既存並列処理言語による実時間オブジェクト指向プログラミング†

丸山勝己†† 渡部信幸††

オブジェクト指向プログラムは，保守性・機能追加性・ソフトウェア部品化に優れており，電話交換機の制御プログラム（以降交換プログラムと呼ぶ）のような大規模（数百k行）・長寿命で頻繁に機能追加が行われるシステムにふさわしい特性を持っている．交換プログラムは，実時間処理・数千のオーダの超多重処理のため Smalltalk 的なモデルでは実行効率が追従できない．しかし実時間オブジェクト指向言語を新規に開発し現用言語に代わって導入することは各種の面から容易でない．そこで国内外で交換プログラム等で使われている Chill によるオブジェクト指向プログラミング法を述べる．本提案手法は以下の特色を持ち，Chill 以外の言語にも適用可能である．●並列オブジェクトによる多重処理実現．●直列オブジェクトによる実時間効率の実現．●既存言語によるオブジェクト指向プログラミング（インヘリタンス機能を含む）．●読解性をオブジェクト指向言語並みに向上させるための簡易プリプロセッサ．

1. まえがき

オブジェクト指向プログラム[1)-3)]は，保守性・機能追加性・ソフトウェア部品化に優れており，電話交換機の制御プログラム（以降交換プログラムと呼ぶ）のような大規模（数百k行）・長寿命で頻繁に機能追加が行われるシステムの記述にふさわしい特性を持っている．この理由によりわれわれはオブジェクト指向に基づく交換プログラム構造の研究を進めており，その効果を確認している．交換プログラムは，実時間処理，同時に数千の電話呼の呼接続を行う超多重処理，各種ハードウェアの制御等が要求されるが，代表的なオブジェクト指向言語である Smalltalk[1)] のモデルでは以下の点で実時間超多重処理には弱い．

- 融通性は高いが実行時オーバヘッドが大きい（全要素がオブジェクト，動的メソッドサーチ等のため）．
- 多重処理に必要な並列実行の機能が弱い．
- Garbage Collection は実時間処理には重すぎる．

したがって，われわれは実時間超多重処理に適応させるために並列オブジェクトと直列オブジェクトからなるモデルを考案した．この実現には新規のオブジェクト指向言語が望まれるが，新規言語を早急に開発し現用言語に代わって導入することは各種の面から容易でない．そこで既に使われている言語を用いて並列オブジェクト指向プログラミングを行うことも有用である．本論文では，実時間多重処理向けのオブジェクト指向モデルと，NTT 始め国内外で交換プログラムなどの作成に使われている Chill[4),5)] (CCITT で設計されたシステム記述言語）による並列オブジェクト指向プログラミング法を述べる．また，簡単なプリプロセッサを用いれば，さらに本格的なオブジェクト指向言語並みの読解性も得られることを示す．本手法は Chill 以外の言語でも適用可能である．

2. 基本的な考え方

電話交換機は，数万個の各種リソース（スイッチ通話路，加入者回線，中継回線，サービス回路等）を制御して同時に数千の電話呼の接続制御を行うシステムである．これらの各種リソースや個々の呼接続制御に対応するソフトウェア構成要素は，各要素自身の状態を有する；指令を受けて所定の動作をする；同一機能を持つものが多数個存在する；などの性質を持つ．つまりオブジェクト指向モデルの特徴によく適合している．

したがって，これらの構成要素を下記のようにオブジェクトとして構成することにより，交換プログラムをメッセージを交信しあうオブジェクトの集まりとして見通し良く構築することができる．

① 各オブジェクトは，その内部状態等を記憶しておりメッセージを受けると自律的に所望の動作をする．オブジェクト定義はそれ自体に閉じた処理を書くだけなので簡単化される．

② オブジェクトの外部インタフェースは，内部の作りに依存しない論理インタフェースとする．オブジェクトを制御するには，目的オブジェクトに論理メッセージを送るのみでよい．

† Real-Time Object-Oriented-Programming in an Existing Concurrent-Process Language by KATSUMI MARUYAMA and NOBUYUKI WATANABE (NTT Communication Switching Laboratories).

†† NTT 交換システム研究所

③ リソース割付はオブジェクト割付に対応する．つまりリソース割付オブジェクトに割付メッセージを送ると，該当リソースが割り付けられそのオブジェクト ID が返される．

④ 構成要素 α に機能追加をしたものが構成要素 β になる関係も多い．この場合にはオブジェクト指向の継承機能により差分・増分のみを書くだけで良くなる．

このようにオブジェクト指向モデルは，構造的には交換プログラムに適している．そこで，実時間処理効率と数千のオーダの多重処理能力を達成するためには，以下の機構を導入する．

（1） **並列オブジェクト**：交換機は同時に数千の呼を処理する多重処理システムなので，各オブジェクトは並列動作能力，つまり平行プロセスの機能が必要である．そこでこれを**並列オブジェクト**[6)~8)]と呼ぶことにする．モデル的には並列オブジェクトのみでシステムを構築するのが最も簡単である．

（2） **直列オブジェクト**：並列オブジェクトは，各オブジェクト対応にプロセス制御域とワークスタックを持ち，実行にはプロセス切換えを要する．つまり静的にも動的にも重く，実時間超多重処理を特徴とする交換プログラムの数万個に上るリソースを並列オブジェクトのみで構築するのは無理である．幸いなことに大部分のオブジェクトはメッセージの送り手に従属して実行させることが可能である．この場合はメソッドをメッセージの送り手のワークスタックを使ってプロシージャ呼出し的に実行させることで静的・動的に効率化できる．これを**直列オブジェクト**と呼ぶことにする．

