ソフトウェア基礎論　23－4
（1987. 12. 10）

並列協調型システム*Harmonia*の構成と機能

Organization and Function of Parallel Coordinated Computing System *Harmonia*

尾内 理紀夫　鶴岡 行雄
Rikio Onai　Yukio Tsuruoka

NTT ソフトウェア研究所
NTT Software Laboratories

あらまし　並列動作する多数の分散化疎結合コンピュータが互いに協調して、coherentな問題解決を図る並列協調型問題解決システムHarmoniaの論理モデル、基本構成、そして通信機能をはじめとする各種機能について述べる。普通・緊急という二レベルのしかもハードウェアを意識する必要のない通信機能と、豊富な知識漸増・質問応答機能により、Harmoniaでは多種多彩な並列協調型問題解決が可能となっている。なお、Harmoniaの実験システム第１版は、Ethernetで結合された4台のAIワークステーションELISからなり、各種機能は複合パラダイム言語TAOによりインプリメントされている。

Abstract　A parallel coordinated computing system Harmonia is composed of loosely-coupled computers which solve problems coherently and harmoniously in parallel. This paper describes the model, basic organization, communication and gradually increasing knowledge of Harmonia. It is possible in Harmonia to attempt various kinds of parallel coordinated problem solving techniques using two levels of communication (ordinary and emergent) which can be used independently of hardware configuration, and many kinds of messages which gradually increase knowledge.

Experimental hardware of Harmonia is composed of four AI work-station ELISs connected by Ethernet and implementing various Harmonia functions using the multiple paradigm language TAO.

１．はじめに

コンピュータの高速化・小型化・低廉化、ネットワークの発達、エキスパートシステムの多様化を背景にその重要度を増している研究の一つに、1970年代後半に開始された分散型人工知能の研究がある（この分野の研究は1980年には第一回のワークショップが開催され、1986年までで六回を数えるに至っている）[Davi 80][Davis 82][Fehl 83][Smit 85][Gass 87]。

分散型人工知能は、互いに協力する分散化された複数ＡＩに関する研究であり、分散、並列、ＡＩといった広い領域にまたがっている。なぜ並列・分散化された複数ＡＩに関する研究が重要かと言えば、将来、ＡＩが十分に"intelligent"であるためには、システムは極めて複雑な多量知識を保持・管理しなければならなく、これは並列・分散化してはじめて可能となるからである。また、人工知能の問題の中にはその性質・特徴からみて、空間的分散あるいは機能的分散が自然な場合が多いからである。

並列協調型問題解決の研究は、分散型人工知能研究の一部分をなし、比較的粒度の大きいサブ問題への機能的分割を特徴とし、歴史的には、Contract Net[Smit 78], Scientific Comunity Metaphore[Korn 81], Hearsay-II [Erma 80]等の研究がある。

並列協調型問題解決システムHarmoniaは、並列動作する多数の分散化疎結合コンピュータから構成され、それらコンピュータ内のエージェントが互いに協調して、coherentな問題解決を図る。

分散されるのは制御とデータの両方であり、個々のコンピュータは自律的に並列動作する。

疎結合という意味は、個々のコンピュータが大部分の時間を通信よりは計算に消費するということである。

また、協調と協力とは意味が異なる。協力とは、同一目標あるいは互いに衝突しない目標をもつもの同志が助け合うことであり、協調とは、時には計画・行動が衝突し、その際には妥協する必要があるということである。すなわち、"協調＝協力＋妥協"、である。

coherentな問題解決とは、多数のコンピュータのうちのいくつかが一時的にゴールと反対方向に動作することはあっても、コンピュータ群全体としてはゴールに向かって動作するということである。

本稿では、並列協調型問題解決システムHarmoniaの論理モデル、基本構成、そして通信機能をはじめとする各種機能について述べる。

なお、Harmonia実験システム第１版は、Ethernetで結合された4台のAIワークステーションELIS[Yama 86][Wata 87]

[Tani 87]からなり、各種機能は複合パラダイム言語TAO [Take 83][Take 86][Take 88]によりインプリメントされている。

2．モデル

Harmoniaが採用しているsolver間の通信を許す動的manager-solver階層モデルについて述べる。

前述したように、Hearsay-IIをはじめとしたいくつかの並列協調型問題解決モデルが提案されてきた［Film 84]。しかし、黒板モデルといわれる、共有メモリを前提とする方式以外にはハードウェアを意識したシステムは少ない。

黒板モデルは、黒板への書き込み、読み出しを実行する存在が、数個から十数個の分野には適しているかもしれないが、それ以上の個数の分野には、黒板へのアクセスが衝突するため不向きとなってしまう。

Harmoniaの目ざすところは、極めて多数の存在が有機的に結合され、通信し合い、全体として処理が円滑に進んでいくシステムを構築することであるから、黒板による密結合モデルではなく、ネットワークによる疎結合モデルを採用する。

Harmoniaでは、並列動作可能な通信機能を持つ能動的存在をIntelligent Agent(以後略してIA)と呼ぶ。 このIAはある時には、managerとなりその配下にあるIA(solvers)に対して問題を付与しその解決進行を管理するが、ある時にはsolverとなり他のIA(manager) から送られてきた問題をできるかぎりローカルな知識で解決しようとする。すなわち、IAはその役割及び関係を動的に変更することができる。

一例を図１に示す。ただし図１は単にある局面でのmanagerとsolverの関係を示すだけであり、manager-solver関係の動的逆転も可能である。

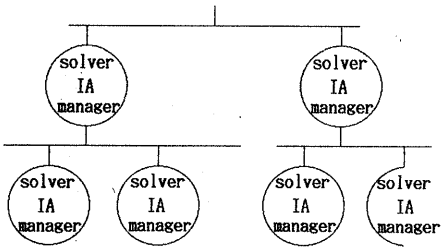

図1 solver間の通信を許す動的manager-solver階層モデル

また、図１からもわかるように、Harmoniaではmanager-solver間の通信だけでなく、solver間(between solvers)の通信も許す。この理由について次に述べる。

