能動的オブジェクトを用いたモデル構築による問題解決

刈 谷 丈 治*

Problem Solving by Modeling with Active Objects

Joji KARIYA

Abstract

To solve a problem without effective algorithm, we must construct a model of the objects, and combine the partial facts, and try to solve it by using heuristics. While a system co-operates with man to solve such problems, it is necessary to behave as man for mutual comprehension between man and the system.

In order to write a program which allows natural description and simulates the process of man's thinking, we extend REWMOL to have object-oriented programming facility. REWMOL is a LISP like language with concurrent process facility.

Problems of cryptomathematics are solved to test on the ability. It is made clear that it have a good property to generate, execute and put together many subgoals concurrently, but it is some difficult to descride discontinuation of thinking.

1．はじめに

プログラム作成支援等のアルゴリズムの無い問題について，人間に対し知的な援助をするシステムを作ろうとすると，人間とシステムとの相互理解が重要となる．そのために，システムは人間の思考過程に近い方法をとらねばならず，対象のモデル化が必要である．忠実なモデル化によりセマンティックギャップをなくし，多くの実験により対象を深く理解した上で，必要なら効率を考えればよい．

筆者は，実世界をモデル化するときの基本要素である対象・作用・時間を実現するために，並列LISPの一種である言語REWMOL（REal World MOdeling Language）を開発した．その後，対象の記述を容易にかつ強化するためにオブジェクトを導入した．人間等の能動的存在を，内部に永久に動作を続けるプロセスを多数含むオブジェクトによりモデル化することが出来る．並列に動作する多数のオブジェクト間のメッセージ交換がモデル要素間の作用である．

オブジェクトはプロセス，プロセデュアであると同時にデータでもあり，各種の探針の埋込みが容易なので，対象の動作と並列にメタレベルの処理を行い，干渉させることもできる．これは単にモデル構築によるシミュレーションが出来るだけでなく，システムに自己構築の能力を与えうることを示す．従って能動的オブジェクトによるモデル化技法は，人間に知的援助を行うシステムを作成するためのかなりの能力を有するといえる．

SMALLTALK[2]で導入されたオブジェクト概念は，抽象データタイプ[3]の発展したものであり，LISP上でも重要な機能となっている．[4)-7)] しかし，多重継承，並列処理，複雑な構造物の扱い等，研究・実験を積みかさねる必要がある．本論文では，2章でオブジェクト機能のとらえかたをSMALLTALKと対比して述べ，3章でオブジェクトで記述した虫喰算の例を示し，並列処理に現れる困難な点に触れる．4章では結論を述べ，他のシステムとの比較をして今後の課題を示す．

2．オブジェクト機能

2.1 オブジェクトのデータ構造

SMALLTALKはシミュレーション用言語SIMULAから発達した言語であり，オブジェクトは実世界の事物を表現するのに適した性質を持つ．SMALLTALKでは基本要素が全てオブジェクトであるのに対し，

*山口大学情報処理センター

REWMOLでは，オブジェクトはデータタイプの一種であり，アプリケーション上の構成要素を表すものに用い，プログラム上のデータは他のタイプを用いて表現することにより，階層性を持たせることができる．

```
class-object── variable-list
  ├── receiver
  ├── method-list
  ├── superclass-list
  class

instance-object ─ variable-list
  ├── receiver
  class
```

Fig. 1 STRUCTURE OF OBJECT

クラスとインスタンスのデータ構造を図1に示す．共にアトムであり，右方向がCDR下方向がCAR部分である．variable-listはオブジェクトに固有の情報を保持し実行時の環境となる．receiverはメソッド名を含んだ全てのパラメータを受け取り，実行する関数を決定する．method-listはキャッシュしたメソッドも含んでいる．superclass-listは上位クラスのリストでメソッドサーチの順序を決定する．

本システムは実験システムなので，このように簡単な構造とし，変更が容易であるようにした．REWMOLの基本部分に付加した機能は，オブジェクトを独立なタイプとしたこと，メッセージ受信において送信者を得る関数を作成したことである．

