circle<<enlarge(n:int)=>int;
  MY.r *= n;
  RETURN MY.r;
END enlarge;
----
VAR c : circle; /* Object variable */
VAR pc: REF circle; /* Pointer to object */
VAR i,j,k,r,s,t: int;

SETUP c << init(0,0,10);
SETUP pc << init (7,5,3);

(s,t) := c << move(i+j,k); /*Output-parameter-style*/
r := c << enlarge(5); /*Function-style*/
(s,t) := pc << move(i,j);
r := pc << enlarge(2);

"var += exp;" means "var := var+(exp);"

```

図1 受動オブジェクト

Fig. 1 Passive object.

(3) 能動オブジェクトの内容

能動オブジェクトは、インスタンス変数、メソッドおよび単一のスレッドからなるカプセルである。能動オブジェクトは、並列処理単位であって、メッセージを受信すると自己のスレッドによって目的メソッドを実行する。能動オブジェクトに対するメッセージ転送は**非同期型通信**を採用している。送信側は受信側の処理を待たずに自己の処理を続行できるので、これを**並列メッセージ**と呼んでいる。“ $\alpha < \mu$ ”は能動オブジェクト α に並列メッセージ μ を送ることを意味する。ここに、 α は“ACTIVE REF”タイプのポインタ（これを**グローバルオブジェクト ID**と呼ぶ）であり、能動オブジェクトを識別する。

返答を要求するメッセージを能動オブジェクトに送る場合も多い。この場合、要求メッセージを送ったオブジェクトは“返答待ち状態”に入り、返答メッセージを受理したら処理を再開する。つまり、能動オブジェクトは**メッセージ待ち**、**返答待ち**、**実行待ち**および**実行中**の状態を持つ。

図2に能動オブジェクトの定義例を示す。キーワード“ACTIVE”が能動オブジェクトを意味する。

(4) 分散処理言語

並列メッセージ転送は完全に分散透過であり、大規

```

TYPE Turtle =
  CLASS ACTIVE
    init (name:string; x,y:int);
    go (speed,time:int);
    turn (angle:int);
    where ()=>(x,y:int);
    .....
  VAR
    .... /* Instance variables */
    .....
  END ;

METHOD Turtle <: init(name:string; x,y:int);
  .....
END init;

METHOD Turtle <: go(speed,time:int);
  .....
END go;

METHOD Turtle <: where()=>(x,y:int);
  .....
END where;

VAR Taro :ACTIVE REF Turtle /* Global object-ID */;