並列協調型システムでの問題解決過程は大きく次の三段階からなる[Davi 80]。

第一段階　問題をサブ問題に分割・分散する。
第二段階　サブ問題を解く。
第三段階　サブ問題の解を統合し、最終解を得る。

特に重要なのは第二段階におけるサブ問題間の結合度(degree of coupling)がどの程度かということである。

結合度がゼロならば、サブ問題は独立に解くことができる。しかし、通常はゼロではない。

結合度がゼロでなければ、各コンピュータはサブ問題の解決の途中状況を通信し合うことによりサブ問題のあいまいさを減じたり、サブ問題解の統合により無矛盾な最終解を得るために部分的な解を交換しなければならない。

また、サブ問題が独立に解くことができる時でさえも、通信による協力は有効である。つまり、サブ問題間通信により最終解をより速く求めることが可能となる。このように、１台のコンピュータによって発見された情報が、他のコンピュータにより利用されることによる全体としての問題解決のスピードアップ効果は、探索空間が広い領域では顕著である。

以上の理由から、Harmoniaでは、manager-solver間だけでなく、solver間の通信も許している。

第一段階においては、機能的にあるいは空間的に分割可能な問題を複数IAへ分割・分散させる。機能的にも空間的にも一つの問題を分割するような意味のないことをしないことはもちろんである。これは、並列性のないプログラムを並列マシンで処理しても意味がないことと同じである。ただ、問題解決過程において、manager-solver間あるいはsolver間の通信量を極力抑制するような、即ち、solverができるかぎりローカルな知識でサブ問題解決をできるような方式の開発は本研究の目的の一つである。

3．構成

物理的構成を決定する際に考慮した技術的背景と、ネットワークとコンピュータと人間からなるHarmoniaの物理的構成について述べる。

3.1 技術的背景

近年の素子技術の発展はここで改めて述べるまでもなくきわめて急速である。この結果、マイクロコンピュータはより高機能に安価に、メモリもより高集積化され安価になってきたし、これからもその傾向は続く。このため、この世の多くのものに、高機能なマイクロコンピュータとある程度の容量のメモリとネットワークインターフェースを安価に具備させることが可能になってきた。即ち、飛行機や船だけでなく、自動車にもエレーベータにも、そしてロボットにも通信機能付きのマイクロコンピュータが搭載されることがあたり前の時代がくるだろう。

コンピュータ・ハードの発展も急速であるが、グローバル・ネットワーク、ローカルネットワークの発展はそれにまさるとも劣らない。その結果、近い将来、世界中がネットワークにより連結され、工場や家庭のoutletは、単なる電源供給口ではなく、情報の出入口になっていくだろう。すなわち、outletおよび各種電子機器のplugの形状は例えば図２のようになり、１チップ・コンピュータあるいは１ボード・コンピュータ組込みの数多くの機器がこの口を通して、極端に言えば全世界と通信できるようになる。（すでにその第一歩として、日本にはHA規格の情報コンセントがある。）

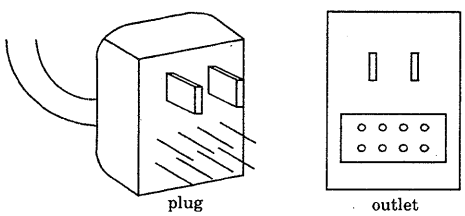

図2　plug と outlet

しかしハード的には通信可能になるにしても、多数の機器が全体として、致命的な衝突なしに並列協調動作していくためのソフトウェアが開発できなければ宝のもちぐされである。

並列協調型問題解決システムHarmoniaは、そのようなソフトウェア開発技術の確立を目指したものであり、その構成を決めるにあたっては、近未来に可能になる全世界規模のコンピュータとネットワークという技術的背景を意識し、考慮に入れた。

3.2 ネットワーク

並列協調型問題解決システムHarmoniaは前述したように全世界を対象にしているため、ネットワークは解決すべき問題にtuneした特殊なものではなく、ISOであれde factであれ、実質的世界標準に基本的に合わせる。

基本的という意味は、Harmoniaによる実験結果に基づき、より適切なネットワークへの改善あるいは提案を行っていくことはあるということである。

このことと、疎結合(個々のコンピュータは大部分の時間を通信より計算に消費する)という基本的枠組から、Harmonia実験システム第一版のネットワークは、最も数多く使用されているローカルエリア・ネットワークであるEthernetを採用した(図3)。実験システム内での通信に関する統計情報収集のためswitchにより、実験システムをisolateできるようになっている。

将来的には、グローバル・ネットワーク接続へとHarmoniaを拡張していく予定である(図4)。

3.3 コンピュータ

実験システムでは、各種IAが存在するコンピュータをＡＩワークステーションELISとする(図3　各ELISには複数台の端末が接続可能)。

現時点において、ELISは、自動車やロボットに搭載するには大きすぎるが、ハードウェア技術の進歩を考えれば、ごく近い将来には自動車はおろか、電子レンジや洗濯機に組込むこともハードウェア的には可能になるだろう。

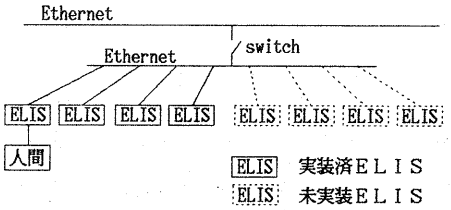

図3　実験システム

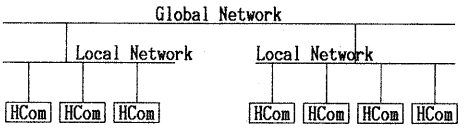

図4　将来のHarmoniaシステム

コンピュータとしてELISを採用したのは、強力な記述力を持つ複合パラダイム言語TAOがインプリメントされ、またマイクロコード化による高速化が可能だからである。よって、Harmoniaの基本機能は複合パラダイム言語TAOとマイクロプログラムにより実現される。ちなみに、IAはTAOプロセス(システム定義クラスprocessのインスタンス)をベースにしている。

TAOは、Lisp、Prolog、オブジェクト指向機能等を調和平均的に融合させたプログラミング言語である。調和平均的という意味は、単にLisp、Prolog、オブジェクト指向機能をとり入れたというのではなく、Lisp関数中で論理変数が使用できたり、逆にPrologのリテラルとしてLisp関数が使用できたり、メソッドを実行中のオブジェクトに対してthrowによって割込みをかけることができるなど、各種言語機能が有機的に結合しているということである。