2.2 実行方式

オブジェクトへのメッセージの送信は

（<- object METHOD argument ...）

とする．METHOD以外は評価される．標準のreceiverを用いると，メソッドキャッシュを内蔵した機構により，objectのクラスでメソッドの定義を検索して関数を得，objectの変数リストを連想リストとする環境の下で関数を実行し，関数値をメッセージの値とする．

受信する側からいえば，メソッドの関数定義はラムダ式で行い，自オブジェクトの変数とメソッドの引数をローカル変数とするプログラムを書けば良い．

メソッド内でスーパークラスのメソッドを使用するときは，

（<<superclass METHOD argument ...）

とする．superclass内でメソッドが検索され，自オブジェクトselfの環境でメソッドが実行される．selfは自動的に定義されるオブジェクトの変数で，自オブジェクトを値として持つ．

このように，メソッドはオブジェクトの持つ変数を環境とするクロージャのように実行される．しかし，プロセスとして見た場合，あくまで同一のプロセスとして実行されるので，オブジェクトの機能を独立のプロセスとして実行したいならば，メソッドの定義関数を，例えば

```
（＊LE (arguvar-s)〈＊LEは　LAMBDAと同目的〉
  (WAKE (FORK (localvar-s)
        accept arguments
        process body
        ...
```

とする．メッセージが単なる関数か，それともプロセスであるかは，利用者のモデルによる．

プロセスを内蔵するオブジェクトは能動的オブジェクトであり，モデル化で特に重要なものである．

2.3 インスタンスとクラス

インスタンスは識別可能な個々の事物を表し，その性質は全てインスタンスの層するクラスに規定されるものと考える．性質は，インスタンスの量的側面を表す要素と，質的側面を表す反応によって定まり，具体的には変数とメソッドからなる．変数の保持するものには，即値と，部分であるオブジェクトと，他のオブジェクトの参照がある．

クラスもインスタンスとしてあるクラス（メタクラスという）に属する．クラスとメタクラスの関係も全く同じで，メタクラスはクラスを規定する．インスタンス―クラスの関係は相対的なものであるが，クラスでないオブジェクトをインスタンスといい，それを基準としてクラス，メタクラスと言うこともある．

SMALLTALKの場合，クラスとメタクラスは一対一に対応し，それぞれのスーパーの関係が平行していなければならないとなっている．これはインスタンスの初期化がメタクラスの役割となっているためである．しかし，インスタンスが生成されてから，活動するのに必要な情報を得るまでには，かなりの幅があり，メタクラスの一アクションですまないものもある．相互参照をしているインスタンスでは特にそうである．すなわち，インスタンスの初期化をインスタンスのライフの一過程ととらえるなら，クラスに記述されるべきものである．メタクラスをクラスの束縛から外し，独立の役割を担わせるべきである．なおflavorではメタクラスを定義できない．

REWMOLの標準のメタクラスCLASS は，クラスに定義された変数の初期化を行うので，パラメータのない初期化であれば通常メタクラスを定義する必要

はない．クラスがインスタンスを持つかどうか，インスタンス生成時に何を行うかはクラスの性質であるから，メタクラスに記述してよく，特殊化した初期化を要するなら，クラスごとにメタクラスを定義してもよい．リソース管理のためのモニタ[8]を定義したところの階層構造を図2に示す．

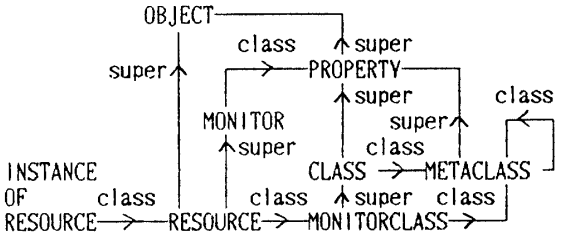

Fig. 2　OBJECTS HIERARCHY

MONITORCLASSはRESOURCEで与えられたメソッドを，排他制御を組込んだ形に変換して定義し，MONITORをスーパークラスとして付加することにより，排他制御用の変数が自動的に作成されるようにし，かつ，WAIT，NOTIFY，SIGNALという機能を提供する．

