C8-5

多層並列環境でのソフトウェアプロトタイピングについて

Software Prototyping System on the Multi-Layered Environment

松本一教　　内平直志　　本位田真一
Kazunori MATSUMOTO, Naoshi UCHIHIRA, and Sin'ichi HONIDEN

（株）東芝　システム・ソフトウェア技術研究所

Systems and Software Engineering Lab., TOSHIBA corporation

あらまし　複数の並列計算機が多層的に結合された環境下でのソフトウェア作成を容易に行うために，オブジェクト指向の要素を取り入れたMENDEL/GHCとよぶ言語を提案する．この言語により，多層並列環境下でのソフトウェアのプロトタイピングを効率良く行える．また，効率の良い並列実行のためのスケジューリングについても考察する．

はじめに

ソフトウェアの作成において，プロトタイピングの手法を用いることが注目されている[1]．これは，目的とするソフトウェアを作成するに当たり，まずそのプロトタイプを速やかに作成し，それを実際に動作させてみることで，真に目的とするプログラムに速やかに到達しようとするものである．

筆者らは，これまでに，このプロトタイピングを支援する目的のために，MENDELとよぶ言語を提案し，実装してきた[1]．これは，プロトタイピングにおいて重要なプログラムの部品化および再利用が容易に行えるよう設計された言語である．当初のMENDELでは比較的少数のプロセッサからなる並列アーキテクチャを念頭に置いていた．本稿では，複数のプロセッサエレメントから構成されるプロセッサクラスタが，グローバルネットワークを介して複数結合された多層的な並列アーキテクチャ（図１）を対象とした，MENDEL/GHCについて報告する．

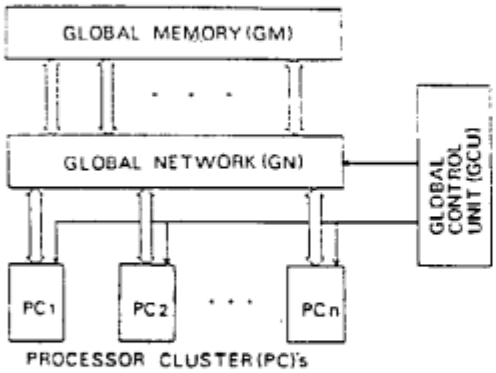

図１　多層並列アーキテクチャ(文献[4]より)

MENDELとは，Actor理論の基本理念を採用した言語である．即ち，プログラムは互いにメッセージを交換するオブジェクトにより構成される．これまでのMENDELでは，このオブジェクトが並列に動作する単位であり，オブジェクトの内部処理は逐次的に実行されていた．これは，先に述べたアーキテクチャ上の想定の他に，より自然に，対象の記述を行うこと，を主たる目的としたからである．本稿で提案するMENDEL/GHCでは新たに，より高速な処理を行う，という目的を達成するために，オブジェクトの内部処理に対しても並列性を導入した．

1．　MENDEL/GHC

MENDEL/GHCでは，オブジェクトの内部処理が並列に行われるので，これまでのMENDEL[1]と差異が生じた．また，現在のインプリメントがFlatGHC[2]による自然な記述を目的としているため，MENDEL/GHCメソッドのガード部には，基本的な組み込み述語しか記述できないという制限もある．

MENDEL/GHCのプログラムは，原子オブジェクトまたは複合オブジェクトである．原子オブジェクトとは，次のようなものである．

```
〈原子オブジェクト〉 ::=
atomic object〈オブジェクト名〉 : (
        dec: ( 〈宣言部〉 ) ;
        method: ( 〈メソッド〉 ) ;
        junk : ( 〈ジャンク〉 ) ;
)

〈宣言部〉 ::=
        〈入力属性宣言〉,
        〈出力属性宣言〉,
        〈内部変数宣言〉

〈入力属性宣言〉 ::=
        input(〈属性〉{,〈属性〉})
```

〈出力属性宣言〉 ::=
output(〈属性〉{,〈属性〉})

