分散システム記述用言語 DIPROL

小森 斉* 田中英彦, 元岡 達 (東大・工)
*(現在 富士通勤務)

1 はじめに

情報処理が社会のすみずみに根を張りつつある今日、社会的にきめ細かいサービスをするシステムは地理的に分散した形態をとっていくであろうし、小規模な用途においても、複数計算機を通信回線などで結んだシステムの実現性は高くなってきた。

本稿では主記憶を共有せず、何らかの通信手段で、互いに疎に結合された計算機システムのために設計した言語DIPROLについて述べる。[1]

分散型のシステムが、長期にわたり広い範囲で使われることになるなら、そのシステムを記述する言語に関しては、(ソフトウェアシステムの開発、保守, 再構成のことを重視する立場から言って,) できるだけ一様な言語の使用が望ましい。単一のシステム記述言語を採用すると、全体の見通しの良さだけでなく、システムの柔軟性を確保できるという大きなメリットがある。

本稿で述べる言語DIPROLはシステム記述言語として、分散システム上のすべてのプロセスを記述の対象とする。しかし、システム記述言語とはいっても分散システム上での可搬性を重視している点で高級言語の趣きをもつ。

2 設計方針

分散システムの結合が疎であることから、ある程度まとまった機能を、単一の主記憶上に載せることが、性能上必要である。その最小単位として、機能の並列性, 並行性の単位であるプロセス(あるいはタスク)をとる。よって、プロセスは、計算機間にまたがることのない、独立に実行しうる制御の流れである。

計算機間の通信の遅延が大きいので、なるべく、独立に実行できる通信動作は並列に行なえることが望ましい。このためにはプロセスの実行の流れと外部への働きかけの実行とを並列動作可能にし、複数の通信を並列にひき起すことが書けるべきである。

以上のことがらをふまえて、以下の方針を立てた。

(1) プロセス間の共有記憶の排除

異なる計算機に載っているプロセスの間では、共有記憶は直接使えないし模擬しても、遅延時間を無視した使い方に終始しがちで意味がないので、もし許すと、プロセスの配置に対する柔軟性を圧迫する。ひとまとまりのプロセスを、同じ計算機に載せることを示す言語上の枠組みを用意し、その中ではモニタなどの、より安全な共有記憶の機構を許す方針もありうるが[1][2], 本言語ではこのような共有記憶も採用していない。それは、モニタ等の受動性は、並列実行の可能性を記述するにあたって本質的に必要なものではないし、[3] プロセス間の直接の連絡を間接的にしてしまうことによる。こうして本言語は、プロセスの間の共有記憶を一切排除し、より一様で単純な形をとる。この方針はプロセス自身についても貫ぬいている。つまり、プロセス中を流れる制御の流れは単一に限り、DP[4]で提案されているような、プロセス中の複数の流れが、プロセス内の変数を介して干渉しあうスタイルは採用しない。

(2) プロセス間通信とそのインターフェース

プロセス単位での可搬性を増し、プロセスあるいは、ひとまとまりのプロセスの集まりを、ある機能をもった単位として活用することができれば、分散型システム上のプログラミングはより系統的

に進められるだろう。そのためには、プロセスあるいはその集まりが果すはずの機能を、外部に対してはっきりさせなければならない。(1)の方針より本言語ではプロセス間の相互作用はプロセス間の通信動作に限ることにした。そこでプロセス間通信のためのインターフェースを各プロセスごとにポートと呼ぶ形で設定し、そのプロセスの機能を代表させる。すべてのプロセス間通信はこのポート間で行なわれ、プロセスの協調や競合もプロセス間通信による。だから並行プロセスの記述に関する信頼性を確保するためには、機能を代表するポートに、そこで起こすプロセス間の通信の性格やその起動のされ方を記述して整合性の検査を行なえるようにすることが重要になる。