RESOURCEが定義されると，入り口のセマフォーキュー，モニタ内実行待ちキュー，条件毎のキューCONDnが作られる．制御用関数と出入口での機能は，

SIGNAL：CONDnの先頭のプロセスに実行権を渡し，実行待ちとなる．

WAIT　：実行待ちプロセスがあれば実行権を渡し，なければ入り口を開け，CONDnで待つ．

NOTIFY：CONDnの先頭のプロセスを実行待ちとする．

入口　：許可されるまでセマフォーのキューで待つ．

出口　：実行待ちプロセスがあれば実行権を渡し，なければ，入り口を開ける．

である．こうしてモニター内では複数のプロセスが存在でき，かつ実行権の受け渡しにより協調性の排他制御が行われる．モニター内のプロセスが死ぬとデッドロックになるので，殺されないようにしている．

2.4　プログラミングシステムとの類比

オブジェクトで構成されたシステムをプログラミングシステムと類比してみる．インスタンスはプログラムの実行時に現れるものであり，実行状態を保持し，実行を進める主体でもある．実行はクラスに定義されたメソッドにより規定されるので，クラスはプログラムに相当する．メタクラスはクラスの生成，すなわちプログラムの作成を規定するのであるから，言語，ツール，仕様といったものに対応する．既成のクラスは実行時ルーチン，ライブラリに相当する．このように考えると，かなり広い範囲のものが，ひとつの枠の中で定義できる．多くの有用なメタクラス，スーパークラスが作成されれば，プログラミング環境が整ったと言えるであろう．

2.5　多重継承

REWMOLでは多重継承を提供している．継承するものは，変数の初期化およびメソッドである．多重継承で問題となるのは多重定義と思われる．自由度を失わないように，かつシステムがはんさとならないようにするために，クラス定義の終結を明示的に宣言させ，定義の終結したクラスのみがスーパークラスとなりえることとした．クラスで有効なメソッドの定義は，SMALLTALK-80[9]と同じで

①そのクラスで定義されている．

②真上のスーパークラスのどれかで有効であり，かつ複数のスーパークラスで有効なら，その定義が同一のクラスから継承したものである．

このチェックをスーパークラスの定義時に行うことにより，スーパークラスの定義順によらずツリートラバーサルにより正しいメソッドが得られ，実行時のメソッドキャッシュで得られるメソッドが定義時に確定する保証を与えている．定義終結後も存在するメソッドの修正はできる．追加も出来るが，その結果の再現性について，現時点では責任を持たない．

従って，メソッドの再定義により同一のクラスを特殊化して得た別々のクラスを，同時にスーパークラスとすることはできないことになる．異なる特殊化は，矛盾する相異なるクラスを定義することであるから当然である．これに対し，同一のクラスに異なるメソッドを付加して特殊化した別々のクラスを同時にスーパークラスとすることができるのはもちろんである．

また，新スーパークラスの導入に伴って導入されるメソッドが偶然既知のメソッドと同名となることも，検出される．

3．虫食算による実験例

3.1　問題のモデル化

虫食算の問題を例にとって，記述のしやすさ，人間の思考法との比較を行う．虫食算とは，図3のような計算式中の，不明な数を決定する問題である．人間のとる方法の一つは，既知の数がまとまっている付近から計算を始め，未知数の絞り込みを行う．それが行き

詰まると，周囲への影響が最も大きそうな数を選んで場合分けをし再び計算を進める．行き詰まりの判断や，場合分けの選択，計算順序等，あいまいな所，逆に言えば，ヒューリスティクスの入る所が多くある．