〈内部変数宣言〉 ::=
state(〈属性〉!〈初期値〉
{,〈属性〉!〈初期値〉})

〈属性名〉 ::= GHCのアトム
〈初期値〉 ::= GHCの基礎項

〈メソッド〉は，メッセージが到達した場合に実行すべきメソッドの集まりであり，

〈メソッド〉 ::=
{method(〈インタフェース〉) 〈-
GHCのガードゴール列 '|' GHCのボディゴール列 ;}

である．'|'より左側をガード部，右側をボディー部とよぶ．メソッドの実行は，〈インタフェース〉で指定された全ての入力メッセージが到着したメソッドのGHCガードゴール列を試み，それに成功すれば引続きGHCボディゴール列を実行する．
(注)メソッドの選択に，OR並列性は導入しない．即ち，起動条件を満たすメソッドは，先に記述されているものが選択される．ただし，

〈インタフェース〉 ::= {〈属性〉?〈変数〉
| 〈属性〉!〈変数〉}

〈変数〉 ::= GHCの変数

であり，?は入力属性であることを，!は出力属性であることを示す．〈属性〉?〈変数〉は，〈属性〉で指定されるメッセージ入力端子に届いたメッセージを〈変数〉で指定された変数に代入することを意味する．ここでの変数は，GHC部分の変数と共有される．同様に，〈属性〉!〈変数〉はその変数がGHC部分で束縛されている値を，〈属性〉のメッセージ端子から送信することを表す（図２）．なお，メッセージはGHCの基礎項とする．

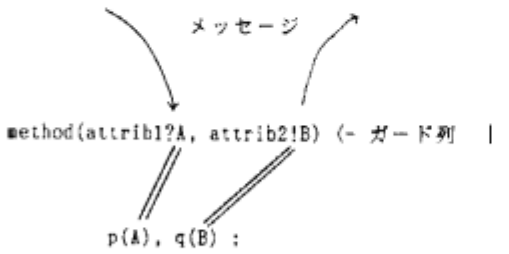

図２ メッセージの送受信

〈ジャンク〉は，メソッドのボディで用いるGHCプログラムを定義する．

〈ジャンク〉 ::= {GHCのプログラム}

(注)MENDEL/GHCでは，原子オブジェクトにメッセージの送受信先を記述することはできない．これは，オブジェクトの汎用性を高めるためである．図３に原子オブジェクトの例を示しておく．

```
atomic object keyCheck(
    declare:( input(word, key),
              output(summary),
              state(wordlist![], flg!off) ),

    method:(word?Word, summary!Result) (- flg == on :
                member(Word, Keywords, Result) ;
    method:(key?Key, wordlist?Old, wordlist![Key!Old]) (-
                flg == off : true ;

    dunk:(  member(X, [X:_], Result) :-
                true : Result = t),
            member(X, [Y:L], Result) :- X =\= Y :
            member(X, L, Result)),
            member(X, [], Result) :- true : Result = nil.)
).
```

図３ 原子オブジェクトの例

複合オブジェクト

複合オブジェクトは，原子オブジェクトが２個以上結合されたものである．ここに，オブジェクトAとBを結合するとは，出力属性で指定されるAのメッセージ送信端子を，入力属性で指定されるBのメッセージ受信端子と一対一に対応付けることをいう．

〈複合オブジェクト〉 ::=
{ connect(〈出力属性〉 of 〈原子オブジェクト名〉,
〈入力属性〉 of 〈原子オブジェクト名〉) }

２． MENDEL/GHCからGHCへの変換

MENDEL/GHCのプログラムは，GHCのプログラムに変換され，実行される．ここではその変換について述べる．並列論理型言語によるオブジェクト指向プログラミングについては，[3]等により提唱されている．ここで述べるオブジェクトの実現は，それを基礎としたものである．とくに，MENDEL/OSとよぶ一種のOSプログラムを新たに導入し，各オブジェクトはMENDEL/OSと交信することでオブジェクト間通信路の生成や消滅を行うものとした．