SETUP Taro; /* Object creation */
Taro <: init("Kametarotaro ", 7,47);
Taro <: go(10,5);
(x,y) := Taro <: where();

```

図 2 能動オブジェクト
Fig. 2 Active object.

模な分散処理も容易に記述できる。

(5) 強いタイプ付言語

強いタイプ付言語にしてエラーチェック、実行時効率、読解性を向上させている。また、オブジェクトはクラスタイプの実行であるため、オブジェクトも普通のデータのようにアクセスや集合データの要素にすることができ、オブジェクト指向と手続き指向が両立している。

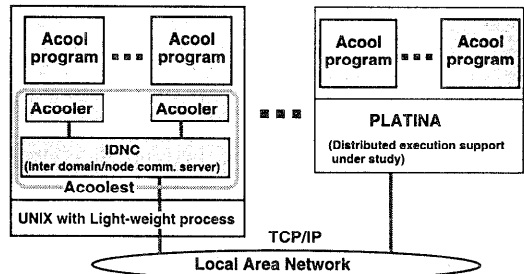
(6) モジュール化機構

外部インタフェースを書く仕様モジュールと内部論理を書く本体モジュールに分離した“プログラムモジュール化機構”を持つ。

3. 本実装法の基本的な考え方

ACOOL は、公衆通信網ノード（交換機やサービス制御ノード）の制御プログラムのような大規模分散処理システムを想定しているので、ネットワーク結合された個々のコンピュータをノードと呼んでいる。各ノードは、任意個の ACOOL プログラムを走らせることができる。個々の ACOOL プログラムは、独立の論理アドレス空間（これをドメインと呼ぶ）が割り付けられて実行される。

ドメインの中では多数の能動オブジェクトが、ドメイン内外のオブジェクトとメッセージ交信しながら並行走行する。



- (1) Acool source files are compiled and linked with the "Acooler" library. This Acool program runs as a Unix heavy process, and is called a "Domain".
- (2) IDNC is a background Unix process to provide inter domain/node communications.
- (3) Acooler and IDNC are collectively called "Acoolest" (Acool execution support).
- (4) PLATINA is a under-study distributed execution support system.

図 3 ACOOL プログラム, Acooler IDNC の位置付け
Fig. 3 Relations of Cool-program, cooler and IDNC.

このような分散処理の実現には、スレッド制御、メッセージ転送、ノード間通信等の機能を提供する“実行支援環境”（これを **Acoolest**=ACOOL execution support と呼んでいる）が必要である。我々は、今後の通信網に要求される高い性能と機能を実現するために、並行オブジェクトモデル型のマイクロカーネル型分散 OS “PLATINA”³⁾ の研究を進めて、その試作と評価も進めている。

また、それほど高性能を必要としない場合や、ACOOL プログラムのデバッグのために、汎用 OS 上で ACOOL プログラムを実行することも必要である。最近の Unix は軽量プロセスをサポートしているので、これを能動オブジェクトのスレッドとして使うことができる。本稿では、SUN-OS の軽量プロセス機能（SUN-LWP と略す）⁴⁾ を使った実装例を説明する（図 3 では“PLATINA”カーネル使用の場合も対比して示した）。

- (a) 個々の ACOOL プログラムは、独立した論理アドレス空間を持つ Unix の (heavy) プロセスとして実行される。これをドメインと呼ぶ。
- (b) 受動オブジェクトにスレッドと非同期メッセージ機能を追加して能動オブジェクトを構築する。
- (c) ACOOL の機能である能動オブジェクトの生成、消滅、制御や非同期メッセージの送信・受信等を、実行モデルの異なる SUN-LWP 機能を用いて実現する必要がある。これらのルーチンの集まりを ACOOL 実行ルーチンのライブラリー **Acooler** (ACOOL Execution Routine library) として実現した。任意個のモジュールから成るソースプログラムは、コンパイルされ、

Ac cooler とリンクされて実行ファイルになる。

- (d) メッセージはオブジェクト ID に従って目的オブジェクトに配送される。ドメイン内通信は Ac cooler 内に閉じて処理されるが、ドメイン間通信とノード間通信は、Unix のプロセス間通信とホスト間通信にマッピングされる。このドメイン間およびノード間通信を支援するサーバプログラムが、**IDNC** (Inter Domain & Node Communication server) である。IDNC はバックグラウンドで走る Unix プロセスで、各ホストごとに一個存在する。
- (e) ACOOL ではグローバルオブジェクト ID の獲得が分散処理の必要十分条件である。別プログラム間でグローバルオブジェクト ID を教え合うには**ネームサーバー**を用いる。これは、“登録検索用キイ情報、グローバルオブジェクト ID”の対を記憶するデータベースである。IDNC はグローバルオブジェクト ID の登録検索のための簡単なネームサーバーの機能も含んでいる。

4. 受動オブジェクトの実装

(1) 受動オブジェクトの内部構造とメソッド結合オブジェクト指向の特長である Polymorphism の実現には、起動されるメソッドの決定をコンパイル時ではなくメッセージ転送実行時に決定する必要がある。しかし Smalltalk⁹⁾ のような実行時メソッドサーチはオーバーヘッドが大きい。そこで図 4 に示すように、そのクラスの各メソッドへのリンクを持つメソッドテーブルをクラス定義対応に設け、また個々のオブジェクト変数域の先頭ワードにこのメソッドテーブルへのリンクを持たせている。オブジェクトにメッセージが到着すると、このリンクをたどって目的メソッドを決定する。目的メソッドは、メッセージ名で固定的に決まるのではなく、メッセージを受信したオブジェクトのメソッドテーブルで実行時に決定されるので、Polymorphism が実現される。

(2) オブジェクトの生成

受動クラスタイプの変数が受動オブジェクトである。Setup 文により図 4 の構造を持つオブジェクトが組み立てられる。

SETUP 変数 [<<メッセージ] ;

Setup 文は、“変数” がクラスタイプ変数の場合はメソッドテーブルリンクの設定のみを行う。“変数” がクラスタイプへのポインタ (REF) 変数の場合は、オ

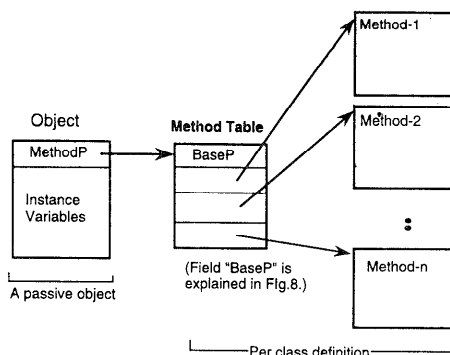


図 4 受動オブジェクトの内部構造
Fig. 4 Internal structure of passive objects.

ブジェクトのメモリ域を動的に割り付けた上で、メソッドテーブルリンクを設定し、オブジェクトのアドレスをポインタ変数に代入する。メッセージが記述してあれば、これが直ちにオブジェクトで実行される。初期設定メッセージとして便利に使える。

(3) 直列メッセージ転送

受動オブジェクトへのメッセージ転送は、メソッドテーブルを介した間接呼び出しに展開する。ACOOL は入力パラメータと出力パラメータを持てる (共に任意個)。入力パラメータは、C 言語の関数呼び出し同様に実パラメータ値をスタック経由で形式パラメータに引き継いでいる。出力パラメータは大変に便利であるが、その処理は少し複雑である。返答値の代入先であ

```
METHOD circle << move(dx,dy: int)
    => (x,y: int);
MY.x += dx;    MY.y += dy;
x := MY.x;     y := MY.y;
END;

