な実行のシミュレーションが行える。

2．ソフトウェアシミュレータの概略

（1）Event List

プロセスは、そのプロセスが実行されるべき時刻に応じて Event List につながれ、 Event List の先頭から順次処理される（図1）。

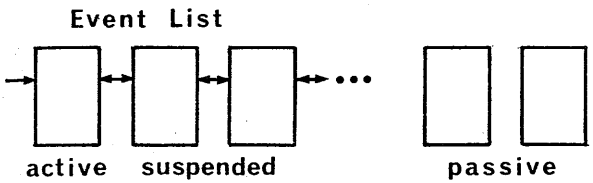

図1 Event List

（2）プロセスコントロール機能

シミュレータに実装されているプロセスコントロール関数の代表的なものを以下に示す。

・Process ＊New (function , stacksize) ;

function を実行するプロセスを生成する。プロセスは stacksize で指定された大きさのスタックを持つ。

・Passivate ( ) ;

プロセスを Event List からはずし passive 状態にする。

・Acitivate (process) ;

passive 状態にあるプロセスを active 状態にする。

・Hold (time) ;

time 時間後にそのプロセスの実行を再開する。プロセスは Event List の time 時間後に相当する位置に挿入される。

（3）シミュレーションモデル

シミュレーションモデルの全体図を図2に示す。UP部はUPとDMに、AC部はACとMMに相当し、UP部とAC部が一組となり Network に接

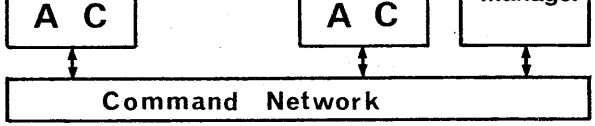

図2 シミュレーションモデルの全体図

3．アクティビティ制御機構

（1）アクティビティ制御機構の役割

アクティビティ制御機構は、並列処理の効率化および論理型言語の拡張機能の実現を目的として、ゴールフレームの選択［2］、関係木情報の管理［1］［3］ を行う。

シミュレータではゴールフレームおよびゴールフレームの導出結果はメッセージという形でUP部、AC部間で送受される。メッセージにより関係木の伸長が行われる。コマンドはAC部間で送受され関係木上の不要なノードや枝の削除等に用いられる。

（2）アクティビティ制御機構のシミュレーション

AC部（AC，MM）のシミュレーションモデルを図3に示す。Message Processor はメッセージを、Command Processor はコマンドを処理する。waiting command list はノードが形成される前にそのノードに到着したコマンドの待機用バッファである。ノードが形成され次第コマンドは Command Processor により処理される。

シミュレータ上のプロセス（図3の＊印）は、2で述べたプロセスコントロール機能を用いて記述されている。プログラム例として Command Processor のプログラムの概略を図4に示す。図4は以下に示す a）～ g）の手順でコマンドを処理する。

a）waiting command list に実行できる待ちコマンドがあれば、それらの処理を行う。

b）command input queue にコマンドが到着していれば command input queue が空になるまで c）

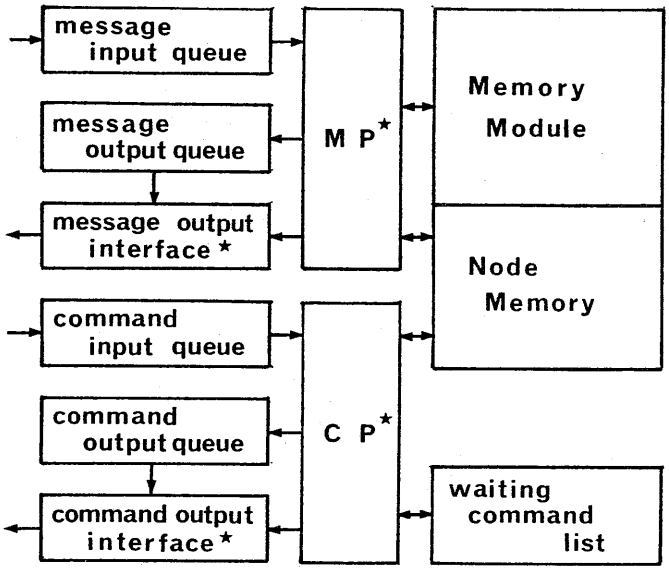

図3　AC部のシミュレーションモデル

～　e）の手順を繰り返す。
c）先頭のコマンドを queue からはずしコマンドに対応するノードがあればそのコマンドを実行する。ノードが形成されていない場合にはコマンドを waiting command list につなぐ。
d）コマンドを実行した結果新たにコマンドが生じたならば、それらを command output queue に入れる。command output interface が passive 状態であったならば active 状態にする。
e）その間に Message Processor によって作られたノードに関する待ちコマンドがあれば、それらのコマンドを処理する。
f）処理できるコマンドがなければ passive 状態となり、コマンドの到着または待ちコマンドを持つノードが生成されたことにより Activate されるのを待つ。
g）a）～f）を繰り返す。

4．おわりに

プロセスコントロール機能をUinx 上のC言語上に実装したことによりプロセスのコンカレントな実行のシミュレーションが容易に行えるようになった。今後、AC部の詳細なシミュレーションを行う予定である。

<参考文献>

［1］　後藤　他“高並列推論エンジンPIEについて”　Logic Prog. Conf. 83　ICOT
［2］　後藤　他：“高並列推論エンジンPIEにおける並列処理の効率化手法について”　信学技報，EC83－9
［3］　丸山　他：“推論向き高並列計算機システムのアクティビティ制御機構”　第26回情処全大，4N－5

```
Command_Processor()
{
    Command *command;

    for(;;) {
      /*
        if ( new nodes are created )
            for all ( created nodes )
                execute_command ( waiting commands );
      */
        while(!Empty(command_input_queue)) {
            out_command(command = First(command_input_queue));
            if(exist_node(command)) {
                execute_command(command);
                Hold(command_execution_time);
                if(commands_are_created()) {
                    into_new_commands(command_output_queue);
                    if(Passivated(Command_output_interface))
                        activate(Command_output_interface);
                }
            } else into_command(waiting_command_list);
          /*
            if ( new nodes are created )
                for all ( created nodes )
                    execute_command ( waiting commands );
          */
        }
        passivate();
    }
}
```

図4　Command Processor のプログラム例