（3） **メッセージ転送**：オブジェクト間インタフェースはメッセージ転送で統一し，オブジェクト β にメッセージ（名前 μ，パラメータ π）を送ることを以下のように記述することとする．

① 並列オブジェクトに対するメッセージ転送（送り手はメッセージを送った後，受け手の動作完了（と返答値）を待たずに処理を続行する）：

$\beta<<<\mu(\pi)$;

② 直列オブジェクトへのメッセージ転送（送り手はメッセージ受け手の動作完了（と返答値）を待つ）：

$x:=\beta<<\mu(\pi)$;

機械語レベルでは，宛先オブジェクトが並列型か直列型により以下のように効率的なコードに展開される．

① 並列オブジェクトへのメッセージ転送：プロセス間の非同期通信に展開する．

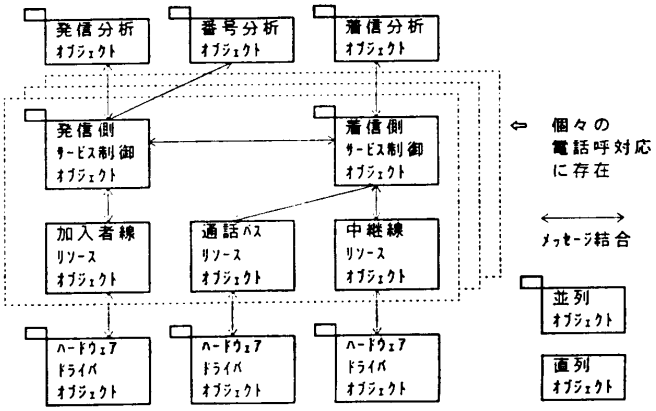

図 1 オブジェクト指向モデルによる交換プログラム
Fig. 1 Exchange system program structure in object-oriented model.

② 直列オブジェクトへのメッセージ転送：手続き呼出しに展開する（パラメータは Call-by-value）．

（4） Smalltalk で行われている実行時メソッドサーチは融通性に富むが，実行時オーバヘッドが大きいのでコンパイル時結合とする．なお，交換プログラムでは設計時にすべてのケースを考慮して開発するのでコンパイル時にメソッドを決定しても不都合はない．

（5） Smalltalk のクラスおよびインスタンスオブジェクトの概念は有用なので導入する．

以上に基づいた交換プログラムのオブジェクト構成モデル例を**図 1** に示す．

3. Chill の概要

Chill は，交換プログラム等の実時間多重処理システム記述用言語であり，データタイプ概念等は Pascal から（構文的には PL/I からも）影響を受けている．本言語は，以下のような特色を持つ．（Chill の具体例は後の記述例で示す．）

（1） Pascal 同様のモード種別（Chill ではデータタイプのことをモードと呼んでいる）とモードチェックを持つ強タイプ付言語でありモード規則に反する演算は許されないが，明示的モード変換機能を用いれば拡張した演算も可能である．

（2） データ宣言と手続き定義のカプセル化機能として，Modula[11)] の MODULE～END 同様のモジュール機能を持つ．モジュール外への参照許可は GRANT 文，他モジュールへの参照は SEIZE 文で明示指定する（それぞれ Modula の Export, Import に相当）．

（3） 軽量・効率的な並行プロセス機能を持つ．プロ

セス間の通信機能としては，直接宛先プロセスを指定してメッセージを送る**シグナル機能**を持っている．本機能は非同期型通信であり，後述のように並列オブジェクト間のメッセージ通信として非常に都合がよい．

(4) 交換処理では，発端末の信号到着，着端末の信号到着あるいはタイムオーバというように複数事象のうちからいずれかが発生するのを待っていて，いずれかの事象が発生したら対応する処理をする例が多い．Chill の並行プロセス機能は，複数事象を待って最初に到着した事象に応答するための**並列事象待ち機構**を持つので，記述性が高い．

(5) C. A. Hoare が提案した Monitor[9] 相当の機能（データ定義と手続き定義を REGION～END で囲んだもので排他的アクセスが保証される）を持つ．これにより条件付クリティカルリージョン等が容易に実現できる．

(6) 使いやすい例外処理機能を持ち，リソース捕捉失敗時などの処理を簡明に記述できる．

4. 並列・直列クラスオブジェクトの定義

Chill では，並列オブジェクトはプロセスとして，直列オブジェクトはインスタンスデータおよびルーチン群のカプセルとして実現できる．

4.1 並列オブジェクト

並列オブジェクトは，次の性質を有する．

① 並列オブジェクトは固有のプロセス制御域とワークスタック域を持ち，並列実行の単位となる（つまり並行プロセスの機能を持つ）．

② 並列オブジェクトに送られたメッセージはいったんそのオブジェクトが持つメッセージ行列に入る．つまり非同期通信とする（同期通信方式は送り手が受け手に強制同期されるので，よほどコンテクスト切換えが早くかつバッファプロセスを介在させない限り実時間システムを同期通信のみで作るのは難しい）．

③ 並列オブジェクトはスケジューラから制御を受けると，中断再開の場合は再開点から処理を再開，さもなければ自分のメッセージ行列から先頭メッセージを取り出し，それが要求する処理を行う．