以上のポートによるインターフェイスの明確化は、プロセスの集まりに対しても繰り返し行なっていけるようにするべきである。

(3) プロセスの動的生成の不採用

プロセスの動的な生成のための機構は、本言語内に用意しない。新しくプロセスを作るには、それを載せる計算機上で、各種の資源を獲得しなければならない。もし言語の一機能として、Ada[4]のようにプロセスの動的な生成能力を与えるなら、やはり分散型システムのどの計算機にも生成することができるべきである。この一様性を達成するには、実行時にあらかじめ各計算機の資源の管理と連絡をとり、プロセスを作るだけの用意がなければならない。しかし、各計算機の資源の管理は、本言語で書こうとする対象であるし、実行時に仮定する核機能はなるべく少なくしたいので、プロセスの動的生成能力は言語の機能には含めない。

(4) 信頼性の確保

分散システムでの信頼性確保の手段として、抽象化データ型の手法を取り入れる。プロセス間の明示的な相互作用だけで、種々の資源や権利を記述しようとすると、管理側のプロセスだけでなく、ユーザ側のプロセスも、ある手順を守っていなければ管理が破綻するという場合がおこりがちである。

そこで、データに対して許される操作の範囲を、プロセスごとに指定できるような、抽象化データ型の手法を言語に導入する。[6] これによってPLITS[7]のA-setと似た安全性の確保ができる。

本言語は上述の(4)の方針にもあるように、システムの記述言語として高級言語よりのものを目ざしているので、コンパイラ試作の便も考慮し、Pascal[8]を母体として設計を行った。

3 DIPROLプログラムの構成

DIPROLのコンパイル単位はプロセスの集まりを記述し、それらの機能を代表するポートを提示する。この単位の中に含まれるプロセスは特に同一の計算機上に載せなければならないわけではない。プロセスの記述においては、そのプロセスを載せる計算機を論理マシン名で示す。プロセスの配置は論理マシン名に、現実の計算機を割りつけることによって決める。この世界を模式的に示すと、図1のようになる。