なお、将来Harmoniaでは、コンピュータはELISだけでなく、Harmonia機能を具備したコンピュータへと拡張していく(図4)。

3.4 人間

人間はきわめて有能なIAと考えることができ、並列協調型問題解決過程で人間をmanagerIAあるいはsolverIAとしたい局面が出現する。そこで、Harmoniaは、ELIS端末をIAに指定し、人間がディスプレイ等の出力を受けて、キーボード等の入力装置により応答することを可能にしている。すなわち、Harmoniaはコンピュータと人間との相補的な調和をも目指す。

4. 機能

Harmoniaの基本機能である通信機能、知識漸増・質問応答機能等について述べる。

4.1 通信機能

本節ではHarmoniaのIA間通信機能とそれによるIA間の既存通信手法(Now型/Past型/Future型通信、Occam同期式通信)の実現を示す。

基本通信機能として、Harmoniaはメイルボックスによる普通通信と、プロセス間割込みによる緊急通信という二レベルのIA間通信を提供する。　この二つを組み合わせることにより、大部分の既存通信法は実現できる。

またIA間通信には、異なるELIS間のIA間通信と、同一ELIS内のIA間通信がある(図6、7)。　そして、IA間通信を行う時、それぞれのIAがどのELISに存在するかという、IAとELISとの対応は通信サーバがめんどうをみる。

このHarmonia通信サーバの用いる通信機構はTCP(Transmission Control Protocol)[Post 81][Mura 87]の上位階層として実現されている(図5)。

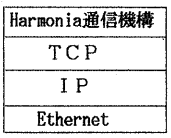

図5 通信階層

TCPは、時間切れと再転送を基本とした高信頼性プロトコルであり、多くのコンピュータにおいて高レベル・プロトコルの下位階層としてインプリメントされているが故に、Harmonia通信機構実現の出発点として選択した。

よって、今後の実験結果を踏まえ、通信機構の最適化のため、IP等のより下位階層プロトコルにより通信機構を再実現することもある。

それでは、二つの通信、普通通信と緊急通信について説明する。

4.1.1 普通通信

まず、TAOのメイルボックス通信機構について簡単に説明する。mailboxによる通信とは、荒っぽく言えば、普通の家庭にある郵便箱による通信と同じである。

mailboxはクラスであり、インスタンス変数としては次のものがある。

・sys:mailbox-process-queue

メイル（郵便物）を待っているプロセス（たとえて言えば、郵便箱に新聞をとりに行ったが、まだきていないので待っている人)は、このsys:mailbox-process-queueという待ち行列につながれる。

・sys:mail-queue

プロセスにより受理されるのを待っているメイル（たとえて言えば郵便箱の中に入れられた手紙)は、このsys:mail-queueという待ち行列につながれる。

主な関数としては次のものがある。

・(receive-mail メイルボックス)

receive-mailにより、メイルボックスに送られてくるメイルを取り出す。receive-mailは該当メイルボックスにメイルが送られてくるまで値は返ってこない。すなわち、受信プロセスはメイルを受け取るまで待ち続ける。

・(send-mail メイルボックス　メイル)

メイルボックスに対しメイルを送る。

mailboxオブジェクトは、いわば、生産者プロセスと消費者プロセスの間の無限長バッファである。生産者プロセスは、send-mailにより sys:mail-queueにメイルを送り込み、消費者プロセスは、receive-mailによりそこからメイルを取り出す。送り込みと取り出しは非同期であり、生産者プロセス、消費者プロセスの数は１つでも複数でもよい。よって１プロセス対１プロセス、多プロセス対１プロセス、１プロセス対多プロセス、多プロセス対多プロセスの非同期かつ非決定的通信がこの mailboxオブジェクトにより実現できる。

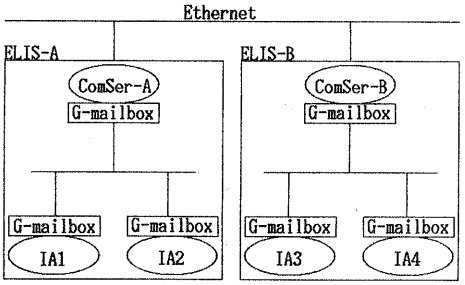

図6　普通通信

send-mailとreceive-mailを用いた１台のELIS内プロセス間各種メイルボックス通信は［Onai 86-2］［Onai 86-3］に述べた。

ここでは、異なるELIS間にわたる場合も含めた普通通信について述べる。

ｼﾝﾀｸｽ：(send-message IAs ﾒｯｾｰｼﾞ)
　　(receive-message)

IAsは、make-IAによって生成されたIAプロセス・リスト。IA1個の時はアトムも可。

これらのIAに順次ﾒｯｾｰｼﾞが送られる。よって複数IAを指定すれば一種の同報通信ができる。

各IAには、汎用メイルボックス(G-mailbox)が一つ付随している。receive-messageはこのG-mailboxからﾒｯｾｰｼﾞを取り出す。なお、これ以外にメイルボックスが必要ならば、

　(make-instance 'mailbox ~)

により生成することができる。

さて、(send-message IA ﾒｯｾｰｼﾞ) は通信サーバ(これもIA) に送られる。 通信サーバは、自ELIS内IAとG-mailboxの対応表、他ELIS内IAとELIS名との対応表、IA名とその実体との対応表を保持、管理している。

(場合1) 相手IAが、send元IAと同一ELIS内にある場合
通信サーバは、自ELIS内IAとG-mailboxの対応表から相手G-mailboxをもとめ、
(send-mail G-mailbox ﾒｯｾｰｼﾞ)
を実行する。

(場合2) 相手IAが異なるELISにある場合
通信サーバは他ELIS内IAとELIS名との対応表から相手ELISを求め、相手ELISの通信サーバに
(mail IAs ﾒｯｾｰｼﾞ)
を送る。
相手サーバは、自ELIS内IAとG-mailboxの対応表から送り先IAのG-mailboxを求め、
(send-mail G-mailbox ﾒｯｾｰｼﾞ)
を実行する。

そのほかの各種普通通信について説明する。

・(receive-personal-message　送信IA)

メッセージの送信IAを指定する。 receive-messageの場合は、自G-mailbox内にメッセージがあればそれを取り出してしまうが、 receive-personal-messageは、指定した送信IAからのメッセージを取り出す。