VAR obj: circle;
(x,s) := obj << move(i+j,k);
```

(a) ACOOL program

```
circle_move(selfp,xAdrs,yAdrs,dx,dy)
circle *selfp; /* オブジェクトアドレス */
int *xAdrs,*yAdrs; /* 出力先アドレス */
int dx,dy; /* 入力パラメータ */
{ int x,y; /* 出力パラメータ */
  selfp->x += dx;    selfp->y += dy;
  x = selfp->x;      y = selfp->y;
  *xAdrs = x;        *yAdrs = y;
}

circle obj;
(*obj.MethodP->move)(&obj,&r,&s,i+j,k);
```

(b) Equivalent C-program image

図 5 出力パラメータ付メソッドとメッセージ転送の展開イメージ

Fig. 5 Compilation image of a method and message-passing with output parameters.

る実パラメータ変数のアドレスをスタック経由でメソッドに引き継いでおき、メソッド実行が終了する際に出力パラメータの値をそのアドレスに代入する。図1の move メソッドとメッセージ転送の展開形を（C言語イメージになおして）説明したのが図5である。展開形においてパラメータ selfp はインスタンス変数域のアドレス、xAdrs, yAdrs は出力先変数のアドレス、dx, dy は入力パラメータである。インスタンス変数アクセス “MY.x” は “selfp->x” に展開され、メッセージ転送 “obj<<move()” は、“obj からメソッドテーブル経由で選択された move メソッド” の呼び出しに展開されている。

5. 能動オブジェクトの実装

(1) 能動オブジェクトの内部構造

能動オブジェクトは、図6に示すように受動オブジェクトにスレッド（網掛け部）を加えた構造をしている。スレッドは、メッセージ行列を含むスレッド制御ブロックとスタック域からできている。並列メッセージは非同期転送なので、メッセージバッファにのせて宛先オブジェクトのメッセージ行列に挿入される。

前述のように、能動オブジェクトの実行と遠隔メッセージ転送は実行支援環境（OS やライブラリルーチン）の支援が必要である。

(2) 能動オブジェクトの生成

能動クラスタイプのポインタ変数（ACTIVE REF …）を指定した setup 文を実行すると、メモリ域確保とスレッドの生成を行って図6の構造を持つ能動オブジェクトを組み立てた上で、グローバルオブジェクト ID 値をポインタ変数に返す。

(3) グローバルオブジェクト ID の構造

能動オブジェクトを識別するグローバルオブジェクト ID は、図6(B)に示すようにノード ID、ドメイン ID、ドメイン内ローカル ID およびアクセスキーを含んでおり、オブジェクトをグローバルにユニークに識別できる（現実装では各フィールドは16ビット長）。

並列メッセージは、メッセージバッファに書かれている宛先グローバルオブジェクト ID に従って目的ノードの目的オブジェクトに配送される。メッセージ転送の際に、メッセージバッファに書かれた宛先のアクセスキーと受信オブジェクトが持つアクセスキーとを比較し、不一致ならばそのメッセージを破棄して違法メッ

セージを防護している。

(4) メッセージ転送とメソッドの決定法

並列メッセージを運ぶメッセージバッファには、メッセージ内容の他に宛先と送り元のオブジェクト ID、メッセージ番号等が書かれている。メッセージ名を文字列として送るのは非効率なので、各クラス定義ごとにメッセージ名の並び順に 1, 2, 3, … として与えたメッセージ番号を用いている。

返答の有無による2形式の並列メッセージ転送：

(a) オブジェクト指定 <:

メッセージ名（式の並び）；

(b) （変数の並び）:=オブジェクト指定 <:

メッセージ名（式の並び）；

は、次のように展開される。①メッセージバッファを割り付ける；②メッセージバッファに内容を記入する；③メッセージ送信の ACOOL 実行ルーチン呼び出し、メッセージを送出する；④形式(a)ならば自己の処理を続行する；形式(b)ならば返答待ちの ACOOL 実行ルーチン呼び出して“返答待ち状態”に入り、返答メッセージが戻って来たら処理を再開してその値を左辺に代入し、返答を運んできたメッセージバッファを解放する（つまり要求メッセージ送信と返答メッセージ受信のペアに展開される）。

(5) メッセージ受信とメソッド実行

能動オブジェクトは、図7に示すメッセージ駆動ルーチン “msg_driven_loop()” を実行している。本ルーチンは、ACOOL 実行ルーチン “K_msg_receive()” にて自オブジェクトにメッセージが到着

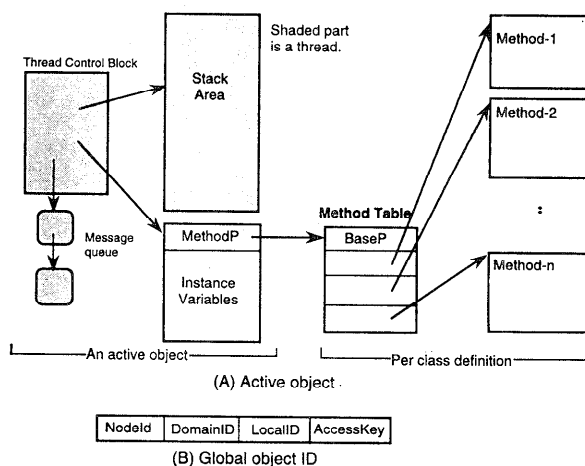


図6 能動オブジェクトとグローバルオブジェクト ID 内部構造
Fig. 6 Internal structure of active objects and global object IDs.