２．１ MENDEL/OS

MENDEL/OS（図４）は，３引数のGHCプロセスであり，各引数はそれぞれ，

(1) MessageStream : 各オブジェクトからのOSメッセージのストリーム変数
(2) StreamList : オブジェクト間通信用のストリームの管理リスト
(3) IoStream : 各オブジェクトからの入出力の受信ストリーム変数

を表している．

```
'MENDEL/OS'([MessageStream|Stream], StreamList,
                    IOstream) :- true |
    os(MessageStream, Stream, StreamList, IOstream).
'MENDEL/OS'([], _, IOstream) :- true |
    IOstream = [display('* MENDEL/OS halted *')].

os(spawn(ObjectName,Parameter), Stream,
                    StreamList, IOstream) :- true |
    spawn_object(ObjectName, OBJtoOS, Parameter,
                    IOstreamFromObject),
    merge(IOstreamFromObject,
                    IOstreamFromWholeSystem, IOstream),
    merge(OBJtoOS, Stream, MergedStream),
    'MENDEL/OS'(MergedStream, StreamList,
                    IOstreamFromWholeSystem).
os(get_instream(StreamName,Head), Stream, StreamList,
                    IOstream) :-
    atomic(StreamName) |
    get_instream(StreamName, StreamList, Head,
                    NewStreamList),
    'MENDEL/OS'(Stream, NewStreamList, IOstream).
os(get_outstream(StreamName,Tail), Stream, StreamList,
                    IOstream) :-
    atomic(StreamName) |
    get_outstream(StreamName, StreamList, Tail,
                    NewStreamList),
    'MENDEL/OS'(Stream, NewStreamList, IOstream).
os(return_instream(StreamName,Head), Stream, StreamList,
                    IOstream) :-
    atomic(StreamName) |
    return_instream(StreamName, StreamList, Head,
                    NewStreamList),
    'MENDEL/OS'(Stream, NewStreamList, IOstream).
os(return_outstream(StreamName,Tail), Stream, StreamList,
                    IOstream) :-
    atomic(StreamName) |
    return_outstream(StreamName, StreamList, Tail,
                    NewStreamList),
    'MENDEL/OS'(Stream, NewStreamList, IOstream).
```