・(get-back-message 相手IA ﾒｯｾｰｼﾞ)

これはいったん出したメッセージを取り返すためのものである。値はnil（すでにメイルボックスからメイルが取り出されてしまった)あるいは、メッセージ(get-back成功)である。

・(receive-message-with-timeout n)

これは、メイルボックスよりメッセージを取り出す際に、n/60秒間しか待たない。そして、その間にメッセージが送られてこなければnilが値である。

・(send-message-with-timeout n 相手IA ﾒｯｾｰｼﾞ)
これは、ﾒｯｾｰｼﾞは出すが、n/60秒以内に相手G-mailboxからとり出されなければそのﾒｯｾｰｼﾞを送信IAに送り返す。よって値は、ﾒｯｾｰｼﾞか t (相手IAによって受信された)である。

・(send-round-message IA ﾒｯｾｰｼﾞid ﾒｯｾｰｼﾞ)
(receive-round-message ﾒｯｾｰｼﾞid)
これらは往復ﾒｯｾｰｼﾞを表す。単に相手からのﾒｯｾｰｼﾞを待つのでなく、自分の出したあのﾒｯｾｰｼﾞの返事を待つという場合に使用する。
ﾒｯｾｰｼﾞidとしては、発信IAのidと時刻を組にする。

・(sync-send-message IA ﾒｯｾｰｼﾞ)
これは、同期送信である。相手IAから確認信号が来たときに値 t を返す。

4.1.2 普通通信による既存通信の実現

本節では、前述した通信機能による既存の通信法について述べる。

(1) Occam同期式通信
C.A.R.Hoareにより提案された CSP (Communicating Sequential Processes)[Hoar 85]をベースにした並行プロセス記述言語がOccam[Onai 86-1]である。 このOccam同期式通信がHarmonia普通通信により実現できることを例を用いて示す。

2つのIAがある。生産者IAは 値iniから始まる整数を次々にチャネル経由で出力し、消費者IAはチャネルを通して受け取った整数を出力装置に出力するものとする。同期式だから、生産者IAがチャネル経由で１つ整数を送り、消費者IAがそれを受け取ってから生産者IAは次の整数を生成する。

Harmoniaプログラムを次に示す。

```
(defun generate (ini IA)
 (loop (&aux n)
       (:init (!n ini))  ;内部変数nに初期値iniを設定
       (:until (= n 101) t)    ;nが101になったら終了
       (sync-send-message IA n)
                ;IAへnを同期送信
                ;相手IAより確認信号を受理したら次へ
       (!n (+ n 1))))           ;nの値を更新

(defun receive (IA)
  (loop (&aux i)
        (!i (receive-message))
                        ;ﾒｲﾙﾎﾞｯｸｽ inから値を取る
        (send-message IA "ack")
                        ;相手IAへ確認信号ackを送出
        (write i)               ;出力装置へ出力
        (:until (= i 100) t)))

(!generate-IA (make-IA 'generate-process)
;これによりgenerate-processという名前のIAが生成され、
;それへのポインタが変数generate-IAにassignされる。
(!receive-IA  (make-IA 'receive-process)
```

```
(defun synchro-para-start ( )
  (start-IA generate-IA
              #'generate (list 1 receive-IA))
  ;generate-IAﾌﾟﾛｾｽをﾌｫｰｸﾌﾟﾛｾｽとして生成し、走行させる。
  ;1,receive-IAはそれぞれ実引数
  (start-IA receive-IA
              #'receive (list generate-IA)))
```

本プログラムは(synchro-para-start)により起動される。

(2) ABCL通信
ABCL(An Object-Based Concurrent Language)[Yone 85][Suzu 85]は、並列・分散処理をはじめから考慮に入れたオブジェクト指向言語であり、Actorモデルをそのベースとしている。ここでは、ABCLの通信・同期機能の大きな特徴である3種類のメッセージ伝達モード(Past型、Now型、Future型)をHarmonia普通通信を用いて記述する。

・Past型
2つのIA間でsend-messageとreceive-messageを実行することにより実現できる。

・Now型
これはOccam同期式通信と同じであり、
(sync-send-message 相手IA ﾒｯｾｰｼﾞ)
とすればよい。

・Future型
Future型メイルを送ったIAは、相手IAからの返事をまたずに自分の仕事を続け、相手からの返事が返ってきた段階でその結果をある変数に代入する。もし返事が到着する以前に、この変数を参照する必要が生じたなら、メイルを送ったIAはそこで返事待ちとなる。このようなFuture型は

```
(send-message 相手IA ﾒｯｾｰｼﾞ)
        :
        :
(receive-message)
```

あるいは、

```
(send-round-mail 相手IA ﾒｯｾｰｼﾞid  ﾒｯｾｰｼﾞ)
        :
        :
(receive-round-mail ﾒｯｾｰｼﾞid)
```

により実現できる。

4.1.3 緊急通信

緊急通信はいわば、電報あるいは電話に類する通信である。 緊急通信は、単に相手IAのメイルボックスにﾒｯｾｰｼﾞを送るのではなく、相手IAに割込みをかけ、ﾒｯｾｰｼﾞが来たことを知らせる。
まずTAOのプロセス割込みについて簡単に説明する。
プロセス割込み process-interruptのシンタクスは、
(process-interrupt ﾌﾟﾛｾｽ fn args flag)
である。

プロセスは割り込みたい相手プロセスである。 flagが tならば、相手プロセスが待ち状態の時でも割込み、argsにfnをapplyする。flagが nilならばプロセスが走行中に fnが実行され、fn がプロセスの実行状態を破壊しないかぎり、fnの実行終了後、元の状態に戻って計算を続ける。

process-interruptが実行されると、相手の環境で fnが実行される。 よって、args内にcatch tagを指定することによって、プロセス間throwを実現できる。

また、TAO特有の機能としてthrow時の多岐への分岐が可能なcatcher-caseがある。

catcher-caseのシンタクスは

```
(catcher-case form (tag1 form11 form12 ・・・)
                          :
                   (tagi formi1 formi2 ・・・)
                          :
                   (tagN formN1 formN2 ・・・))
```