```

void msg_driven_loop()
{
    msg_buf_t *msgp;
    for(;;){ /*メッセージ駆動ループ*/
        /*メッセージ受信待ち*/
        msgp = k_msg_receive();
        obj_adrs = オブジェクトのアドレス;
        /*メソッド実行*/
        exec_method(obj_adrs, msgp);
        /*返答メッセージの処理*/
        if (返答有) k_msg_reply(msgp);
    }
}

```

(A) Message-driven loop program

```

void exec_method(obj_adrs, msgp){
    obj_t *obj_adrs; /*オブジェクトのアドレス*/
    msg_buf_t *msgp; /*メッセージバッファのアドレス*/
    { void (*method)(); /*メソッドのポインタ*/
        method_tbl_t *method_tbl_p;
        /*メッセージテーブルのアドレスを求める*/
        method_tbl_p = obj_adrs->method_p;
        /*メッセージ番号が指定するメソッドを選択*/
        method = (*method_tbl_p)[msgp->hdr.msg_num];
        /*選択されたメソッドを実行*/
        (*method)(obj_adrs, msgp->body);
    }
}

```

(B) Method execution program

図 7 メッセージ駆動ルーチンとメソッド実行ルーチンの概要
Fig. 7 Outlines of message-driven routine and method-execution routine.

するのを待ち、メッセージが到着すると ACOOL 実行ルーチン “exec_method()” を起動して目的メソッドを実行することを繰り返している。

ルーチン “K_msg_receive()” は、自オブジェクトにメッセージが到着していれば直ちに、さもなければ到着するのを待って、メッセージ行列からメッセージバッファを取り出して、そのアドレスを返す。

ルーチン “exec_method()” は、入力パラメータとしてオブジェクトアドレスとメッセージバッファアドレスを持ち、オブジェクトアドレスからメソッドテーブルにアクセスし、次いでメッセージ番号 (=メッセージバッファに書かれている) でメソッドテーブルをインデックスして目的のメソッドを選択する。そして、選択されたメソッドにオブジェクト (インスタンス変数域) アドレスとメッセージバッファのアドレスをパラメータ引き継ぎして実行する。

並列メッセージは一般にサイズが大きいので、メッセージバッファで送られてきた入力パラメータを通常のサブルーチン呼び出しのようにスタックに積んでメソッドに引き継ぐのでは、スタックに積み直す処理が実行オーバーヘッドとなる。そこで、効率化のために、メッセージバッファのメモリ域をそのままメソッドの形式パラメータ域として活用することにした。つまり、入力パラメータ値が記憶されているメッセージバッファのメモリ域をそのままメソッド側の形式パラメータ

(入力パラメータを意味するローカル変数) として利用する。また、出力パラメータに該当するローカル変数も、メッセージバッファ上に割り付けている。本手法により、並列メッセージが非常に効率化された。

出力パラメータ付メソッドの場合は、ACool 実行ルーチン K_msg_reply() を実行すると、出力パラメータの載ったメッセージバッファが要求元に返送される。

(6) メソッド内部での選択的メッセージ受信

ABCL/1⁶⁾同様にメソッドの内部での選択的メッセージ受信も可能であり、実用プログラムでは利用価値が高い。例えば、次の receive-case 文:

```

RECEIVE CASE (msg1, msg3, msg6);
? msg1(x, y: int):      処理 a;
? msg3(x, z: shortint): 処理 b;
? ELSE :                処理 c;
ESAC;

```

は、以下の内容に展開される。①msg1, msg3, msg6 の何れかのメッセージが到着するのを待つ (メッセージ番号対応ビットに '1' を建てたビット列を ACOOL 実行ルーチン K_msg_selective_receive() に渡しして); ②到着したメッセージが msg1/msg3 ならば、処理 a/処理 b を実行する。メッセージバッファ上のパラメータ域がそのままパラメータ変数 x, y/x, z として使われる (ここにパラメータ変数 x, y/x, z の有効範囲は各分岐内である); ③ msg6 が到着した場合は処理 c を実行する; ④それぞれの処理が終わるとメッセージバッファを解放する; ⑤これら以外のメッセージはメッセージ行列に保存されたままである。

(7) 委譲 (Delegation)

実行文 『DELEGATE TO α AS μ ;』は、現メッセージの処理をオブジェクト α のメソッド μ に委譲することを意味する (μ は同一メッセージインタフェースを持たなければならない)。これは、現在処理中のメッセージバッファを、メッセージ番号を μ の番号に変換して α に転送してメソッド μ を実行させ、現メソッドの実行を終了させることで実現している。非常に簡単かつ効率的に委譲が実現できる。

6. 継承と実行時拡張

(1) 継 承

図 8 に継承の実現法を示す。クラス定義の “BASE (β)” は、クラス β (こちらをベースタイプと呼ぶ) を継承した拡張タイプであることを意味する。

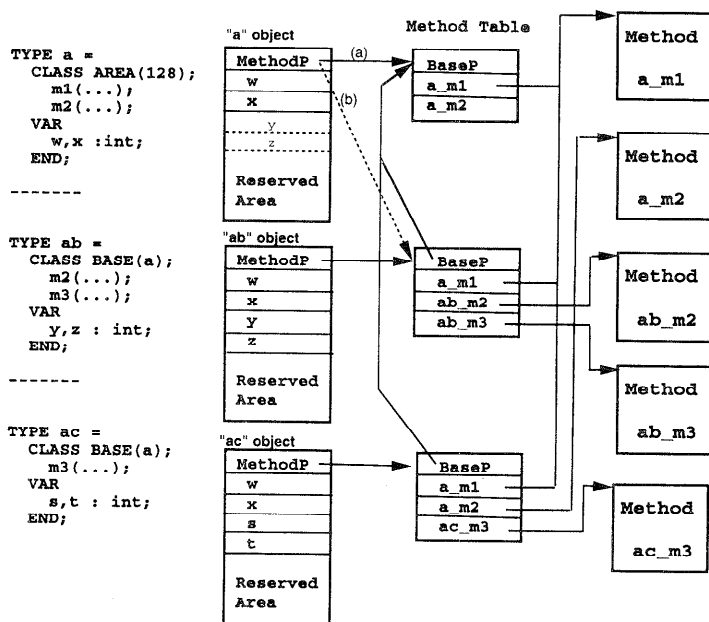


図 8 継承と動的拡張の仕組み

Fig. 8 Mechanism of inheritance and dynamic extension.

本例にてクラス“ab”および“ac”は、クラス“a”の拡張タイプである。クラス“ab”のメソッドテーブルは、メッセージ“m1”に対してはクラス“a”のメソッド“m1”に、メッセージ“m2”と“m3”に対しては新たにクラス“ab”で定義された各メソッドにリンクしており、これによりメソッドの継承が簡単に実現されている。受動・能動オブジェクトともに継承が可能である。

(2) 実行時のタイプチェック

各オブジェクトはメソッドテーブルへのリンクを持っている。図8に示すようにメソッドテーブルの第1ワード (BaseP) はベースタイプのメソッドテーブルにリンクしている。これは継承関係を表現しており、以下のように活用している。

- ACOOL の組み込み関数 **IS_BASE(α, β)** は、 α が β のベースタイプであれば TRUE を返す関数で、実行時に継承リンクをたどって真偽を判定する。
- 次項でのべる動的拡張では、正しい継承関係にあるかを実行時にチェックする。
- 実行効率よりもプログラムの融通性を重視する分野では、この継承リンクを用いてメッセージ転送の実行時タイプチェックを行うことも可能である。

(3) 動的拡張

ACOOL は、オブジェクトを元のタイプからその拡張

タイプに実行中に拡張することができる。例えば、図8で定義されたクラス“a”とその拡張クラス“ab”、“ac”において、“a”として生成されたオブジェクトを実行中に条件に応じて“ab”タイプあるいは“ac”タイプに拡張することができる。

図8のクラス定義“a”に書かれた“AREA(128)”は、動的拡張ができるようにオブジェクト生成時にインスタンス変数域として予備域込みで128バイトを割り付けておくことを意味する。本属性は動的拡張のための属性で、普通の継承では記述不要である。

ACOOL 組み込み関数 **EXTEND** (目的オブジェクト、拡張クラスタイプ) を実行すると、

以下の仕組みにより動的拡張が行われる。①メソッドテーブル間の継承リンクを使って、ベース/拡張クラスの関係にあるかをチェックし、不当ならばエラーコードを返し、正当ならば以下に進む。②元のオブジェクトの予備域 (AREA 属性で確保された領域) を拡張タイプで追加されたインスタンス変数域として使う。③オブジェクトのメソッドテーブルリンクを拡張タイプのメソッドテーブルに切り換える。例えば“a”タイプオブジェクトのメソッドリンクを実線(a)から破線(b)に切り替えると、本オブジェクトは“ab”タイプの機能を持つようになるわけである。

7. 実行支援環境

実行支援環境 Acoolest は、スレッド制御、メッセージ転送、ノード間通信等の機能を提供する。本章では、Unix 上で ACOOL 分散処理プログラムを実行させるための実行支援環境を説明する。これは、第3章の図3で説明したように、実行ルーチンライブラリ“Acooler”と、ドメイン間・ノード間通信を行うサーバー“IDNC”からなる。IDNC はバックグラウンドで走る Unix プロセスである。ACOOL ソースプログラムは、コンパイルされ、Acooler とリンクされて実行ファイルになり、Unix の (heavy) プロセスとして実行される。