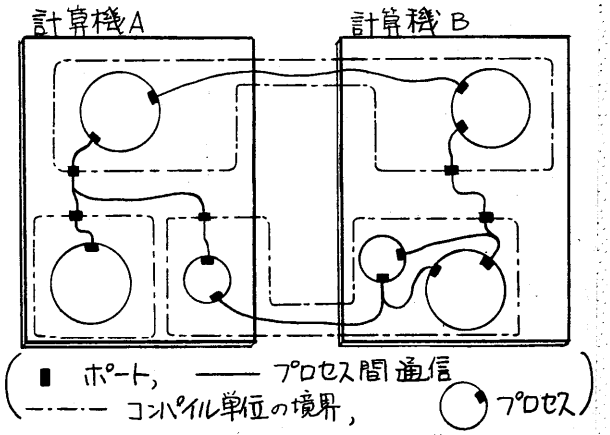


図1. DIPROLのコンパイル単位を基にした世界

コンパイル単位の構文グラフは図2のとおり。では、このコンパイル単位の要素について説明していく。

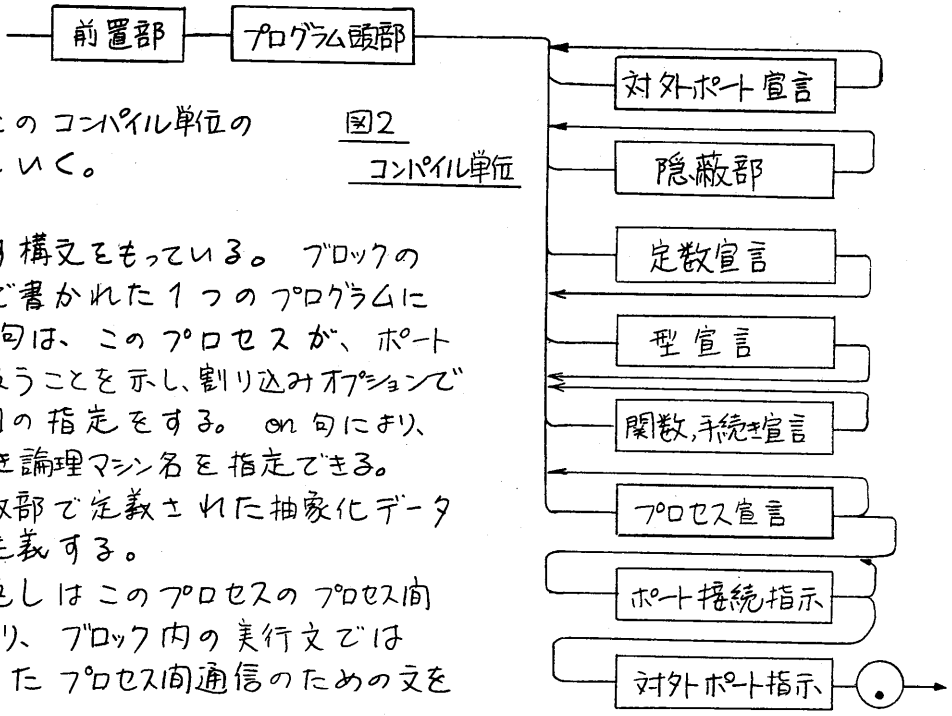


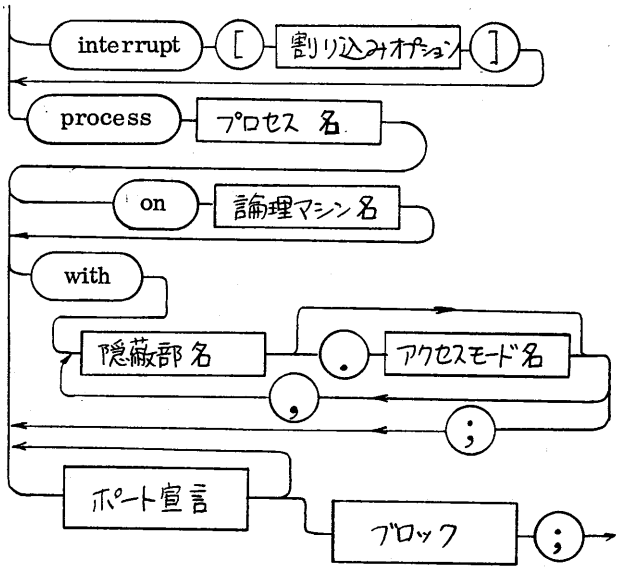
図2 コンパイル単位

(1) プロセス

プロセスは図3で示す構文をもっている。ブロックの部分はだいたいPascalで書かれた1つのプログラムに相当する。interrupt句は、このプロセスが、ポートのように割り込みを扱うことを示し、割り込みオプションで割り込み名とその要因の指定をする。on句により、このプロセスを載せるべき論理マシン名を指定できる。with句によって、隠蔽部で定義された抽象化データに対する操作能力を定義する。

ポート宣言の繰り返しはこのプロセスのプロセス間通信のインターフェースであり、ブロック内の実行文ではこれらのポートを介したプロセス間通信のための文を書くことができる。

図3 プロセス宣言



(2) プロセス間通信のための文

プロセス間の通信のための文は、そのプロセスで宣言されたポートに対する4種類のコマンドである。図4参照。。

1. 呼び出し型：ポート名で始まる書き方をする。ポートを起動し、相手ポートとの通信が終了するまで、プロセスの流れを停めておく。
2. 起動型：ポート名にstartを前置する書き方では、このポートに通信を始めるように指示するが、その通信動作とはかかわりなく、プロセスは走りつづける。