図4　MENDEL/OSの主要部

MessageStreamを通じて，オブジェクトがMENDEL/OSに送信するメッセージには例えば次のものがある．

spawn　：　オブジェクトを生成し，それを初期化する

get_instream　：　オブジェクト間通信用ストリームの入力端子をOSから借用する

get_outstream　：　オブジェクト間通信用ストリームの出力端子をOSから借用する

return_instream　：　オブジェクト間通信用ストリーム入力端子をOSに返還する

return_outstream　：オブジェクト間通信用ストリーム出力端子をOSに返還する

さて，オブジェクト間の通信についてであるが，オブジェクト間は，固有の名前を持つ通信ストリームにより交信する．通信ストリームの構造については，各通信ストリームは，MENDEL/OS の第２引数StreamListに連想リストとして管理されている．１つの通信ストリームは，Name,Head,Tail)により表現され，Nameはそのストリームの名前を，HeadとTailによりストリームの先頭と末尾を表す．従って，通信ストリームが開設されたときにはHeadとTailは単一化され，ストリームが空であることを示す．開設は，メッセージget_instreamおよびget_outstreamをそれぞれMENDEL/OSに送信することで行う．実際には，get_instreamはストリームの頭部と第２引数を単一化し，get_outstreamはストリームの末尾と第２引数を単一化する(図５)．この通信路の開設のためのメッセージは，複合オブジェクトを変換する際に，自動的に用意される．ただし，IOのためのストリームが複数存在する場合は，io_objectなるオブジェクトが生成され，それらの端子はio_objectと結合される．これは，今回使用したGHC処理系の制約によるものである．なお，いったん通信ストリームが開設されると，オブジェクト間の通信はMENDEL/OSを経由することなく直接的になされる．

```
get_instream(Name, [], LendHead,
            StreamList) :- true |
        StreamList = [(Name,_,LendHead)].
get_instream(Name, [(Name,Head,T)|StreamList],
            LendHead, NewStreamList) :- true |
        NewStreamList = [(Name,_,T)|StreamList],
        LendHead = Head.
get_instream(Name, [(Another,H,T)|StreamList],
            LendHead, NewStreamList) :-
        Name =\= Another |
        NewStreamList = [(Another,H,T)|NextStreamList],
        get_instream(Name, StreamList, LendHead,
            NextStreamList).

return_instream(Name, [], ReturnHead, StreamList) :-
        true |
        StreamList = [(Name,ReturnHead,_)].
return_instream(Name, [(Name,_,T)|StreamList],
            ReturnHead, NewStreamList) :-
        true |
        NewStreamList = [(Name,ReturnHead,T)|StreamList].
return_instream(Name, [(Another,H,T)|StreamList],
            ReturnHead, NewStreamList) :-
        Name =\= Another |
        NewStreamList = [(Another,H,T)|NextStreamList],
        return_instream(Name, StreamList, ReturnHead,
            NextStreamList).
```

図５　ストリームの借用と返還

２．２　オブジェクトの変換

各オブジェクトは，自己再帰によるGHCの無限プロセスとして実現される．それは，MENDEL/OSと通信するための２つのストリームを持つ．１つは，入出力の要求を伝えるためのものであり，他の１つは，MENDEL/OSのコマンドの実行を要求するためのものである．また，spawn_object述語も変換の際同時に生成される．これは，MENDEL/OSのspawnコマンドの実体であり，そのオブジェクトを生成するためのものである．図６に，図３のオブジェクトを変換した場合を示した．

```
spawn_object(keyCheck, ToOS, Parameter,
            IOstream) :- true |
        State = unused,
        ToOS = [get_instream(word,Word),
                get_instream(keyword,Keyword),
                get_outstream(check,Out)|NextToOS],
        keyCheck_object(Word, Keyword, Parameter,
            State, NextToOS, IOstream, Out).

keyCheck_object(InWord, Key, Parameter, State, ToOS,
            IOstream, Out) :-
        InWord = [Word|InWord1] |
        member(Word, Key, Result),
        keyCheck_object(InWord1, Key, Parameter, State,
            ToOS, IOstream, Out1).
keyCheck_object(In, Key, Parameter, State, ToOS, IOstream,
            Out) :-
        In = [] |
        ToOS = [],
        IOstream = [],
        Out = [].

member(X, [X|_], Result) :- true | Result = t.
member(X, [Y|L], Result) :- X =\= Y |
        member(X, L, Result).
member(X, [], Result) :- true | Result = nil.
```

図６　変換されたMENDEL/GHCオブジェクト

３．スケジューリング

本章では，MENDEL/GHCのスケジューリング手法について述べる．ここでのスケジューリングとは，MENDEL/GHCのオブジェクトを多層並列アーキテクチャのプロセッサクラスタ(Pc)に割り当てることである．Pc内部でのスケジューリングについては，例えば(5)の方法の適用を考え

ているが，本稿では言及しない．

MENDEL/GHCでは，スケジューリング規則もプロトタイピングにより生成する．即ち，いくつかのスケジューリングアルゴリズムを用意しておき，それらをユーザの指定により適用してみることでより良いスケジューリングを得ようというものである．ところが，一般の場合にこの問題を解決することは，きわめて難しいことが知られている．そこで，現段階では限定された場合にのみ適用可能なアルゴリズムを用意している．それは，MENDEL/GHCのプログラムからループのないタスクグラフが生成できる場合である．この場合には各オブジェクトの処理時間が与えられていれば，効率の良いスケジューリングを決定できる．次に，その方法の概要を示しておく．詳細については，文献[4,6]を参考にされたい．