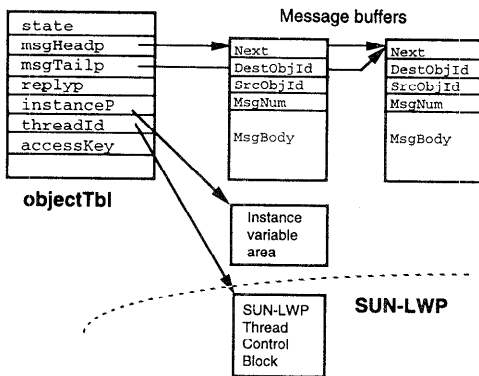


図 9 オブジェクトテーブル
Fig. 9 Object table.

7.1 実行ルーチンライブラリ

Acooler は以下のルーチンを含んでいる。

(1) 能動オブジェクト生成ルーチン

SUN-LWP は、ACOOOL の並列メッセージ実現に必要な非同期通信機能を持っていない。そこで、オブジェクト状態、メッセージ行列、インスタンスアドレス、スレッド ID 等を記憶するオブジェクトテーブル(図 9)を Acooler 内に設けて非同期通信や状態管理を行わせている。図 6 の Thread Control Block の部分機能に相当する。

能動オブジェクト生成ルーチンは、①空きオブジェクトテーブルを確保し、②スタック域とインスタンス域を確保し、③SUN-LWP によりスレッドを生成して、④図 6 のリンクを構築する。

(2) メッセージ送信ルーチン

メッセージ送信ルーチンは、メッセージバッファに書かれた宛先が自ドメイン内か否かを判定し、自ドメイン内の場合は、宛先オブジェクトのオブジェクトテーブルにアクセスし、アクセスキイ等の正当性チェックを行い、正当ならばメッセージをメッセージ行列に FIFO として挿入する。また、宛先オブジェクトがメッセージ受信待ち状態であったら、それを実行可能状態にする。他ドメイン宛ならば、IDNC にメッセージを転送する。返答送信ルーチンの場合は、相手が“返答待ち状態”であることを確認してメッセージを渡し、相手を実行可能状態にする処理が加わる。

(3) メッセージ受信ルーチン

自己のオブジェクトテーブルのメッセージ行列にメッセージが到着していたら、そのメッセージを外して返す。さもないければ“メッセージ待ち状態”(返答受信の場合は“返答待ち状態”)に入る。

(4) 外部からのメッセージの配送機構

他ノードや他ドメインから本ドメイン内オブジェクト宛のメッセージが送られてきたならば、例外ハンドラが起動されてメッセージを宛て先オブジェクトに転送して(2)と同様の処理を行う。

(5) スタートアップルーチン

ACOOOL プログラム開始時に走り、各テーブルの初期設定を行い、Unix の socket 機能 (AF_UNIX, SOCK_STREAM 型) を使って IDNC サーバーに接続要求を送ってドメイン-IDNC 間の通信コネクションを設定する。

7.2 ノード間通信サーバー “IDNC”

個々のメッセージはパケットとして転送される。

IDNC は Unix のバックグラウンドプロセスであり、ドメイン間およびノード間通信のパケット交換を行う。遠隔通信はパケット損失があるので、再送制御等が必要である。再送制御を個々のオブジェクト間で行うのはオーバーヘッドが大きい。そこで、ドメイン間にコネクション通信を張って再送制御を行い、ドメインの内部ではオブジェクト ID によるコネクションレス通信とした。つまり、コネクション型のドメイン間通信の上に、コネクションレス型のオブジェクト間通信を積んだプロトコルスタック構造をもつ。IDNC で多重化されるので、各ホスト (=ノード) 間コネクションはドメイン数によらず一本である。なお、PLATINA を利用した場合は、ノード間がコネクション通信、ノード内はオブジェクト ID によるコネクションレス通信となるので、この点からも更に効率化される。

具体的には、IDNC 間 (Unix ホスト間) 通信には TCP プロトコルを、IDNC-ドメイン間通信には、コネクション型通信 (Unix の SF_UNIX, SOCK_STREAM 型 socket 機能) を採用した。

①IDNC は、立ち上げ時に接続ホスト名の書かれた構成ファイルを見て通信相手のノード (ホスト) との間に TCP コネクションをはる。

②パケットが IDNC に到着すると、IDNC はパケットの宛先を調べ、同一ノード内であったら目的ドメインにパケットを転送する。他ノードであったら目的ノードの IDNC にパケットを転送する。

③IDNC は、単純なネームサーバー機能も備えている。ネームサーバーは、オブジェクト登録メッセージを受けると、自己のデータベースに登録するとともに他の IDNC にも通知して登録させる。検索メ

ッセージを受けると自己のデータベースを探して該当オブジェクト ID を返す。

8. コンパイラ

ACOOL は Modula-2⁷⁾ 並にコンパイラ作成の容易化も考慮した設計なので、非最適化コンパイラならば 10 K 行以下で実現できる。しかし、RISC プロセッサには最適化が必須であり、また各種のターゲットマシンに対応できるコンパイラであることも望ましい。そこで Free Software Inc. の GNU-C コンパイラ GCC⁸⁾ を流用して ACOOL コンパイラを開発した。GCC は高度な最適化を含み、マシン記述マクロの定義により各種マシンに対応できる優れたコンパイラである。