問題をモデル化してみると，まず実体として，問題の式全体，それを構成する，各数，計算があり，積や和の所では，構成上まとまった単位がある．活動要素として，問題を解く，場合分けされた全体の計算を進める，個々の計算を行うがある．図4のように単位を定めると，図3の問題は図5のようなパイプラインあるいはネットワークのように表現でき，各計算が並列に局所的に可能な数を求め，変化があれば数を媒介に関連する計算を起動することにより，制約条件を伝播させ，全体の計算を進めることができる．

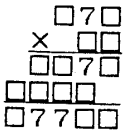

Fig. 3 A PROBLEM

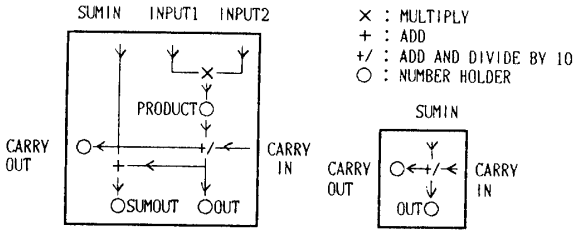

Fig. 4 PRODUCT UNIT AND SUMUNIT

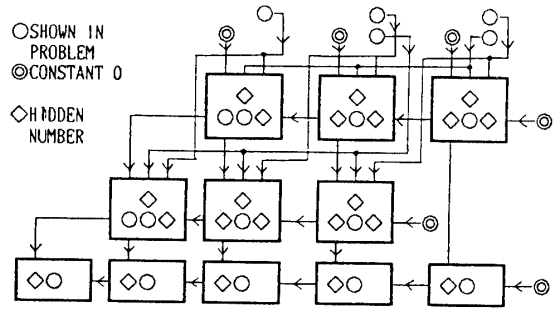

Fig. 5 CONFIGULATION OF CASE

プログラム全体は約600行にもなるので，各クラスの説明と，クラス間のメッセージ，中心となる実行管理について解説する．

クラス PROBLEM-SOLVER：問題を受けてCASE-SOLVERを生成し，解を探索させる．行き詰まりになったら場合分けをし，各CASE-SOLVERを生成する．解が見つかったら，別解を探すかどうかを問い合わせ，探さないなら残っているCASE-SOLVERを停止させる．

クラス CASE-SOLVER：与えられた数字の組み合わせから，NUMBER-HOLDER, COMPUTATIONのネットワークを構成し，可能な数を制限する計算を起動する．計算終了を判断して，解有り，解無し，行き詰まりのいずれかを PROBEM-SOLVER に報告する．

クラス PRODUCT-UNIT：図4のPRODUCT UNITを生成，複写するときのテンプレートとして使用されるだけで，プログラム作成の簡易化の役目だけをする．

クラス SUM-UNIT：図4のSUM UNITのテンプレート．

クラス NUMBER-HOLDER：可能な数のリストを持ち，変化すると関係する計算を起動する．

クラス COMPUTATION：下記の各種の計算に共通する制御用のメソッドの定義を持つ．

クラス MULTIPLY：2数の積を計算する．

クラス ADD ：2数の和を計算する．

クラス ADD-SPLIT：和を10位と1位に分ける．

以下に重要なメソッドの説明を

受信クラス<— メソッド（送信クラス） 機能

という形式で記述する．メッセージは送信者を限定しないのだが，ここではプログラム上，主たる通信関係を記述するために，送信クラスを書く．

PROBLEM-SOLVER<-RESULT
(CASE-SOLVER)

解有り，解無し，行き詰まりを報告する．

CASE-SOLVER<-PROBLEM
(PROBLEM-SOLVER)

問題を与え実行のできる構造を生成させる．

CASE-SOLVER<-START (PROBLEM-SOLVER)

計算の実行を開始させる．

CASE-SOLVER<-STOP (PROBLEM-SOLVER)

計算を停止させる．

CASE-SOLVER<-COPY (PROBLEM-SOLVER)

CASE-SOLVERのコピーを作らせる．

CASE-SOLVER<-CANDIDATE
(PROBLEM-SOLVER)