これら2種の書き方においては、ポート名の後に、送信データあるいは受信のためのデータの格納場所を指示する。（≫は受信，≪は送信である。）

割り込みオプション

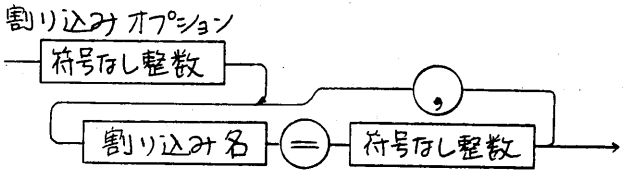
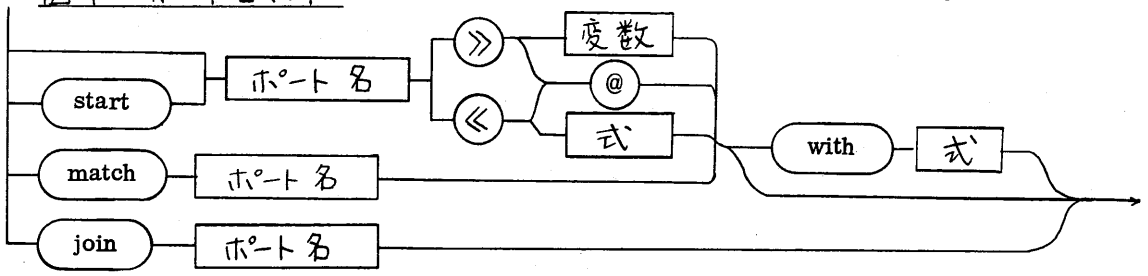


図4 ポートコマンド



通信動作はCSP[9], Ada[5]等と同様に両側の同意が成立してはじめて行われる。すなわち、接続されている両側のポートに呼び出し型あるいは、起動型のコマンドが発行されたことがわかった時点で、通信が起こる。

3. match を ポート名に前置すると、相手ポートが、上述の 通信実行の条件を満たすまでこのコマンドを発したプロセスを待たせる。このコマンドでは通信は実行されない。

4. join を ポート名に前置すると、すでに start によって 起動しているそのポートの通信終了まで、プロセスを待たせる。

プロセスと並列に動作する受信動作のデータの格納場所が、プロセスにより自由に扱えるのは危険であるから、このような受信データの格納場所は、ポートに付属するポートバッファという特別な変数に制限する。コマンドの記法としては ≫ の受信シンボルの後に @ を置いて表現する。この位置以外では、ポートバッファはポート名に @ を後置した形でアクセスできるが、ポートが起動されている際中(startからjoinまで)は、これをアクセスしようとすると、例外(ASYNCERR)を発生する。

joinを除いては、with句で、相手ポートを1つにしぼることができる。ここの式は、ポートID型という既定の型だけが許される。指定されたポートIDの値が、ポートに接続されている相手ポートのなかにないと、ポート名と同じ名の例外を生じる。また、joinのコマンドを、startにより起動したまま以外のポートに対して出すと、同様の例外を生じる。

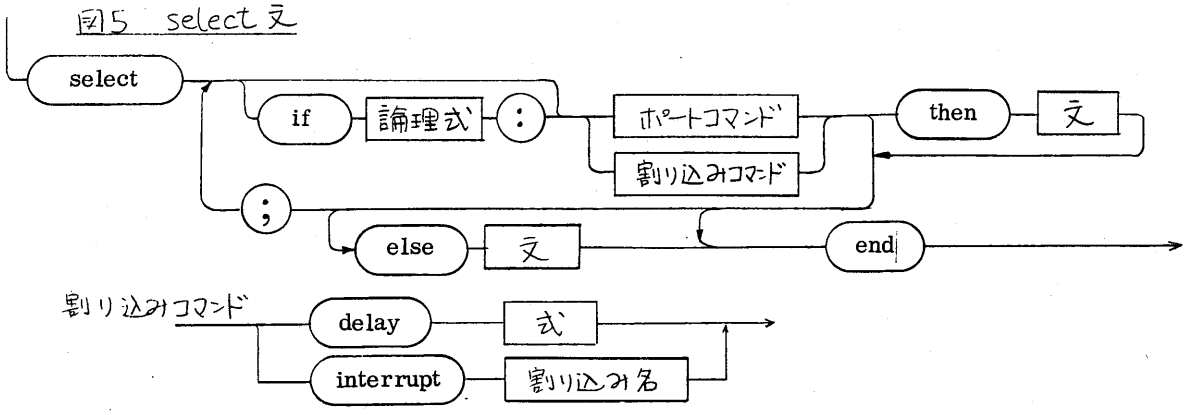
(3) 非決定性の選択待ち (select文)