Chill のプロセスは自律的動作主体であり，かつシグナルは宛先プロセス指定型のプロセス間非同期メッセージ通信そのものであるので，まさにわれわれのモデルの並列オブジェクトに適合する．**図2**に Chill プロセスによる並列オブジェクトの実現例を示す．

4.2 直列オブジェクト

直列オブジェクトは，以下の性質を持つ．

① メッセージの送り手は，直列オブジェクトにメッセージを送ると，対応したメソッドが実行されそれが終わるまで待って処理を再開する．

② オブジェクト固有のインスタンスデータとメソッドをカプセル化したもので，プロセス制御域，ワークスタック域など並行プロセスの機能は持たない．

③ メッセージを受けた直列オブジェクトは，メッセージ送り側の並列オブジェクトに従属して（i. e. 並列オブジェクトのワークスタック上で）手続き呼出し的にメッセージが指定したメソッドを実行する．

④ 直列オブジェクトは，複数のメッセージが到着した場合の応答の仕方で2タイプに分けられる．

(a) 排他アクセス型：複数のメッセージが到着していても単一メッセージしか受け付けない機構を持つ (Hoare の Monitor のように排他アクセスされる) タイプ．

(b) 非保護型：排他アクセスの機構を持たず複数並列オブジェクトから同時にアクセスされうる．複数の並列オブジェクトから並行アクセスされた場合に伴う

```
SIGNAL  Msg1 = (INT, BOOL);    ---  シグナル（プロセス間メッセージ）の定義
SIGNAL  Msg2 = (INT, INT) ;
DCL  PObj, CObj     INSTANCE;  ---並列オブジェクトを識別するための変数

Producer: PROCESS( ); --- 並列オブジェクト( プロセス) の定義
      ....
    DO FOR EVER;
      ....
      SEND  Msg1( i, j) TO  CObj;    ---メッセージ Msg1 を CObj に送る
      ....
      SEND  Msg2( i, K) TO  CObj ;   ---メッセージ Msg2 を CObj に送る
    OD;
      ....
END Producer;

Consumer: PROCESS( );
          ....
    DO FOR EVER;
      RECEIVE CASE
      (Msg1 IN x , y) :  .... Msg1受信時の処理(メソッド) ...
      (Msg2 IN x , z) :  .... Msg2受信時の処理(メソッド) ...
      ESAC ;
    OD ;
      ....
END  Consumer ;
PObj := START Producer(  ); --- 並列オブジェクトを生成しその Id をPObjに設定
CObj := START Consumer(  ); --- 並列オブジェクトを生成しその Id をCObjに設定
```

図 2 Chill プロセスによる並列オブジェクトの実現例
Fig. 2 Concurrent-object implementation with Chill processes.

危険はユーザ責任である.

直列オブジェクトは，基本的にはインスタンス変数とメソッド群のカプセス化であり，Chill のモジュール機能を利用して，**図3**のようにインスタンス変数をモード定義に，メソッドを手続きに対応させることにより，直列オブジェクト定義とすることができる．大部分のリソースは一つの並列オブジェクトに専有されて動くので直列オブジェクトとすることができる.

4.3 クラスオブジェクト

図3において，モジュール "CircleM" を Smalltalk のクラス定義に対比させて考えると，以下のことが言える.

① モード定義 "Circle" は，インスタンス変数のタイプ定義に相当する.

② 変数 "TotalNum" は，クラス変数に相当する.

③ 手続き "New", "GetTotalNum" は，クラスメソッドに相当する.

④ 手続き "Enlarge", "Move" はインスタンスメソッドに対応する.

```
CircleM :MODULE;
      GRANT  New, GetTotalNum, Enlarge, Move;  --- 外部からのアクセスを許可
      GRANT  Circle  FORBID ALL ;              --- [注1]
      MODE  Circle = STRUCT(Center  STRUCT( x, y  INT), r  INT);
            ---オブジェクトのインスタンス変数のモード定義
      DCL   TotalNum  INT  INIT:=0;
  ----
  New :PROC(  )  RETURNS( REF Circle); --ｵﾌﾞｼﾞｪｸﾄを生成するクラスメソッド
       DCL  p  REF Circle;
         p := ALLOC(Circle);
         p -> .Center := [0.0];   p -> .r := 1;
         TotalNum  += 1;          --- [注2]
         RETURN p;
       END;
  GetTotalNum:PROC( ) RETURNS(INT); -- ｵﾌﾞｼﾞｪｸﾄ総数を知るクラスメソッド
              RETURN TotalNum;
            END;
  ----
  Move :PROC ( ObjP  REF Circle, x, y  INT);
        ---オブジェクト(第一パラメータで指定)を移動するインスタンスメソッド
           ObjP -> .Center.x  += x;   --- [注2]
           ObjP -> .Center.y  += y;
        END ;
  Enlarge :PROC( ObjP  REF Circle, n INT);
          ---オブジェクト(第一パラメータで指定)を拡大するインスタンスメソッド
            ObjP -> .r  *= n;          --- [注2]
           END;
END  CircleM ;
  ----
DCL c1  REF Circle ;     ----  オブジェクトポインタの宣言
c1 := New(  );           ----  オブジェクト c1 を生成
Enlarge( c1, 10 );       ----  オブジェクト c1 を10倍に拡大
Move ( c1,  15, 7 );     ----  オブジェクト c1 を  [5,7] 並行移動
```

注 1) "GRANT Circle FORBID ALL;" は，外部アクセスを許すが，内部構造は見せないことを意味する．Modula II ならば，Opaque タイプを利用するところである.