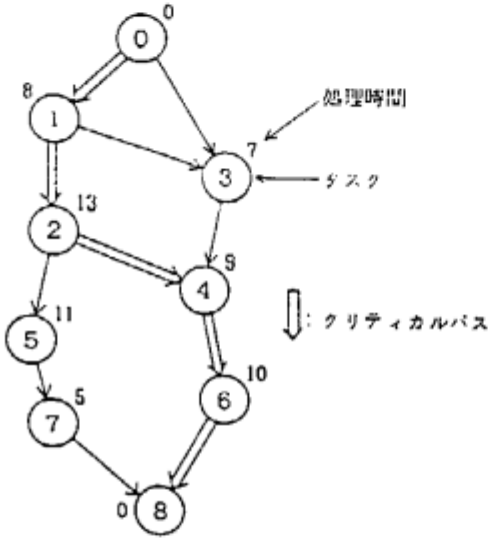

図７　MENDELのタスクグラフの例[4]

(1)　プログラムからタスクグラフ（図７）を生成する．即ち，オブジェクトをタスクグラフにおけるノードと考え，オブジェクトを連結する通信路を先行制約を表すアークに置き換える．ただし，複数の外部との通信路を持つような場合は，処理時間0のダミーオブジェクトを付加することで，外部との通信路を１つにすることができる．（図７で，アークに付けられた数値はそのタスクの処理時間を示す）．

(2)　タスクグラフからCP/MISF法あるいはDF/IHS法によりスケジューリング規則を決定する[4,6]．これらの方法は，スケジューリングの決定に要する時間や，実行時の効率にそれぞれ長短があるので，ユーザがいずれかを指定する．

(3)　決定されたスケジューリング規則を反映して，MENDEL/GHCオブジェクトをGHCに変換する．具体的には，これまでのspawn_objectをプラグマ記述を用いてallocate(Pc)@spawn_objectのようにスケジューリングを反映したものに変更する．ここに，allocate(Pc)とは，@で指定されるプロセスをPcのクラスタで実行することを表すものとする．

おわりに

MENDEL/GHCは，ソフトウェアプロトタイピングの目的に限定して設計された言語である．従って，他の多くのオブジェクト指向言語に見られる，継承機能は含まれていない．これは，プロトタイピングの手法はプロトタイプの作成とその変更（デバグ）という過程を繰り返すが，全てが陽に記述されていた方が，そこでのデバグの効率が向上するとの観点からである．継承機能を欠くためのソフトウェア作成上の効率低下は，各種の知的支援ツールにより，防止できると考えている．実際，そのうちのいくつかについては既に予備実験を終了し，報告を済ませたものもある[7]．即ち，MENDEL/GHCは支援系までを含めて使用されるべき言語である．

現在MENDELSと名づけられたMENDEL/GHCによるソフトウェア作成を支援するプログラミングシステムを構築中であり，そこで得られた結果については機を改めて報告することにする．

謝辞

本研究は，第５世代コンピュータプロジェクトの一環として行われた．研究の機会を与えて下さった方々に感謝致します．また，日頃御指導いただく，中村英夫主任研究員および議論に参加していただいた(株)日本ビジネスオートメーションの藤田俊之氏に謝意を表します．

参考文献

[1]内平　ほか，MENDELにおける並列プログラムの部品合成，情報処理学会ソフトウェア工学研究会，46-8,1986.
[2] K. Ueda, Guarded Horn Clauses, TR-103, ICOT, 1985.
[3] E. Shapiro et al, Object Oriented Programming in Concurrent Prolog, New Generation Computing, Vol. 1, No. 1.
[4] 甲斐　,並列処理環境における動的タスクスケジューリングに関する研究，早稲田大学学位論文，1988.
[5] M. Sato et al, KL1 Execution Model for PIM Cluster with Shared Memory, Proc. of the 4th ICLP, 1988.
[6] H. Kasahara et al, Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, IEEE Trans. Comput. C-33, 11.
[7]　伊藤　ほか，MENDELにおける意味ネットを用いた部品結合，第２回AI学会全国大会，1988.