GCC コンパイラの構造は、大きく分けてパーシメントリートをやる言語依存部、レジスタ転送言語生成部、機械語生成部に分かれている。ACOOL 化は GCC コンパイラの言語依存部のみの修正で済んでいる。シンボルテーブルは処理の簡易化のために数フィールドの追加を行った。ACOOL 化の書換え規模は、語彙分析+構文解析が lex+yacc+C で約 1000 行、ネームテーブル作成関連の変更が C で約 5000 行である。本コンパイラは GCC のシンボリックデバッキングインタフェース、最適化モード、各種マシンへの移植性等を受け継いでいる。また、Acooler と IDNC は C でそれぞれ約 1100 行、700 行である。

9. 分散処理の実験例

(1) ファイルサーバープログラム

分散処理の応用例として、XINU⁹⁾の無状態ファイルサーバープログラム (C 記述で約 500 行) は ACOOL では約 350 行で書ける。サーバープログラムを能動オブジェクト、個々のファイルを受動オブジェクトとして記述することにより、効率とプログラム解読性を両立できた。内容の分かりやすさは歴然としている。

(2) 遠隔メッセージ転送能力

二台の SUN 4 を用いて図 10

に示す試験プログラムと環境により、二つの能動オブジェクト間で要求メッセージを送って返答メッセージ (共に 64 バイト) を受けとるまでの経過時間とクライアント側の CPU 時間を実験した。①同ドメイン内、②同一ノード内の異ドメイン間、③異ノード間について調べた結果を表に示す。異ノード間通信でもメッセージの一往復の時間は 4.9 ms (CPU 時間 1.2 ms) である。この値は、ドメイン-IDNC 間通信、ノード間通信、メッセージバッファの割付・解放時間も含まれており、Unix 上でも速い。

本構成は、ノードやドメインの個数が多いことを想定したので、IDNC プロセスを設けてドメイン間通信を中継させている。もしドメインの個数が少なければ、Acooler に IDNC 機能を組み込んで、各ドメインが直接通信をする構成にすると、IDNC-Acooler 間のプロセス間通信が削減されるので、例えば図 10 のノード間往復時間 4.9 ms は約 3 ms に高速化できる。

10. 関連研究

オブジェクト指向モデルのメッセージ転送とカプセル化は、分散処理には最適なアプローチである。オブジェクト指向言語の Smalltalk⁶⁾, C++¹⁰⁾, Eiffel¹¹⁾ は分散並列処理機能を提供していない。ABCL/I⁶⁾,

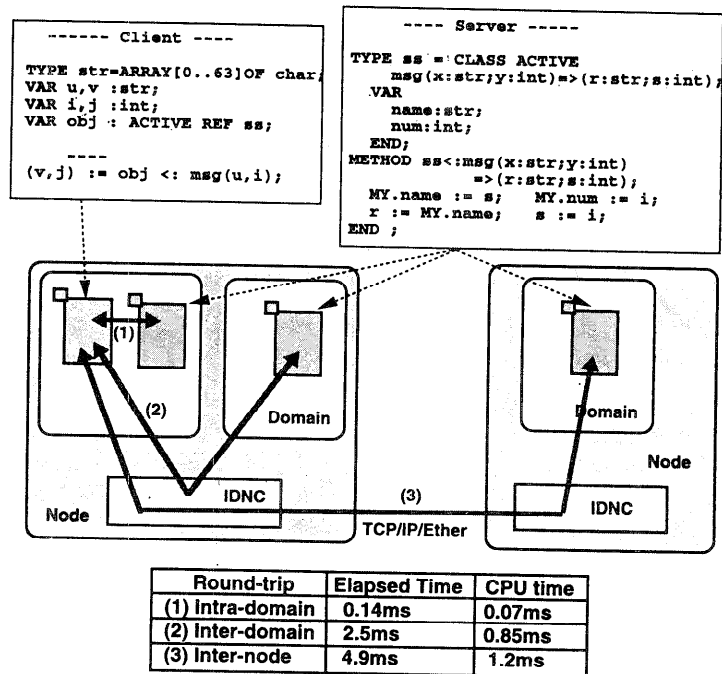


図 10 分散処理試験例

Fig. 10 Distributed processing example.

Concurrent-Smalltalk⁶⁾は先駆的な並列処理機能を有するが、本稿のような超多重実時間処理分野を狙った実装ではない。ABCL/M¹²⁾は、非同期メッセージをサポートする分散オブジェクト指向言語であり、オブジェクトを解釈実行型仮想スタックマシンとした実装をとっている。これを高速化するには、本稿の手法が適用できる。Emerald¹³⁾は、オブジェクトの内部にプロセス記述を含ませて管動オブジェクトを実現する。メッセージは同期型であり、遠隔通信はオブジェクト管理テーブル経由のリモートプロシージャコールとなり、融通性は高いが実行速度は下がる。Distributed Smalltalk¹⁴⁾は、遠隔オブジェクトの代理を行う代理オブジェクトとリモートからのアクセスを管理するリモートオブジェクトテーブルを使った実装である。Smalltalk 両立が利点であるが、効率に限界がある。Presto¹⁵⁾は、共用メモリ型マルチプロセッサ上のパラレル処理向を狙ったC++拡張言語である。仮想計算機に当たるスレッドオブジェクトがあり、単一オブジェクトの中でマルチスレッドを走らせることができるが、本稿の分野とは異なる。

本稿の手法は、簡単な機構で効率の良い分散プログラムが実現できる。

11. おわりに

能動オブジェクトと受動オブジェクトのシンプルで統一的な実装、グローバルメッセージ転送による分散を意識させない分散処理記述、強タイプ付言語を活用することにより、簡単な機構で高い機能を効率をもつ分散並行オブジェクト指向言語システムを達成できた。直列メッセージ通信は間接プロシージャコールに展開されるので、受動オブジェクトのオーバーヘッドは無視できる。能動オブジェクトの主なオーバーヘッドは並列メッセージ転送とスタック域であるが、処理モデルが簡単なので効率よく実現できた。分散処理能力はUnix上でも高い効率を示したが、別途研究中の通信網用の分散処理OS“Platina”を用いれば、スレッド実行やメッセージ通信が最適化させるので、更に高い効率が得られる。