注 2) $\alpha*=\beta$; は $\alpha:=\alpha*(\beta)$; の意味.

図 3 Chill による直列オブジェクトの実現例

Fig. 3 Sequential-object implementaion in Chill.

このように，モジュール定義をクラス定義に対応させると，クラスオブジェクト／インスタンスオブジェクト概念も自然に導入できる．また，MODULE～END の代わりに REGION～END とすると排他アクセス型のクラスオブジェクトを構成できる.

5. Chill による並列オブジェクト指向プログラムとプリプロセッサ展開

ここでは，Chill によるオブジェクト指向プログラミング手法を示す．Chill のままではオブジェクトおよびメッセージ転送の仕組みをプログラマが明記する必要があるが，オブジェクト指向の効果は十分に得られる．さらにプログラムの容易化と読解性の向上を図るには，簡単なプリプロセッサを使用してオブジェクト定義とメッセージ転送をオブジェクト指向言語並みに記述できるようにすればよい．図 4～7 では，左側にプリプロセッサ入力プログラム，右側にプリプロセッサで展開後の Chill プログラムを示した（複雑なプリプロセッサを用いれば高度な技巧も実現できるが[10] プログラムデバッグを考えると簡単な展開の方がよい）.

5.1 直列オブジェクトのクラス定義

（1） クラス定義

4.3 節で述べたように Chill のモジュール定義を次の内容を持つクラス定義として利用できる.

```
クラス名：CLASS ;
        Export/Import の定義
    CLASSVARS
        クラス変数の定義
    INSTANCEVARS
        インスタンス変数の定義
    CLASSMETHODS
        クラスメソッドの定義
    INSTANCEMETHODS
        インスタンスメソッドの定義
CLASSEND ;
```

図 4 にクラス変数 "*i*", "*j*" とクラスメソッド "New", インスタンス変数 "*a*", "*b*" とインスタンスメソッド "Run", "Delete" を持つクラス aaClass の定義例を

```
【プリプロセッサ入力記述】                    【プリプロセッサ展開形(変化行のみ示す)】
aaClass:CLASS ;                          ⇒  aaClass: MODULE;
  OBJECT   aa ;                          ⇒     GRANT ( ->aaClass)! aa FORBID ALL;
  EXPORT   New, Run, Delete;             ⇒     GRANT ( ->aaClass)! New, Run, Delete;
                                                    --- [注1]
  CLASSVARS                              ⇒    --CLASSVARS (コメント化: 以下同様)
    DCL  i, j  INT ;.
  INSTANCEVARS                           ⇒    --INSTANCEVARS
     MODE  aa = STRUCT ( a INT, b BOOL);
  CLASSMETHODS                           ⇒    --CLASSMETHODS
    New : PROC( x  INT) RETURNS(REF aa);
           DCL  p  REF aa;
           p := ALLOC( aa );
           p -> .a := x;
           p -> .b := FALSE ;
           RETURN p ;
         END;
  INSTANCEMETHODS                        ⇒    --INSTANCEMETHODS
   Run: PROC( SELF , x  INT, y  BOOL);   ⇒    Run: PROC( SelfP  REF aa, x INT, y BOOL);
         SELF.a := x ;                   ⇒        SelfP -> .a := x ;
         SELF.b := y ;                   ⇒        SelfP -> .b := y ;
          ....
       END;
  Delete:PROCESS (SELF )                 ⇒     Delete:PROCESS (SelfP  REF  aa );
          FREE  (SELF );                 ⇒        FREE  (SelfP -> );
        END;
  CLASSEND  aaClass;                     ⇒  END  aaClass;

 [注1]
 GRANT ( ->aaClass)! New ;
  は, 定義名  New を外部には
  aaClass!New という名前で見せる
  ことを意味する。
 [注2]
 SELF は自オブジェクトを示すキイワード
```

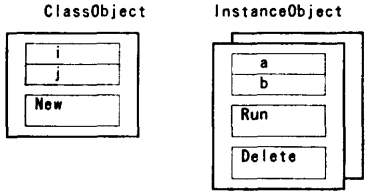

図 4 Chill による直列オブジェクトのクラス定義
Fig. 4 Class definition of sequential object in Chill.

示す.

(2) クラスオブジェクトへのメッセージ転送

クラスオブジェクトは, クラス名で指定(識別)できるので, クラスオブジェクトへのメッセージ転送は以下のように記述する. これによりメッセージ名で指定されたクラスメソッドが起動される. ここではプロシージャ名は名前の衝突を避けるために Chill のモジュール名修飾機能を利用して, クラス名で修飾した "**クラス名!メッセージ名**" としているが, 別手法を7章で述べる. 値が返る場合にはこれを代入文の右辺に書けばよい. **図5**のようにプリプロセッサにより, プロシージャ呼出しに展開される.

```
(a)オブジェクトポインタ変数等の宣言

  SEIZE   aaClass! aa, New, bb, Delete ; --- aaClassはクラスオブジェクトのID
  DCL  MyObj  REF aa ;                   --- MyObjはインスタンスオブジェクトのID

(b)クラスオブジェクト(本例のaaClass)へのメッセージ転送
        【プリプロセッサ入力】                【プリプロセッサ展開後】

  MyObj :=  aaClass  <<  New (10);   ⇒    MyObj := aaClass! New(10);

(c)インスタンスオブジェクト(本例のMyObj)へのメッセージ転送

  x  :=  MyObj << bb( i );           ⇒     x :=  aaClass! bb(MyObj, i );
  MyObj  << Delete(k);               ⇒     CALL  aaClass! Delete( MyObj, k);
                                           (第1 パラメータでオブジェクトを指定)
```