あらかじめ生起する順序が定まらないような事象をプロセスだけの世界でうまく扱うには、この言語要素が必要である。図5に構文を示す。処理の内容はDijkstraのガーデッドコマンドのやり方である。ある条件を満した選択項目のうちから1つが選ばれて実行される。まず選択項目の頭についている論理式が評価され、偽の項目は選択の範囲からはずされる。(論理式の評価の順序は規定しない。)ここですべての選択項目が不通で、else以下があれば、else以下を実行する。else句もないと、例外(SELECTER)を発生する。選択項目が残ったなら、それぞれのポートコマンド、割り込みコマンドのトリガ条件のどれかが成立するまで待つ。成立しているトリガ条件をもつ項目のうちから1つを選んで、続くthen以下を(もしあれば)実行して、select文を終わる。

ポートコマンドのトリガ条件は、joinを除き通信の意向が相手ポートより寄せられていることである。joinのときは、起動されたポートの通信が終了したことである。

割り込みコマンドは単独の文としても使える。この場合 delay は後続の式で

図5 select文



与えられた秒数の間プロセスを待たせ、interruptコマンドは指定した割り込み名に対応する割り込み要因が発生するまでプロセスを待たせる。

選択項目中でのトリガ条件はdelayの方が、時間の尽きること、interruptの方は割り込み要因が生じていることである。

delayの式が0以下のときは0とみなされる。

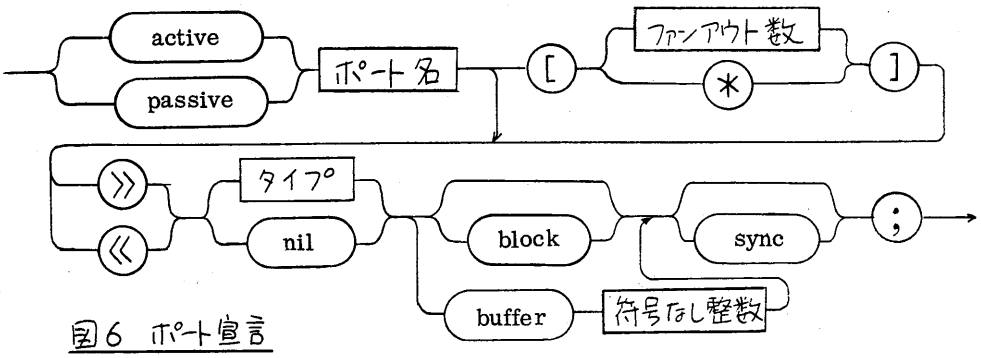


図6 ポート宣言

(4) ポート宣言

ポート宣言の構文を図6に示す。ポート宣言はポート名や種々の属性を宣言し、誤りの防止に努めている。

ファンアウト数は、宣言するポートに、相手ポートをいくつまで接続できるかの余裕を示す。正整数で示すか、＊によって、制限のないことを示す。（省略値は1。）

送受を区別するシンボル≪,≫の後に、送受されるメッセージのデータのタイプを指定する。nilは送受データが空であることを示す。

bufferの指定はプロセスの外側にポートに直列にFIFOのバッファを設けることを意味し、そのサイズをメッセージ単位の数で指定する。このFIFOバッファは、通信系に複数メッセージのアウトスタンディングを許し連続転送にも役立つ。プロセスからは、ポートの先に、バッファリング用のプロセスがあるように見える。

block/syncは、接続される両側のポートが、同期に関して整合性をもつことを確認するためのものである。blockを指定するのは、そのポートに対してのポートコマンドを呼び出し型のものだけに限り、必らず待ち合せすることを表明する場合である。syncは相手のポートがblockを表明しなければならないという指定である。

active/passiveの区別は通信動作の開始の意向を自から相手ポートへ伝えるか否かを示す。passiveなポートの方に、通信開始を判断する情報が集まるので、passiveなポートの方が機能が大きい。

ポートとポートの接続に際してなされるチェックは、メッセージの型の整合性、sync指定にblockが対応していること、さらにpassive同士のポートの接続の禁止、ファンアウトの余裕の検査、送受の向きの正しい組み合せなどがある。