場合分けを行うのに適したNUMBER-HOLDERを選択させる．

CASE-SOLVER<-CHANGE
(PROBLEM-SOLVER)

指定した位置のNUMBER-HOLDERの値を変更する．

CASE-SOLVER<--NO_SOLUTION
(COMPUTATION)

解無しの報告．

COMPUTATION<--RESTART
(PROBLEM-SOLVER, NUMBER-HOLDER) 計算を開始，再開させる

COMPUTATION<--STOP (PROBLEM-SOLVER)

計算を停止させる．

COMPUTATION<--TERMINATE
(COMPUTATION)

自分自身を停止状態とする．

COMPUTATION<--ACTIVE(PROBLEM-SOLVER)

計算中，停止の状態問い合わせ．

COMPUTATION<--SPAWN(COMPUTATION)

計算プロセスの生成．(計算の種類により異なる関数が実行される)

NUMBER-HOLDER<--CHANGED_BY
(COMPUTATION)

可能な数の変更を通知する．

このほかに各オブジェクトを構成するために初期値やオブジェクト間関係を設定するメソッドがあり，その構造から通信先を知って通信が進行する．

3.2 問題解決上の特徴

この問題には二つの並列性があり，その取り扱いが焦点となる．第一は，場合分けをしたそれぞれのケースが並列に実行されることである．この場合，実行上での干渉は殆ど無いので，問題は複雑な状態をどのようにしてコピーするかである．モデルによりアイデンティティが異なり，従ってコピーの内容が異なる．すなわちコピーされるものが構成要素としてのオブジェクトであるか，外のオブジェクトの参照であるか，外のオブジェクトもコピーされ参照の対応関係が保存されるのか等によりコピーの方法が変わってくる．このような詳細な知識をひとつにまとめてコピーするプログラムを作ると，モジュラリティは悪く信頼性の全く無いものができる．

構成部品間に関係がある時は，その関係付けはインスタンスの生成時にも行うので，コピーのメソッドを構成要素をコピーするステップと構成要素の関係付けをするステップに分解し，コピーされる各オブジェクトについて定義する．こうするとコピーの具体的方法は，必要な知識の存在する各オブジェクトの定義，すなわちモデルの中に記述されるので，容易に信頼性が得られる．

第二の問題は，各ケースの計算の終了（解ありまたは行き詰まり）をどのようにして判定するかである．各ケースに含まれる多数のCOMPUTATIONは同時に動作しており，終了したり，NUMBER-HOLDERを介して他のCOMPUTATIONを中止させ更起動したりする．今回採った簡単な方法は，CASE-SOLVERがCOMPUTATIONの状態を一度に調べる方法である．

別の通常使われる方法は，動作中COMPUTATION数を管理し，COMPUTATIONの終了時，開始時に更新し，零になることを検査する方法である．この方法では，COMPUTATIONの状態の変化と，動作中COMUTATION数の更新の対応を完全にするために，この処理を，中断させられずに行わねばならない．動作中COMPUTATION数はCASE-SOLVERの属性であるので，この更新をメッセージで行うと，メッセージの実現法によっては，処理が中断されないという保証はなく，上位オブジェクトの状態を，下位オブジェクトが直接更新するという，望ましくないことを行わなければならない．

待行列を持つセマフォーを用いてCOMPUTATIONのRESTART, TERMINATEメソッドの中で排他制御を行う方法だと，Vオペレーションを行う前に停止させられると，セマフォーの実現法によってはデッドロックになる．このように，更新を伴う同期処理は，プロセスが停止させられる可能性のある時は危険であり，一般に行うべきでない．多プロセスが主体的に動作する問題では，サブジェクト[10]概念に基づく，制限された相互作用が適当と考えられる．