図 5 直列オブジェクトへのメッセージパッシング
Fig. 5 Message passing to sequential objects.

```
クラス名<<メッセージ (パラメータ並び);
        ⇩
CALL クラス名!メッセージ名
                (パラメータ並び);
```

(3) インスタンスオブジェクトへのメッセージ転送

インスタンスオブジェクトは任意個数生成できるので，どのオブジェクトであるかの指定が必要である．これには，インスタンス変数へのポインタ値を用いればよいので，インスタンスオブジェクトへのメッセージ転送は以下のように書くこととする．

```
オブジェクト ID<<<メッセージ名
                    (パラメータ並び);
          ⇩
CALL クラス名!メッセージ名
      (オブジェクト ID, パラメータ並び);
  (ここに，オブジェクト ID はインスタン
   ス変数へのポインタ)
```

プリプロセッサでは，図 5(b)に示すようにメッセージ名をプロシージャ名に変換し，かつ“オブジェクト ID (Object へのポインタ変数)”を第一パラメータとしたプロシージャ呼出しに展開する．

このように直列インスタンスオブジェクトへのメッセージ転送は，第一パラメータで目的オブジェクトを指定したプロシージャコールに展開される．Smalltalk とは違って，コンパイル時にメソッドを決定してしまうので，融通性は減るがオーバヘッドがなくなり実時間処理に耐えられるようになる．

5.2 並列オブジェクトの定義

(1) 並列オブジェクトの定義

Chill のプロセス定義は，並列オブジェクト定義そのものである．また，プロセス定義の外側のモジュールは 5.1 節同様にクラスオブジェクトとみなすことができる．クラスオブジェクトは常に直列オブジェクトである．例を**図 6** に示す．

(2) オブジェクトの識別法とメッセージ転送

Chill では START 文の実行によりプロセスが生成され，その識別 ID が INSTANCE モードの値として返される．したがって，並列オブジェクトの指定（識別）には生成時に返される識別 ID 値を用いればよいので，プロセスオブジェクトへのメッセージ送信は以下のように記述し，プリプロセッサが Chill の Send 文に展開する．ここに，メッセージ名は Chill のシグナル名である．

```
並列オブジェクト ID<<<メッセージ名
                    (パラメータ並び);
          ⇩
SEND メッセージ名 (パラメータ並び)
                TO 並列オブジェクト ID;
```

(3) 実時間処理とスケジューリング

並列オブジェクトは，実時間スケジューラの下で優先度対応に実行される．(Chill のスケジューラは，プログラムが設計開発して作り込める機構にしている．交換プログラム用スケジューラは，一般に周期割り込みで実行される High/Low の 2 レベルと，さらに優先度付 FIFO で実行される Base レベルとを持っている．)

5.3 分散処理環境でのリモートオブジェクトへのメッセージ転送

メッセージの受信側オブジェクトが送信側オブジェクトと異なるプロセッサ上にある場合，これをリモー

```
aa: CLASS ;                    ⇨  展開形は自明なので略す
  EXPORT  ....   ;
  SIGNAL  s1 = (INT, INT);
  SIGNAL  s2 = (INT, BOOL);
 CLASSVARS
    DCL  pp, cc  INSTANCE;
      -- INSTANCE は並列Objectへのポインタ
CLASSMETHODS    --- クラスメソッドの定義
  StartUp:
    PROC( );
      pp:=START producer( );
      cc:=START consumer( );
    END;

PROCESSDEFS
  Producer:   --- 並列オブジェクトProducerの定義
    PROCESS( );
      ......
      cc <<< s1(i,j) ;      ⇨  SEND  s1(i,j) TO cc ;
        ....
      cc <<< s2(i,b) ;      ⇨  SEND  s2(i,b) TO cc ;
        ....
    END;
  Consumer:   --- 並列オブジェクトConsumerの定義
    PROCESS( ) ;
      ....
      DO FOR EVER ;
        RECEIVE CASE
          (s1 IN x, y) :  -- メッセージ s1 受信時の処理 ---
          (s2 IN x, z) :  -- メッセージ s1 受信時の処理 ---
        ESAC ;
      OD ;
          ....
    END;
CLASSEND  aa ;
```

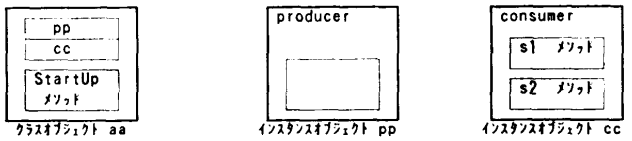

図 6 並列オブジェクトの定義
Fig. 6 Concurrent object definition.

トオブジェクトと呼ぶことにする.

(1) リモート並列オブジェクトへのメッセージ転送

並列オブジェクト間のメッセージ転送の実行は, Chill の並列処理用実行時ルーチンの役割である. 実行時ルーチンでは, メッセージの宛先アドレスを見て, これがローカル並列オブジェクトならばそれにメッセージを配送し, リモート並列オブジェクトであったらそのメッセージを通信ポート経由で目的サイトに遠隔転送する. すなわち言語レベルの機能としては, オブジェクト ID をグローバル化しておくだけで済む.