ポートの属性と、プロセス内のポートの使い方の整合性のチェックも行なう。まず、blockの表明が守られていること、送受信データ型の整合性、送受の方向を検査する。さらに、select文の選択項目に表われるポートコマンドは、（joinを除き）すべて、passiveなポートを対象にしなければならない。これは分散システム上での決断が、局所的に行なわれるために重要な制約である。matchに関しては、いつでも対象となるポートがpassiveでなければならない。

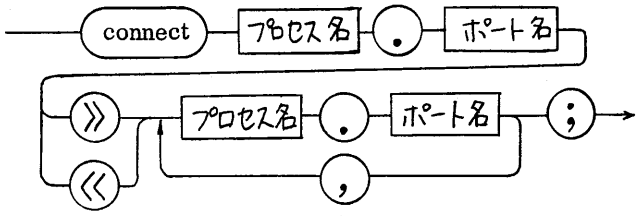
ポート宣言の構文に現われていない制約としては、ファンアウト値を2以上にできるのは、passiveなポートだけであり、bufferの指定は、ファンアウトが1のポートに限るということがある。

(5) ポート接続指示他

ポート接続指示は、コンパイル単位内の静的なポートの接続を記述する。構文を図7に示す。この指示が(4)で述べた条件を満すことはコンパイラによってチェックされる。

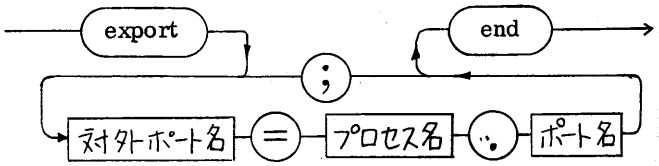
対外ポート宣言は、コンパイル単位外に見せるポートの宣言であって、プロセスのポート宣言と同じ構文である。

対外ポート指示は、コンパイル単位内のプロセスのポートのうちのどのポートを対外ポートとして割り当てるかを指示する。(構文は図8に示す。) 対外ポートと、対応させられたプロセスのポートの属性の不整合がチェックされるし、ファンアウトの余裕も考慮される。



↑図7 ポート接続指示

↓図8 対外ポート指示



(6) 隠蔽部

抽象化データ型の一まとまりの定義をして、これらに対する操作を元とする集合の部分集合に名前をつけて、操作の能力をクラス分けしている。これらをアクセスモードと呼ぶ。アクセスモードは隠蔽部ごとに1つ以上定義できる。プロセスはこのアクセスモードを隠蔽部名とともにwith句で掲げることでその操作能力を使うことができる。図9に構文を示す。

init ～ endの間の文の並びは、その型の実体が生成されるときの初期化を表現する。この文の並びの中では、初期化される型の名と等しい名をもった変数を初期化するように記述する。

続く手続き又は関数宣言の繰り返しで、上述の型に対しての許される操作を記述する。

access以降の部分で、アクセスモードの定義をする。1つ1つのアクセスモードは、上述の手続き又は関数宣言で定義された名前を要素とする集合の定数のように記述する。

プロセスがwith句でアクセスモードを引用すると、まず、その隠蔽部で定義した型の名が見えるようになる。そしてさらにアクセスモードの指定する手続きや関数によって操作ができるようになる。

withによるアクセスモードの列挙は同一の隠蔽部ごとには集合の和として操作能力が単純に増えるが、異なる隠蔽部のアクセスモード間で型名や手続き名が重複しうる。この場合、型なら後から出てきたものが優先し、手続き名、関数名については二重定義のエラーとなる。今のところこれを回避する手段は用意していない。