3.3 実験結果

パズル誌[11]から得た3桁×2桁から3桁×4桁の問題16題を解いてみた．桁数によらないプログラムが作成されたのもモジュラリティの良さを示す．9題は問題分割無しで．6問は1回分割を行い，平均3つの問題を生成して解決した．1問は7回の分割で55の問題を生成した．これからこのように局所的な計算でも十分な能力があることが分かった．困難であった問題でのプロセス数の時間変化を図6に示す．ピーク時には約30の問題と 100を越えるプロセスが生成された．小さな問題でも数多くのプロセスが発生消滅を繰り返し，多プロセスの管理能力の必要性を示している．このような問題を用いると制御法の改善，ヒューリスティクスの開発等を行う事ができる．実行中のトレースの一部を図7に示す．

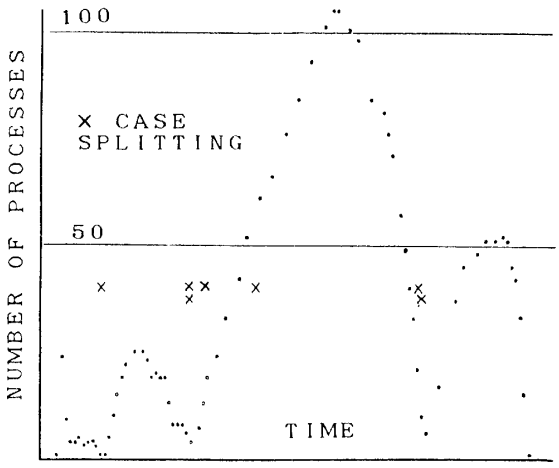

Fig. 6 TIME vs. NUMBER OF PROCESSES

```
(G31882670 / ADD-SPLIT) -> (G31882666 / NUMBER-HOLDER) (CHANGED_BY G31882670 (2 9 6
(G31882666 / NUMBER-HOLDER) -> (G31882671 / ADD) (RESTART)
(G31882666 / NUMBER-HOLDER) <- (G31882671 / ADD) G31883499
(G31882670 / ADD-SPLIT) <- (G31882666 / NUMBER-HOLDER) NIL
(G31882695 / ADD-SPLIT) -> (G31882698 / NUMBER-HOLDER) (CHANGED_BY G31882695 (0))
(G31882695 / ADD-SPLIT) <- (G31882698 / NUMBER-HOLDER) NIL
(G31882695 / ADD-SPLIT) -> (G31882676 / NUMBER-HOLDER) (CHANGED_BY G31882695 (11))
(G31882695 / ADD-SPLIT) <- (G31882676 / NUMBER-HOLDER) NIL
(G31882665 / NUMBER-HOLDER) -> (G31882670 / ADD-SPLIT) (RESTART)
(G31882665 / NUMBER-HOLDER) <- (G31882670 / ADD-SPLIT) G31883500
(G31882669 / MULTIPLY) <- (G31882665 / NUMBER-HOLDER) NIL
(G31882669 / MULTIPLY) -> (G31882669 / MULTIPLY) (TERMINATE)
(G31882669 / MULTIPLY) <- (G31882669 / MULTIPLY) NIL
(G31882850 / ADD) -> (G31882815 / NUMBER-HOLDER) (CHANGED_BY G31882850 (7))
(G31882850 / ADD) <- (G31882815 / NUMBER-HOLDER) NIL
(G31882850 / ADD) -> (G31882845 / NUMBER-HOLDER) (CHANGED_BY G31882850 (3))
(G31882850 / ADD) <- (G31882845 / NUMBER-HOLDER) NIL
(G31882850 / ADD) -> (G31882847 / NUMBER-HOLDER) (CHANGED_BY G31882850 (10))
(G31882849 / ADD-SPLIT) <- (G31882845 / NUMBER-HOLDER) NIL
(G31882849 / ADD-SPLIT) -> (G31882849 / ADD-SPLIT) (TERMINATE)
(G31882849 / ADD-SPLIT) <- (G31882849 / ADD-SPLIT) NIL
(G31883024 / ADD-SPLIT) -> (G31883019 / NUMBER-HOLDER) (CHANGED_BY G31883024 (72))
(G31883024 / ADD-SPLIT) <- (G31883019 / NUMBER-HOLDER) NIL
(G31883024 / ADD-SPLIT) -> (G31883021 / NUMBER-HOLDER) (CHANGED_BY G31883024 (7))
(G31883021 / NUMBER-HOLDER) -> (G31883016 / ADD-SPLIT) (RESTART)
(G31883011 / NUMBER-HOLDER) -> (G31883016 / ADD-SPLIT) (RESTART)
(G31883011 / NUMBER-HOLDER) <- (G31883016 / ADD-SPLIT) G31883502
(G31883015 / MULTIPLY) <- (G31883011 / NUMBER-HOLDER) NIL
(G31883015 / MULTIPLY) -> (G31883015 / MULTIPLY) (TERMINATE)
(G31883015 / MULTIPLY) <- (G31883015 / MULTIPLY) NIL
```