(2) リモートの直列オブジェクトへのメッセージ転送

直列オジブェクトへのメッセージ転送は, サーバプロセスを介したリモートプロシージャコールに展開する必要がある. 宛先がリモートの場合にはリモートプロシージャコールに展開する機能をプリプロセッサに持たせて, ソースプログラムではローカル/リモートを隠蔽することも可能ではあるが, サーバプロセスの適用法等はプログラマに判断させた方がよいので, リモート直列オブジェクトの場合はプログラマにサーバプロセスインタフェースを明示記述させるものとする.

6. Chill によるインヘリタンスの実現

Chill の GRANT-SEIZE 機能は, GRANT 側モジュールで GRANT された対象を SEIZE 側モジュールで使えるようにする働きを持つ (Export-Import の機能である). したがって, 親クラス側で GRANT しているインスタンス変数モードおよびメソッドを子クラス側で SEIZE することにより, 子クラス内で親クラスの機能を使える. この機構により, オブジェクト指向のインヘリタンス機能を模擬できる. **図7**に, クラス Caa の機能をサブクラス Cdd が引き継いでいる例を示す.

本インヘリタンス手法は, オブジェクト変数のモード定義でインヘリタンス関係を意識する必要がある

```
【プリプロセッサ入力記述】                ⟹   【展開形(変化行のみ示す)】

(1)スーパクラス側の記述
Caa :CLASS ;                              ⟹  Caa :MODULE;
     EXPORT  aa ... ;                     ⟹       GRANT ( -> Caa)! aa ... ;
  CLASSVARS
  INSTANCEVARS
    MODE  aa =
        STRUCT( a1 INT, a2  INT );
  CLASSMETODS
    New:PROC( ) RETURNS( REF aa);
        ....
     END;
  INSTANCEMETHODS
    Maa:PROC(SELF ... );                  ⟹    Maa:PROC(SelfP  REF aa ...);
        SELF .a1  :=  ... ;               ⟹          SelfP-> .a1  := ... ;
        ....
     END ;
  CLASSEND  Caa ;                         ⟹   END  Caa ;
-------------------------------------------------------------------------------
(2)サブクラス側の記述
Cbb  :SUBCLASS OF Caa ;                   ⟹  Cbb  :MODULE ;
                                          ⟹      SEIZE  Caa  ALL;
   EXPORT   bb ... ;                      ⟹      GRANT  ( -> Cbb)! bb ... ;
   CLASSVARS
   INSTANCEVARS
     MODE  bb =
        STRUCT( F   Caa! aa,
                b1  INT,  b2  CHAR);
  CLASSMETODS
    New:PROC( ) RETURNS( REF bb);
        ....
     END;
  INSTANCEMETHODS
    Mbb:PROC(SELF  ...    );              ⟹    Mbb:PROC(SelfP  REF bb ...);
        SELF.b1 := ...   ;                ⟹         SelfP-> .b1 := ...   ;
        SELF.F.a1 := ...  ;               ⟹         SelfP-> .F.a1 := ...;
        --- 親の a1 フィールド
        ....
     END ;
  CLASSEND Cbb ;                          ⟹   END  Cbb;
-------------------------------------------------------------------------------
(3)メソッド転送の記述
ObjA := Caa  << New ( );                  ⟹    ObjA :=  Caa! New( );
ObjB := Cbb  << New ( );                  ⟹    ObjB :=  Cbb! New( );
     -- ObjA, ObjB を夫々生成

ObjA << Maa( i );                         ⟹    CALL   Caa! Maa( ObjA, i);
     -- aaClassのオブジェクトに aaClassのメソッドを作用

ObjB << Mbb( j );                         ⟹    CALL   Cbb! Mbb( ObjB,  j);
     -- bbClassのオブジェクトに bbClassのメソッドを作用

ObjB << Maa( i );                         ⟹    CALL   Caa! Maa( ObjB, i);
     -- bbClassのオブジェクトに aaClassのメソッドを作用
```

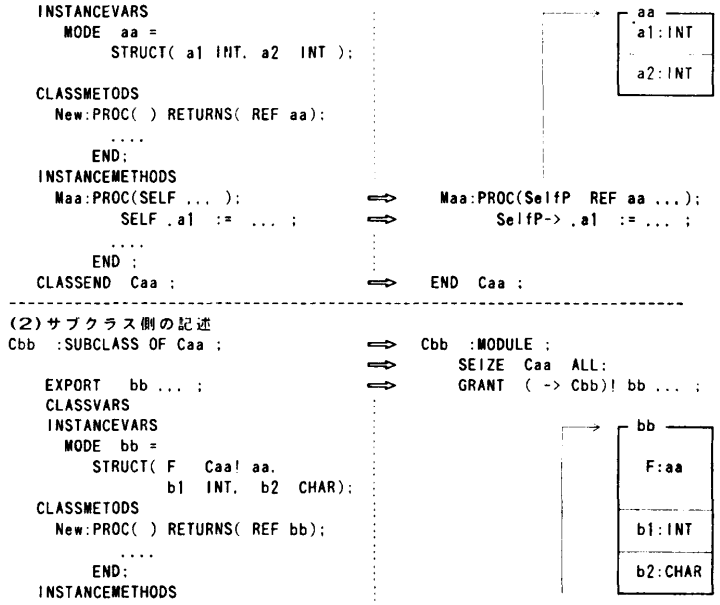

図 7 インヘリタンスの実現
Fig. 7 Implementation of inheritance.