である。 form中で、例えば (list tagi value)がthrowされれば、formi1以降に制御が移る。formi1以降ではtagiは変数として束縛される。tagiの初期値はvalueである。tagNをtにしておけば、throwのcatch tagがtag1 ~ tagN-1に一致しない場合、制御はformN1以降に移る。

もう一つの割込みに関するTAO特有機能throwablepについて説明する。

throwablepのシンタクスは (throwablep tag) である。一致するtagがあれば t、なければ nil を返す。

たとえば、一台ELIS内の緊急通信は、process-interruptとcatch&throwを用いて次のように実現できる[Omai 86-3]。

```
work-process0
    :
  (process-interrupt work-process1
    #'throw (list 'telegram メッセージ) nil)
    :

work-process1
   (!telegram-message
     (catch 'telegram
        :
     ))
```

それでは異なるELISにわたる場合を含む緊急通信について述べる(図7 ComSerは通信サーバ) 。

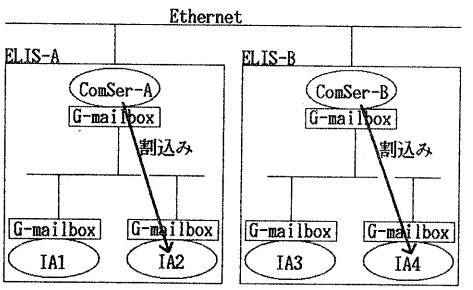

図7 緊急通信

シンタクス : (send-e-message IA メッセージ)

eは、expressあるいはemergencyのeである。これはいったん自ELIS内通信サーバに送られる。 通信サーバは、G-mailboxからメッセージを取り出すとき、sys:mail-queueをみて、もしe-messageがあればそれを先に取り出す。この緊急通信は通信サーバからのreturn値を待つ、即ち、同期送信である。

(場合1)相手IAがsend元IAと同一ELIS内にある場合

通信サーバはthrowablepを実行し、値がtならば

(process-interrupt IA #'throw メッセージ flag)

を実行することにより、相手IAに割込みをかけ、メッセージをわたす(メッセージのcar部にcatch tagがある)。そしてsend-e-messageの値をtとする。

throwablepの値がnilならば、send-e-messageの値をnilとする。

なお、flagがtならば相手IAが待ち状態の時でも割込み、nilならば、相手IAが走行中に割込む。

(場合2) 相手IAが異なるELIS内にある場合

通信サーバは他ELIS内IAとELIS名との対応表から相手ELISを求め、相手ELISの通信サーバに (e-mail IA メッセージ)を送る。

異ELISの通信サーバはthrowablepを実行し、値がtならば

(process-interrupt IA #'throw メッセージ flag)

を実行することにより相手IAに割込みをかけ、メッセージをわたす。そして送信元IAへ通信サーバ経由でsend-e-messageの値としてtを送る。

throwablepの値がnilならば、送信元IAへ通信サーバ経由でsend-e-messageの値としてnilを送る。

(緊急通信例)

managerが複数のsolver(たとえばsolver1とsovler2)にサブ問題を与え、各solverはそれぞれのlocalな知識を用いて、問題解決処理を実行しているとする。問題解決過程のある時点においてmanagergが、solver1に処理の中止命令を緊急通信により指示するという局面を考えてみる。

managerは、

(send-e-message solver1 (list 'stop stop-reason))

を実行する。

受け側のsolver1のプログラムは例えば次のようである。

```
(catcher-case サブ問題解決処理
    (stop   後処理)
    (suspend 情報待避処理)
    (deadlock   資源解放処理))
;たとえば
;suspendは、solver1による問題解決の一時中断。
;deadlockは、deadlockに陥ったことの知らせ。
```

solver1がサブ問題解決処理中に割り込まれたならば、すなわち、throwablepがtならば、catch tagがstopに一致するので、stop-reasonの値に基づき後処理が実行される。

throwablepがnilならば、send-e-messageの値としてnilが返り、managerは緊急通信が失敗したことを知る。

このmanagerとsolverとの緊急通信において、managerもsolverも互いが同一ELIS内にいるのか、異なるELISにいるのかということを意識することなしに通信していることに注意してほしい。

4.2 知識漸増・質問応答機能

人間が人間に"何か"を教えるとき、その"何か"についての全ての知識を与える訳ではない。教えられる側の人間が必要とする範囲の知識を与える。同様に 並列協調型問題解決システムにおいても、managerはsolverが問題解決するために必要な知識を与えればよい。もしmanagerが、より高度な解決をはかりたくなった時はそのために必要な、より高度な知識をsolverに与えればよい。solverは、より高度な知識を、以前に与えられた知識に融合させ、即ち、以前の知識を動的に変更し、より高度な問題解決をはかる。すなわち、知識漸増・質問応答機能は、並列協調型システム実現のための重要な要素機能の一つである。

我々は階層性とモジュール性に着目し、Harmoniaにおける知識表現の出発点としてオブジェクト指向を選択した。よって、managerはsolverに対して、知識としてのクラス定義、メソッド定義そしてインスタンス定義を与える。いったん与えたクラス定義等に変更を加える必要が将来にわたってないならば問題は起こらない。しかし、知識が変化するのは当然のことであり、最初に完全なクラス定義等を与えることは不可能である。よってmanagerがsolver内の各種定義を動的に変更することが必要となる。しかし、オブジェクトの各種定義は静的であり（例外はCLOS,これについては後述）defclass,make-instance実行後、クラスに変更を加えたくなっても、たんにクラス定義を再定義しただけでは、その下のインスタンス・オブジェクトは古いクラス定義に依ったままである。新たなクラス定義に基づくためには、ユーザーが再度make-instance等をしなければならない。そこで、Harmoniaはオブジェトクトの動的変更機構[Onai 87-1]を具備している。

本節では、この動的変更機構によるオブジェクト(動的オブジェクト)を用いた柔軟な知識漸増と質問応答について述べる[Onai 87-2][Tsur 87]。

4.2.1知識漸増

manager-solver間あるいはsolver同志のオブジェクト指向型知識の漸増(付与と変更)と質問応答は4.1節に述べたIA間メッセージ通信により行われる。よって、互いに相手の知識を直接見ることはできない。これは、human-metaphorで言えば、他人が何を考えているかを脳の中を直接見ることにより知ることができないのと同じである。