Fig. 7 TRACE OF THE EXECUTION

4．結　論

並列処理機能を持つREWMOLの上にオブジェクト機能を実現し，複雑な並列性を有する問題の記述，実行を行うことができた．オブジェクト指向は，対象物は明確だが関係が複雑であるような場合に適した問題解決の方法であり，身近にある問題をモデルの作成から始める時には[12]利用されていくと思われる．その際例えば多数の能動的対象が存在するOSの実験[13]のように，独立主体間の相互作用が重要である問題をモデル化して実験を行うには，REWMOLのように，能動的オブジェクトが作成でき，並列処理を陽に制御できるベース言語が望ましい．

次に他のLISP上のオブジェクト機能との比較を行い，今後の課題を述べる．

LOOPSは複合オブジェクトの定義のために，TEMPLATEというメタクラスを持ち，部分構造の少し異なるオブジェクトの定義を容易にしている．スーパー関係に基づく性質記述の方法に比べ，部分構造の記述及び部分構造に基づいて発生するメソッドの類型化は遅れているので，この種のメタクラスの開発は重要である．またプログラミング環境としてのメタクラスの整備も重要である．

flavorのデモンはスーパー関係に分散したメソッドの統合を行う．これはメソッドの定義が単なる独立した手続きの付加でなく，潜在した性質の活性化を行うことも意味する．すなわちクラス定義の変化に伴い自己組織化する機能が重要になると思われる．

TAOはLISP, PROLOG, SMALLTALKを統合したLISPマシンELIS上の言語である．[14] メッセージの識別子としてパターンを使用できる[15]点に特徴がある．REWMOLでは個々のオブジェクトがreceiver を持つので，メッセージ授受の方式を複数持つことは勿論，オブジェクトごとに方式を変える実験も行うことができる．本論ではメッセージを単に作用としたが，さらに分類と意味付けをしなければならない．

現在REWMOLは山口大学情報処理センターのACOS850上で一般利用者に提供されており，山口大学短期大学部のVAX730のVMSの下に移植され，一部OS依存機能を除いて稼働している．REWMOLは殆ど言語Cで書いてあるので，容易に移植が完了した．

附　録　REWMOLの概要

1．LISPとしてのREWMOL

REWMOLは，拡張性の高いLISPを基本とするが，並列実行を特に重要と考えるため，LISPのインタプリタに並列処理機構を埋込んだものとなっている．REWMOLのLISPとしての能力はアトムと関数の種類で推定されよう．現在アトムの種類は，シンボル，文字列，生成アトム，整数（文字），実数，ファイル，READTABLE，プロセス，オブジェクト，関数である．

関数には次の種類がある．マクロ，コンビネータ，
引数評価・環境引継型，引数非評価・環境引継型
引数評価・環境保持型，引数非評価・環境保持型

約180の基本関数を含む約260の関数が提供されている．[16] 数値関数などの広い範囲の努力はしていないが，殆ど言語Cで書いてあるので拡張は容易である．中断してTSSの直接の制御下に移り，エディタ等の作業をした後，REWMOLの実行を継続することもできるので実用上十分な能力となっている．