が, 実現が容易であること, マルチプルインヘリタンスも可能であることなど, 実用的な効果は大きい.

なお, 図7のままではモードエラーが出るので強制モード変換が必要であるが煩雑なのでここでは略記してある. Chill の強制モード変換とは, $\beta(\alpha)$ という記述 (ここにαは式もしくは変数, βはモード名) により, αのモードをβとして解釈させる機能である. 強制モード変換の使用が煩わしければ, Chill では Type-Free-Pointer である PTR モード (PL/I の POINTER タイプと同等) も用意されているのでタ

イプチェックをプリプロセッサに行わせて PTR を使う手もある．なお，モードチェックと融通性を両立させるには，REF モードの代入規則の以下のような拡張が望ましい．

```
MODE  a =STRUCT (f INT);
MODE  ab=STRUCT (f INT, g BIT (32));
DCL  Pa   REF  a ;
DCL  Pab  REF  ad;
Pa := Pab; …これを許す
Pab:= Pa ; …これはエラーとする
```

7. オブジェクトとメソッドのリンク

5章ではメソッド名の衝突を避けるためにクラス名で修飾する手法を説明したが，下記手法も効果的である（**図8**参照．われわれのオブジェクト指向型交換プログラムの研究には本手法を採用）．

- 各メソッドへのポインタを内容とする構造体 “EntryTable” を設ける，
- 各インスタンス変数は本 EntryTable へのポインタを持たせる．
- 実行時にはインスタンス変数がポイントする EntryTable にアクセスして目的のメソッドを決定する．

本手法では，内部構造が異なっても外部インタフェースが同一な（つまり EntryTable のモードが等しい）オブジェクトは，区別なく扱うことができる．例えば図8では，オブジェクトポインタ変数 BStack に StkA が代入されていても StkB が代入されていても同一に操作できる．交換プログラムの場合では，中継回線オブジェクト（回線の信号方式ごとに内部構造が異なる）の外部インタフェース（EntryTable のモード）を同一に設計しておけば，空き回線ハントして得られた中継回線オブジェクトを信号方式によらず同一に扱える．

8. 他の並列オブジェクト指向言語との関連

本論文の並列オブジェクトモデルは，Actor モデルと Lisp をベースとする ABCL/1 や Smalltalk に上位 Compatible な Concurrent-Smalltalk とは以下のような類似性もある（ここではそれぞれ ABCL, CStalk と記す）．

（1） ABCL のオブジェクトは並列型であるが，“受けるメッセージがすべて NOW 型で，各メソッドが Reply を実行すると終了し，同時に複数のメッセージが来ないことが保証されている” オブジェクトの場合には，本稿の直列オブジェクトのように実現して効率化を図れる．

（2） CStalk は Smalltalk-compatible な NonAtomicObject と，到着メッセージがシリアライズされ FIFO 順に実行される AtomicObject を持つ（本稿の排他アクセス型クラスオブジェクトに類似．なお本稿の並列オブジェクトもシリアライズの性質を持つ）．

```
【定義側】
StkA: CLASS ;   GRANT  ( ->StkA)! Stack, Methods, Entry, New ;
CLASSVARS
  MODE  Methods = STRUCT( Push  PROC(SELF, INT),
                          Pop   PROC(SELF)RETURNS(INT)
                          Free  PROC(SELF)          );
  DCL  Entry Methods INIT=[Push, Pop, Free]; -- Entry-Table
INSTANCEVARS
  MODE Stack =
       STRUCT(E  REF Methods, -- EntryTableへのポインタ
              ..... ) ;     -- Instance変数
CLASSMETHODS
    New:PROC( )RETURNS(REF Stack);
        DCL  p  REF Stack;
        p :=ALLOC(Stack);        --メモリ割付
        p-> .E := ADDR(Entry);  --メソッドリンク設定
        ......
        RETURN  P;
    END New;
INSTANCEMETHODS
    Push: PROC(SELF, x INT);          .....  END Push;
    Pop : PROC(SELF) RETURNS(INT);  .....  END Pop;
    Free: PROC(SELF);              .....  END Free;
END   StkA  ;
----------------------

StkB: CLASS ;   GRANT  ( ->StkB)! Stack, Methods, Entry, New ;
  ----  StkAと内部構造は異なるが，外部インタフェースは同一
END   StkB  ;

【使う側】
---【プリプロセッサ入力】  ⇒  【プリプロセッサ展開後】
DCL  AStack REF  StkA! Stack ;
DCL  BStack REF  StkB! Stack ;

AStack := StkA<< New();     ⇒    AStack := StkA! Entry.New( );
            --- StkA のメソッドNewが実行される．

BStack := AStack ;          ⇒    BStack := StkB! Stack(AStack);
         ---  このようにモード変換して代入をしても以下の操作は安全

BStack << Push(i);          ⇒    CALL  BStack->.E->.Push(BStack, i);
         ---StkBでなくStkAのメソッドPushが実行される．

i := BStack << Pop();       ⇒    i :=  BStack->.E->.Pop (BStack);
         ---StkBでなくStkAのメソッドPopが実行される．
```

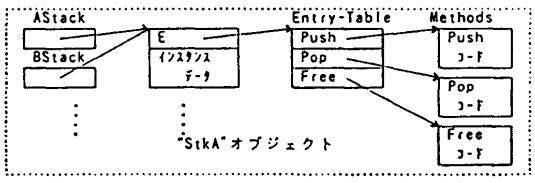

図8 外部インタフェースの等しいオブジェクトの定義と使用
Fig. 8 Object definitions and applications of the same external interfaces.