また、ACoolの仕様モジュールは、分散プログラム間のインタフェース記述の役割も果たすので、インタフェース記述言語もスタブプログラムジェネレータも不要であり、使いやすさ、性能ともに優る。C関数のインタフェース定義を書いた仕様モジュールを用意しておくだけで、ACoolプログラムからC関数を自

在に呼び出せるので、既存Cプログラムの活用も自由である。

本稿では説明を省略したが、本システムは一台のホストで複数ノードを擬似する機能も有する。これは、分散処理プログラムのデバッグ容易に大役役立つ。

謝辞 本研究に対し貴重な意見や使用経験のフィードバックを頂いたNTTネットワークサービスシステム研究所の諸氏およびSRA社の林氏に深謝します。また、論文の推敲のための有益なコメントを頂いた査読者に深謝します。

参考文献

- 1) Maruyama, K. et al.: A Concurrent Object-Oriented Switching Program, *IEEE Comm. Magazine*, Vol. 29, No. 1, pp. 60-68 (1991).
- 2) 丸山勝己: 並行オブジェクト指向言語 COOL, 情報処理学会論文誌, Vol. 34, No. 5, pp. 963-972 (1993).
- 3) Maruyama, K. et al.: Platform for Telecommunication and Information Network Applications: PLATINA9, *Proc. of TINA Conference*, pp. 31-43 (1992).
- 4) SUN Microsystems: SUN-OS-4.1 マニュアル (1991).
- 5) Goldberg, A. et al.: *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley (1983).
- 6) Yonezawa, A. and Tokoro, M. (ed.): *Object-Oriented Concurrent Programming*, The MIT Press (1987).
- 7) Wirth, N.: *Programming in Modula-2*, Spriger-Verlag, New York (1982).
- 8) Free Software Foundation Inc.: GNU-C compiler のマニュアルとソースコード (1988).
- 9) Comer, D.: *Operating System Design—Vol. 2*, Prentice-Hall International (1987).
- 10) Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley (1986).
- 11) Meyer, B.: Eiffel: The Programming Language for Reusability and Extendability, *SIGPLAN Notice*, Vol. 22, No. 2, pp. 85-96 (1987).
- 12) Yonezawa, A.: *ABCL: An Object-Oriented Concurrent System*, The MIT Press (1990).
- 13) Black, A. et al.: Distribution and Abstract Types in Emerald, *IEEE Trans. SE*, Vol. SE-13, No. 1, pp. 65-76 (1987).
- 14) Bennet, K. J.: Experience with Distributed Smalltalk, *Software-Practice and Experience*, Vol. 20, No. 2, pp. 157-180 (1990).
- 15) Bershad, B. et al.: PRESTO: A System for Object-Oriented Parallel Programming, *Software-Practice and Experience*, Vol. 18, No. 8, pp. 713-732 (1988).

(平成6年4月7日受付)

(平成6年10月13日採録)



丸山 勝己 (正会員)

1944 年生. 1968 年東京大学工学部電子工学科卒業. 1970 年同大学院修士課程修了. 同年日本電信電話公社入社. 現在 NTT ネットワークサービスシステム研究所主席研究員. 電子交換機の実時間増設方式, 高水準言語, 最適化コンパイラ, 並行オブジェクト指向プログラミング, 交換プログラム構造, 通信網用分散処理などの研究実用化に従事. 1985~88 年 CCITT 第 X 研究委員会副議長. 著書『交換用プログラミング言語 CHILL』(電気通信協会). 工学博士 (東京大学). 1990 年度本学会論文賞受賞. 電子情報通信学会会員.