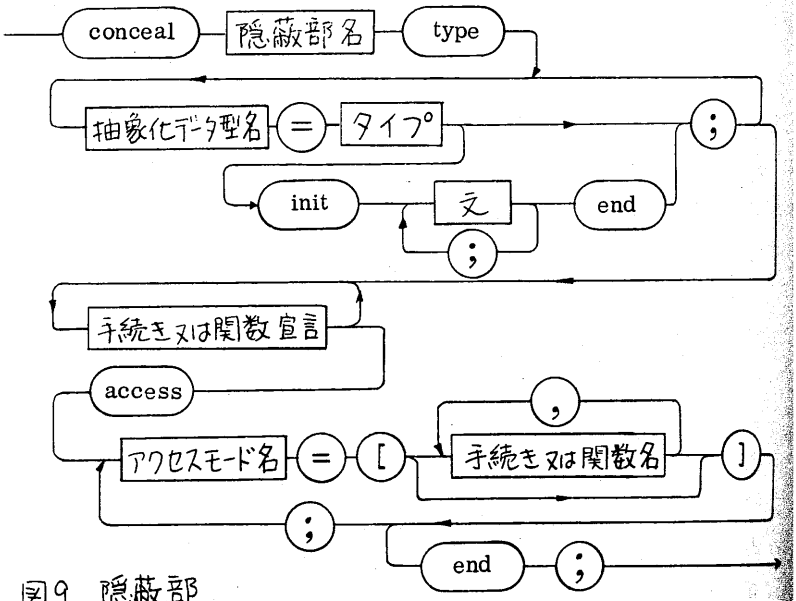


図9 隠蔽部

DIPROLは この他 Adaに近い例外の取り扱いができるが省略する

```
*)
 1       18 CONST MAXUSER=128;
 2       18       MAXINTEGER=32767;
 3       18 TYPE  RQBLK=RECORD WHO:1..MAXUSER;
 4       18                    TIME:INTEGER;
 5       18                    OKPORT:PORTID
 6       18                   END;
 7       18 PROGRAM SHORTSCHEDULE(ABCDE);
 8       18  PASSIVE Q[127]>>RQBLK;
 9       18  PASSIVE O[127]<<CHAR SYNC;
10       18  PASSIVE L[127]>>CHAR ;
11       18
12       18 PROCESS SJN ;
13       18   PASSIVE REQUEST[128]>>RQBLK;
14       18   PASSIVE OKACK[128]<< CHAR SYNC;
15       18   PASSIVE RELEASE[128]>>CHAR ;
16       18   VAR  QUEUE: SET OF 0..MAXUSER;
17       28        RANK: ARRAY[0..MAXUSER] OF RQBLK;
18     1318        USER,NEXT,I:0..MAXUSER; RB:RQBLK;
19     1334        MIN: INTEGER;
20     1336   BEGIN
21        3      REPEAT
22        5        SELECT
23        5          REQUEST>>RB THEN WITH RB DO BEGIN
24       13              QUEUE:=QUEUE+[WHO]; RANK[WHO]:=RB;
25       24              NEXT:=0
26       24              END;
27       28          RELEASE>>@ THEN USER:=0
28       34         END (*SELECT*);
29       47
30       47        IF (NEXT=0) AND (QUEUE<>[]) THEN BEGIN
31       55              MIN:=MAXINTEGER;
32       57              FOR I:=1 TO MAXUSER DO
33       65                  IF ( I IN QUEUE ) THEN
34       69                        IF RANK[I].TIME<MIN THEN BEGIN
35       77                                  NEXT:=I; MIN:=RANK[I].TIME
36       84                                  END
37       86              END;
38       90        IF (USER=0) AND (NEXT<>0) THEN BEGIN
39       98              OKACK<<@ WITH RANK[NEXT].OKPORT;
40      104              USER:=NEXT; NEXT:=0
41      107              END
42      110       UNTIL FALSE
43      110   END
44      112   ;
45        3 PROCESS USER1;
46        3   ACTIVE REQ<<RQBLK;
47        3   ACTIVE OKAK>>CHAR BLOCK;
48        3   ACTIVE REL<<CHAR;
49        3   BEGIN
50        3   (* SOMTHING  BEFORE REQUEST *)
51        3     REQ<<@;
52        6     OKAK>>@;  (* WAIT FOR GRANT *)
53        7   (* USE RESOURSE *)
54        7     REL<<@;
55        8   END
56        8   ;
57        3 CONNECT USER1.REQ>>SJN.REQUEST;
58        3 CONNECT USER1.OKAK<<SJN.OKACK;
59        3 CONNECT USER1.REL>>SJN.RELEASE;
60        3
61        3 EXPORT Q=SJN.REQUEST;
62        3        O=SJN.OKACK;
63        3        L=SJN.RELEASE
64        3 END.
```