以下に知識漸増のための各種"ﾒｯｾｰｼﾞ"について述べる。"ﾒｯｾｰｼﾞ"は4.1節のﾒｯｾｰｼﾞに相当する。すなわち、以下の各種ﾒｯｾｰｼﾞは普通通信あるいは緊急通信により他IAに送ることができる。

(1) クラス生成と変更

ﾒｯｾｰｼﾞ:(d-defclass ｸﾗｽ c変数 i変数 ｽｰﾊﾟｰ ｵﾌﾟｼｮﾝ)

d- は動的という意味であり、d-defcalssは関数名である。ｸﾗｽはクラス名、c変数はクラス変数、i変数はインスタンス変数、ｽｰﾊﾟｰはスーパークラスである。

これはクラスの生成と変更のために使用される。

追加/変更時には、追加/変更対象以外の引数はnilにする。この形式は主として二つ以上の引数の変更のために使用される。

また、一引数の追加/変更のためにはクラス定義変更の簡略形がある。つまり、知識の差分のみの記述が可能である。 以下それについて述べる。

・クラス変数変更

ﾒｯｾｰｼﾞ：(d-class-var ｸﾗｽ c変数)

4種類のc変数形式がある。

(:delete c変数)	c変数の削除
(c変数 値)	新c変数とその値の付加
c変数	新c変数とその値nilの付加
(:set c変数 新値)	c変数の旧値が新値により置換

例えばsolver1のあるｸﾗｽに初期値0のクラス変数countを普通通信により新たに定義したければ、
(send-message solver1 (d-class-var ｸﾗｽ名 (count 0))
あるクラス変数を削除したければ、ﾒｯｾｰｼﾞ部分を
(d-class-var ｸﾗｽ名 (:delete ｸﾗｽ変数))
とする。

・インスタンス変数変更

ﾒｯｾｰｼﾞ : (d-instance-var クラス名 i変数)

5種類のi変数形式がある。

(:delete i変数)	i変数の削除
(i変数 値)	新i変数とその値の付加
i変数	新i変数とその値nilの付加
(:set-all i変数 値)	i変数の旧値が新値により置換
(:set-if-undef i変数 値)	i変数の旧値が未定義値ならば、新値が設定

たとえば、あるインスタンス変数を削除したければ、
(d-instance-var クラス名 (:delete i変数))
とする。

・スーパークラス変更

ﾒｯｾｰｼﾞ : (d-hierarchy ｸﾗｽ ｽｰﾊﾟｰ)

2種類のｽｰﾊﾟｰ形式がある。

スーパークラス	スーパークラス追加
(:delete スーパークラス)	スーパークラス削除

・オプション変更

ﾒｯｾｰｼﾞ: (d-option ｸﾗｽ ｵﾌﾟｼｮﾝ)

2種類のｵﾌﾟｼｮﾝ形式がある。

オプション	オプション追加
(:delete オプション)	オプション削除

(2) メソッド生成と変更

ﾒｯｾｰｼﾞ:(d-defmethod ｸﾗｽ-ｾﾚｸﾀ対 メソッド本体)

同じクラスに属する他のメソッド定義は、 セレクタが異なれば、影響は受けない。セレクタが衝突した時、つまりセレクタが同じであった時には古いメソッド定義は新しいメソッド定義により置換される。

インスタンス・オブジェクトの内部表現内にはメソッドに関する情報は格納されてない。よってメソッド定義の追加、変更によりインスタンス・オブジェクトは影響を受けない。

(3) インスタンス・オブジェクト生成と変更

・インスタンス生成

ﾒｯｾｰｼﾞ : (instance ｸﾗｽ ｲﾝｽﾀﾝｽ i変数)

たとえば、managerがsolverに、pochiはクラスdogのインスタンスだと教える時は
(send-message solver '(instance dog pochi))
と送る。

このメッセージを受け取ったIAは、その内部で
(d-make-instance クラス インスタンス)
を実行し、クラスに所属するインスタンス・オブジェクトを生成する。

・インスタンス変数値の変更
メッセージ:(instance クラス インスタンス i変数)

i変数形式を以下に示す。
(:set i変数 値)　　i変数の旧値が新値により置換
(:set-if-undef i変数 値)　i変数の旧値が未定義値ならば、新値が設定
・所属クラスの変更
メッセージ：(instance (:revise クラス) インスタンス i変数)

これは、たとえば、pochiはクラスdogのインスタンスではなくて、クラスcatのインスタンスだということにあたる。この時、managerは、solverに
(instance (:revise cat) pochi)
と送る。
当然、solverにはcatのクラス定義はすでにある。なければ、そのクラス定義たとえば、
(d-defclass cat (•••)(•••)(•••))
を送っておく。
solverでは、
(d-make-instance cat pochi)
が実行され、これにより、以前のインスタンス・オブジェクトpochiは消去される。

4.2.2. 質問応答
質問応答のためのメッセージについて述べる。
通信には知識漸増と同じく普通通信と緊急通信が可能である。
以下に述べる質問により他のIA内の知識を知ることができる。よって、これら質問は、Harmonia組込みの一種のメタ・メソッドと考えることができる。

(1)クラスに関する質問と応答
・質問メッセージ：(question-class クラス キー)

キー	意味
なし	そのクラスが存在するか?
c-var	全クラス変数とその値は？
(c-var クラス変数)	そのクラス変数とその値は？
i-var	全インスタンス変数とその初期値は？
(i-var インスタンス変数)	そのインスタンス変数と初期値は？
instance	全インスタンスは？
(instance インスタンス)	そのインスタンスは存在するか?
super	全スーパークラスは？
(super スーパークラス)	そのスーパークラスは存在するか?
option	全オプションは？
(option オプション)	そのオプションは存在するか?