非同期メソッドコールと返答演算“↑↑”の使用により，各オブジェクトを並列実行できる．

(3) 並列オブジェクトへのメッセージ転送は，ABCL の過去型，CStalk の非同期メソッドコールに類似する．

(4) 直列オブジェクトへのメッセージ転送は，ABCL の現在型，CStalk の同期メソッドに類似する．

本論文の手法では，このように既存言語を用いながら一般性のあるオブジェクトモデルを実現しており，かつベース言語の効率を活用できる．

9. 評　価

本論文の手法は既存言語上でのオブジェクト指向記述法として，以下のように効果的である．

(1) 並列オブジェクトと直列オブジェクト概念の導入により，一般的かつ効率的な並列処理モデルを実現しており，実時間超多重処理に適用可能となった．

(2) 直列オブジェクトの導入，直列オブジェクト宛メッセージ転送のプロシージャ呼出しによる展開，コンパイル時メソッド結合，C++[10] のようにまとまった構成要素のみをオブジェクトとして扱う等により，実時間処理に耐える効率を達成している．これらによりオブジェクト指向の融通性は多少低下するが，交換プログラムにオブジェクト指向を適用できる効果は大きい．直列オブジェクト自体の従来手法に対する動的オーバヘッドは，単機能ごとにメソッド化されることによるプロシージャ（メソッド）呼出し増加，プロシージャ間接呼出しの増加などによるもので，小規模試作プログラムの経験では2割程度と見込まれる．

(3) 本論文で提案した Chill によるオブジェクト指向プログラムでは，プログラムをオブジェクトとその間のメッセージ転送というモデルで記述できるので，効果的にプログラムを構造化できる．ただし，オブジェクト指向にするための仕組みをプログラマが一々記述する必要があるので煩わしい．そこで，オブジェクト構造やメッセージ転送をさらに明示して，さらに記述法・読解性も本格オブジェクト指向言語並みに向上させるには，本論文で併せて提案している簡易なプリプロセッサを用いればよい．

(4) モジュール間の GRANT-SEIZE 機構の活用により，制限付ながらインヘリタンスを実現できる．なお，Chill は強タイプ付言語なので，インヘリタンスを記述する場合には強制タイプ変換が必要になり，煩わしさは残る（プリプロセッサで処置することも可能）．なお，Chill のタイプ規制を6章で述べたように変更すれば，コンパイル時チェック能力を損なわずに強制タイプ変換を不要にすることも可能である．

理想的な実時間並列オブジェクト指向言語の開発に比較すれば，本手法は既存言語による暫定解ではあるが新言語開発の時間・費用・プログラマ教育の点では利点がある．

10. あ と が き

実時間と数千のオーダの超多重処理を特徴とする交換プログラムにもオブジェクト指向概念は有効であるが，並列処理機能を持ち実行効率のよいオブジェクト指向言語を即座に開発導入することは困難である．本論文では既存の並列処理言語を用いることにより，効率のよい並列オブジェクト指向プログラムが実現可能であることを示した．われわれは，本手法による交換プログラムの研究を進めておりオブジェクト指向の効果を確認している．

本論文ではベース言語として Chill を選んだが，本手法は Modula II[11] 等他の言語でも適用可能である．

参 考 文 献

1) Goldberg, R.: *Smalltalk-80*, Addison-Wesley (1983).
2) Dahl, J. et al.: *The Simula 67 Common Base Language*, Norwegian Computing Centre, Forskningsveien 1B, Oslo 3 (1968).
3) Hewitt, C. E.: Viewing Control Structures as Patterns of Passing Messages, *Artif. Intell.*, Vol. 8, No. 3, pp. 323-364 (1977).
4) CCITT-Recommendation: Z. 200 CCITT High Level Language Chill (1980, 1988 (Revised)).
5) 松尾，丸山：交換用プログラム言語 CHILL，電気通信協会 (1985).
6) 米沢，柴山，Briot, J.-P.，本田，高田：オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1，コンピュータソフトウェア，Vol. 3, No. 3, pp. 9-23 (1986).
7) 横手，所：並列オブジェクト指向言語 Concurrent Smalltalk，コンピュータソフトウェア，Vol. 2, No. 4, pp. 582-598 (1985).
8) Yonezawa, A. and Tokoro, M. (eds.): *Object-Oriented Concurrent Programming*, The MIT Press (1987).
9) Hoare, C. A. R.: Monitors: An Operating System Structuring Concept, *Comm. ACM*, Vol. 17, No. 10, pp. 549-557 (1974).
10) Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley (1986).
11) Wirth, N.: *Programming in Modula-2*, Springer Verlag (1982).

(平成元年 3 月 9 日受付)
(平成元年 10 月 11 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 交換システム研究所勤務．電子交換機の実時間増設方式，高水準言語と最適化コンパイラ，交換プログラム構造などの研究実用化に従事．1975～80年まで CCITT の Chill の設計に参画．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)．電子情報通信学会会員．

渡部 信幸（正会員）

1959年生．1982年千葉大学工学部電子工学科卒業．同年日本電信電話公社入社．現在 NTT 交換システム研究所勤務．電子交換機の制御プログラム，交換プログラム構造などの研究実用化に従事．電子情報通信学会会員．