図10 DIPROLによる SJNの記述例

4 DIPROLによる記述例

DP[4] による Shortest Job Next スケジューラ をDIPROLで書き直した例を図10に示す。この例には スケジューラ プロセス(SJN) だけでなく ユーザのプロセス(USER1) も まとめて コンパイル単位を つくってある。

スケジューラ が相手にできる ユーザの数を ポートの ファンアウトに はっきり出した書き方の例でもある。 Sync と block の 対応が SJN.OKACK と USER1.OKACK の間で とられているのも わかる。

5 まとめ

分散型システムの記述言語として、プロセスの相互作用の一様性と、プロセス単位の可搬性を重視した言語 DIPROL について述べた。

プロセスの相互作用を メッセージの通信だけに しぼり、そのための インターフェースとして ポート を設けることにより、プロセス あるいは プロセスの集まり に対し 機能を明確にした。むやみに プロセスを増すことなく

通信の遅延が累積するのを避けるために、通信の実行とプロセスの流れの間の関係に融通性をもたせている。しかし送信が、対応する受信動作と全く独立に働いて、無限のバッファリングを必要とするような形にはせず、ポートごと、1つのメッセージの通信動作(起動～終了)を基本としており、バッファの指定で、これを拡張している。

一方、同期を確実に行うため、プロセスと決して並列に通信をしないという属性(block)と、相手ポートにその性質を期待する属性(sync)をポートの仕様に加えて、ポートの整合性のチェックを強化している。その他、ポートの仕様にpassive/activeの区別を設けた。これは非決定性の選択文の乱用などによる通信系への理不尽な機能分担を抑えるためのもので、ポートの属性と使われ方の組み合せを制限している。

システムの安全なプログラミングのためのもう一つの機能として、アクセスモードをもつ抽象化データ型が導入された。プロセスごとに抽象化データ型への操作能力を設定することができ、柔軟な制限の手段を与えている。

このDIPROLのコンパイラをPascal-P4コンパイラに手を入れて試作した。Pascalで書かれたこのコンパイラは約5000行である。

コンパイルで出力されるプロセスの集りは中間コードであり、それらをより大きな単位にするためのツール他が作製されておらず、DIPROLはまだ実動には至っていない。

〈参考文献リスト〉

[1] Cook, R.P. : *MOD --a language for distributed programming. Proceedings of the 1st Int. Conf. on Distributed Computing Systems, Huntsville, Alabama, (Oct. 1979) 233-241.

[2] 辻野 他 : 分散型システム記述言語 Concurrent-C 電子通信学会技術研究報告 EC-81-13

[3] KESSELS, J.L.W. : The Soma : A Programming Construct for Distributed Processing. IEEE TSE Vol.SE-7, No.5 pp502-509.

[4] Brinch Hansen, P. : Distributed Process: A Concurrent Programming Concept. CACM Vol.21, No.11 pp934-941 Nov.1978

[5] DoD: Reference Manual for the Ada Programming Language. (1980)

[6] F.Tarrini et al : A Network System Language. Proceedings of the 1st Int.Conf. on Distributed Computing Systems, Huntsville, Alabama, (Oct. 1979) 305-314

[7] Feldman, J.A. : High Level Programming for Distributed Computing. CACM Vol.22, No.6 pp353-368 (June 1979)

[8] Jensen, K. and Wirth, N. : PASCAL User Manual and Report, Springer(1974); 2nd ed(1975)

[9] Hoare, C.A.R. : Communicating Sequential Processes CACM Vol.21, No.8, pp666-677 (Aug. 1978)

[10] Mao, T.W. and Yeh, R.T. : COMMUNICATION PORT A LANGUAGE CONCEPT FOR CONCURRENT PROGRAMMING. IEEE TSE Vol.SE-6, No.2 pp194-204 (Mar.1980)

[11] 小森 : "メッセージ通信に基づく分散システム記述言語 DIPROL" 東京大学 情報工学専門課程 修士論文 (Mar. 1982)