・応答メッセージ：(answer-class クラス 回答)

回答	意味
:not-exist	そのクラスはない
:exist	そのクラスは存在する
(クラス変数 nil :not-exist)	そのクラス変数はない
((クラス変数 値) ……)	クラス変数と値
((インスタンス変数 値) ……)	インスタンス変数と値
(インスタンス変数 nil :not-exist)	そのインスタンス変数はない
:instance-not-exist	インスタンスはない
(インスタンス :not-exist)	そのインスタンスはない
(インスタンス :exist)	そのインスタンスは存在する
インスタンス	存在するインスタンスのリスト
:super-not-exist	スーパークラスはない
(スーパー :not-exist)	そのスーパークラスはない
(スーパー :exist)	そのスーパークラスは存在する
スーパークラス	存在するスーパークラスのリスト
:option-not-exist	オプションはない
(オプション :not-exist)	そのオプションはない
(オプション :exist)	そのオプションは存在する
オプション	存在するオプションのリスト

(2) インスタンスに関する質問と応答
・質問メッセージ：(question-instance インスタンス キー)

キー	意味
なし	そのインスタンスが存在するか? 存在するならそのクラス名は？
(i-var インスタンス変数名)	そのインスタンス変数と値は？
i-var	全インスタンス変数とその値は？

・応答メッセージ：(answer-instance インスタンス 回答)

回答	意味
:not-exist	そのインスタンスはない
クラス名	そのインスタンスは存在し、クラスに所属する
(インスタンス変数 nil :not-exist)	そのインスタンス変数はない
(インスタンス変数 値)	そのインスタンス変数と値
((インスタンス変数 値) ……)	インスタンス変数と値

(3)メソッドに関する質問と応答
・質問メッセージ：(question-method クラス キー)

キー	意味
なし	そのクラスに付随するメソッドは?
メソッド名	そのメソッド名のメソッドは?

・応答メッセージ：(answer-method 回答)

回答	意味
メソッド名とメソッド本体	そのクラスに付随するメソッド
nil	メソッドはない
(メソッド名 :not-exist)	そのメソッドは存在しない

ただし、今のところ、コンピュータがメソッド本体を見て、その処理を理解することはできない。よって、メソッドに関する質問応答は、managerが人間である場合にのみ有効である。

4.2.3 質問・応答による知識漸増例
質問応答による各種知識漸増例を示す。
以下においては、managerからsolverへのメッセージ送信を m⇒s 、solverからmanagerへのメッセージ送信を s⇒m とする。このメッセージも普通通信あるいは緊急通信により送受される。
なお、他IAに与えたクラス、インスタンスは、そのIAで記憶しておく。

(1) クラスとインスタンスの漸増例

```
m⇒s (question-class dog)  ;クラスdogは存在するか?
s⇒m (answer-class dog :not-exist)      ;存在しない
m⇒s (d-defclass dog ()((eye 2))())     ;ｸﾗｽdog付加
    (instance dog pochi)
                ;eyeが2つのdogのｲﾝｽﾀﾝｽ pochiの生成
```

(2)インスタンス変数の漸増例

ここでは、dogには足もあり、普通は4本であることを追加する。

```
m⇒s (question-class dog (i-var leg))
s⇒m (answer-class dog (leg nil :not-exist))
m⇒s (d-instance-var dog (leg 4))
```

この時、(instance dog pochi)を再度送る必要はない。なぜなら、動的変更機構がインスタンス・オブジェクトpochiを変更するからである。なお、このdogは4.3節で述べる固有知識であり、managerがdogに変更を加えても、メッセージ送信をしないかぎり、solverのdogには影響は及ばない。

(3) インスタンス・オブジェクト内知識の漸増例

(2)の例で、pochiがたとえば、交通事故で足を1本失っていた時は、

```
m⇒s (instance dog pochi (:set leg 3))
```

を送ることにより、インスタンスpochiのlegのみを変更することができる。

(4)スーパークラスの漸増例

dogのスーパークラスをanimalとする。

```
m⇒s (question-class dog super)
s⇒m (answer-class dog :super-not-exist)
m⇒s (d-defclass animal   ～ )
    (d-hierarchy dog animal)
```

もしここで、

```
m⇒s (question-class dog super)
```

とすれば、

```
s⇒m (answer-class dog (animal))
```

となる。

このように、これら知識漸増メッセージと質問応答メッセージを使用することにより、柔軟な質問応答と知識漸増、すなわち、クラス、インスタンス変数等、オブジェクト指向型知識の全てに関する質問応答と変更追加が可能となる。

動的オブジェクト機能を持つCLOSとの比較について述べる。

CLOS[Bobr 87]はLispにオブジェクト指向プログラミングを融合させたものであり、クラス再定義時にその下のインスタンスも再定義される機能がある（いつインスタンスが再定義されるかはインプリメントによるが）。これを本システムの動的オブジェクト機能と比べてみる。

CLOSでは、新インスタンス変数を追加するときも、旧インスタンス変数をクラス再定義内に陽に記述しなければならない。すなわち、追加分だけを記述することはできない。差分のみを記述したい時は、サブクラスを作らねばならない(差分プログラミングの徹底)。　一方、本方式では、知識の追加・変更分のみを記述すればよく、動的変更機構がそれをもともとのクラスに融合させる。

CLOSでは、旧インスタンス変数値と新インスタンス変数値との関係、すなわち、overwrite/reserve/等は固定的であり、自由に指定することは難しい。一方、本方式では、各種追加・変更を可能にするための各種メッセージを用意し、自由に指定できるようになっている。

これらの差は、本システムが「知識は常に変化する」という立場にたって、オブジェクト指向型知識表現を考えていることに起因する。

4.3 知識のスコープ機能

従来のオブジェクト指向知識はクローバルであり、すべてのIAから参照することができた。しかし、IAが異なれば、知識も異なる方が自然であり、同名のオブジェクトでも異なるIAに所属するならば、それらは別物であるべきである。また、IAグループ内のみで共有する知識も必要である。

そこで、IAを単位としたスコープをオブジェクト指向型知識に導入した。

(1)グローバル知識(Global Knowledge)

全てのIAにより共有される。

(2)固有知識(Own Knowledge)

IAに固有の知識である。

managerは、ある物に関するクラス定義すべてを最初からsolverに与える必要はない。solverが問題解決に必要な部分定義を送ればよい。よって、ある物に関するmanagerに固有の知識とsolverに固有の知識は必ずしも一致しない。　たとえば、クラス名が同じdogでも、managerのdogと、solverのdogは別物である。

このような知識が固有知識であり、本方式では、クラス名の頭にIA識別番号を付加して区別している。

よって、たとえば、managerがdogに変更を加えても、solverのdogに影響は及ばない。

(3) IA間知識(Inter-Agent Knowledge)