2．並列処理機能

実行可能なプロセスがサイクリックにタイムスロットだけ実行し交代する．タイムスロットはEVALの実行回数である．実行可能なプロセスが存在しないとき，システムはプロセスモードから基本モードへ替わり，LISPのトップレベルとなる．各プロセスには活動，休止，死の状態があり，自由に制御できる．次に並列処理の基本的な関数を列記する．

(＊SPAWN FUNCTION ARGUMENTS) APPLYと同様だが，実行するかわりにプロセスを返す．

(FORK (VAR-S) FORM FORM-S) ブロック構造の一種だが実行するかわりにプロセスを返す．最初のFORMはプロセス間のインターフェイスをとるために，親プロセスとして実行される．

(WAKE PROCESS)　プロセスを活動状態とする．

(SLEEP PROCESS)　プロセスを休止させる．

(SLEEP)　自プロセスが休止する．

(KILL PROCESS)　プロミスを殺す．

(DIE)　自プロミスが死ぬ．

(SUSPEND)　次のディスパッチを待つ．

(CRITICALSEQ FORM-S)　ブロック構造の一種で実行している間にディスパッチされることがない．

(PROTECT FORM-S) ブロック構造の一種で実行している間に殺されることがない．

(MYSELF) 自プロセスが得られる.
(PROCEED) 基本モードからプロセスモードへ移る.

以上の機能を用いて，各種の排他制御やメッセージ交換を作成している．このとき，プロセスがアトムであり，属性リストを持つことを有効に利用している．

参考文献

1) 刈谷丈治，石原好宏：実世界モデル記述言語REWMOLの開発，電信論，Vol. J 68-D, 2, PP114-121 (1985)
2) Goldverg, A. : SMALLTALK-80 : THE INTERACTIVE PROGRAMMING ENVIRONMENT, Addison-Wesley
3) Wirth, N. : THE MODULE : A SYSTEM STRUCTURING FACILITY IN HIGH-LEVEL PROGRAMMING LANGUAGE, Lecture Notes in Computer Science 79, PP1-24, Springer-Verlag
4) Weinreb, D., Moon, D. : Flavors : Message Passing in the Lisp Machine, A. I. Memo No. 602 (1981)
5) Cannon, H. I. : Flavors : A non-hierarchical approach to object-oriented programming
6) Burke, G. S., Carrette, G. J., Eliot, C. R. : NIL Notes for Release 0.259 (1983)
7) Bobrow, D. G., Stefik, M. : The LOOPS Manual, Memo KB-VLSI-81-13, XEROX Corporation
8) Andrews, G. R., Schneider, F. B 訳中川，松原.：並列型プログラミングにおける概念と表記法，コンピュータ・サイエンス，共立
9) Borning, A.H., Ingals, D.H.H.: Multiple inheritance in smalltalk-80, AAAI-82, PP234-237
10) 久野功，諏訪基：協調問題解決機構を目指したサブジェクトに基づく知識表現，情報処理学会知識工学と人口知能研究会資料84-36
11) パズル通信ニコリ，㈱ニコリ
12) 安西裕一朗，近藤公久，中村久肇，浦昭二：オブジェクト指向型表現を用いた物理問題解決システム，情報処理学会知識工学と人口知能研究会85-38
13) 田胡和哉，益田隆司：オペレーティング・システムの構造記述に関する一試み，情報処理学会論文誌，Vol. 25, No. 4, PP.524-534
14) 大里延康，竹内郁雄，奥乃博：TAOにおける代人計算機構，情報処理学会記号処理研究会85-31
15) 奥乃博，大里延康，竹内郁雄：Lispにおけるオブジェクト指向型プログラミング，情報処理学会記号処理研究会83-26
16) 刈谷丈治：REWMOL (V3.6)取扱説明書 (1986)

(昭和61年4月15日受理)