IA間知識はIAグループによって共有される。　例えば、(make-instance 'mailbox ~)により生成され、複数IAによりプロセス間クロージャとされたメイルボックスはそれら複数のIA間知識であり、グループ外のIAからは見えない。(現在のところ、このIA間知識は1台のELIS内に限定されている。)

5．おわりに

並列動作する多数の分散化疎結合コンピュータが互いに協調して、coherentな問題解決を図る並列協調型システムHarmoniaの論理モデル、Ethernetで結合された4台のAIワークステーションELISからなるHarmoniaの実験システム第1版の基本構成、通信機能、知識漸増・質問応答機能について述べた。

通信機能に関して言えば、普通通信と緊急通信という二レベルの基本通信とその組み合わせにより、各種の通信が可能になっている。また、Harmoniaの通信主体である分散化IAの仮想化を目指し、その第一歩として、普通通信と緊急通信において、IAの存在するELIS名ではなく相手IAを記述すればよいように通信サーバを構築した。

また、知識処理に対応するためHarmoniaは動的オブジェクト機構を具備し、それにより柔軟な知識漸増と質問応答

を可能にしている。

現在のオブジェクト指向言語を知識表現言語という立場から見た時に最も不便な点は、オブジェクト指向言語においては、新たな機能付加は新たなサブクラスを生成するという差分プログラミングによって実現されるということである。本稿で示したように、並列協調型システムでは知識の変更追加は、サブクラス生成という形でこれを吸収するよりは、単に差分を与え、あとはオブジェクトの動的変更機構によって、古いオブジェクトと差分とを融合する方法の方がより自然であり、クラス数も増加しないので効率的でもある。

以上のような基本的ツールである多種類のIA間通信機能と豊富な知識漸増・質問応答機能により、Harmoniaでは多彩な並列協調型問題解決が可能となっている。

Harmoniaによる並列協調型問題解決プログラミングは稿を改めて報告することとする。

また今後は、並列協調環境でのデバッガ、各種通信機構の最適化、並列協調化の評価法について検討する予定である。

最後に、御討論、御教示いただいた竹内郁雄、日比野靖、大里延康、村上健一郎、天海良治、山崎憲一諸氏に深謝する。

＜参考文献＞

[Bobr 87] Bobrow, D.G., DeMichiel, L.G., Gabriel, R. P., Keene, S., Kiczales, G., and Moon, D.A. : Common Lisp Object System Specification, Draft X3 Document 87-002 1987.

[Davi 80] Davis, R. : Report on the Workshop on Distributed AI, SIGART Newsletter No.73, 1980.

[Davi 82] Davis, R. : Report on the Second Workshop on Distributed AI, SIGART Newsletter No.80, 1982.

[Erma 80] Erman, L.D., F.Hayes-Roth, V.R.Lesser, and D.R.Reddy : The Hearsay-II Speech-Understanding System : Integrating Knowledge to Resolve Uncertainty, Comput. Surv., vol.12, No.2, pp.213-253, 1980.

[Fehl 83] Fehling, M. : Report on the Third Annual Workshop on Distributed AI, SIGART Newsletter No.84, 1983.

[Film 84] Filman, R.E. and Friedman, D.P.: Coordinated Computing, McGraw-Hill, 1984.(訳本 協調型計算システム、 マグロウヒルブック)

[Gass 86] Gasser, L. : The 1985 Workshop on Distributed AI, AI Magazine, Summer, 1987.

[Hoar 85] Hoare, C.A.R. : Communicating Sequential Processes, Prentice-Hall, 1985.

[Korn 81] Kornfeld, W.A., and C.E.Hewitt : The Scientific Community Metaphor, IEEE Trans. Syst. Man Cybern., vol.SMC-11, No.1, pp.24-33, 1981.

[Mura 87] 村上健一郎:LANプロトコルのオブジェクト指向による実現、情報処理学会 第34回全国大会、1P-7, 1987.

[Onai 86-1] 尾内理紀夫: Occamとトランスピュータ,共立出版, 1986.

[Onai 86-2] 尾内理紀夫、竹内郁雄：TAOによる通信と同期の実現、情報処理学会 第32回全国大会 2G-8, 1986.

[Onai 86-3] 尾内理紀夫、竹内郁雄：TAOによる並列問題解決プログラミング、電子情報通信学会 COMP86-58,1986.

[Onai 87-1] 尾内理紀夫、鶴岡行雄 ：オブジェクトの動的変更機構、情報処理学会 第35全国大会 3R-2, 1987.

[Onai 87-2] 尾内理紀夫、鶴岡行雄 ：動的オブジェクトによる知識漸増と質問応答、ソフトウェア科学会 第4回全国大会 A-2-1, 1987.

[Post 81] Postel, J. :Transmission Control Protocol, DARPA Internet Program Protocol Specification, Sept. 1981.

Smit 78] Smith, R.G., and Davis, R. : Distributed Problem Solving : The Contract Net Approach, Proc. 2nd Natl. Conf. Canadian Soc Comput. Stud. Intell, Toronto, pp278-287, 1978.

[Smit 85] Smith, R.G. : Report on the 1984 Distributed AI, AI Magazine, Fall, 1985.

[Suzu 85] 鈴木則久編: "オブジェクト指向 解説と WOOC' 85からの論文",共立出版,1985.

[Take 83] Takeuchi, I., Okuno, H., and Ohsato, N. : TAO-A harmonic mean of Lisp,Prolog and Smalltalk, ACM SIGPLAN Notices,Vol.18,No.7,1983.

[Take 86] Takeuchi, I., Okuno, H., and Ohsato, N. : A List Processing Language TAO with Multiple Programming Paradigms, New Generation Computing, vol.4, No.4, p.401-444, 1986.

[Take 88]竹内郁雄 : マルチパラダイム言語TAO, to appear in bit 1月号 共立出版、1988.

[Tani 87] 谷島宜之 : AI専用ﾒｰｶｰ、AI市場拡大を狙い新製品を続々投入、日経コンピュータ、10.12, pp111~122, 1987.

[Tsur 87] 鶴岡行雄、尾内理紀夫 : 動的オブジェクトの遅延変更機構、ソフトウェア科学会 第4回全国大会 A-2-2, 1987.

[Yama 86] 山田,大野,日比野,竹内: "AIワークステーションELISの検討" ,情報処理学会研究報告(マルチメディアと分散処理), 86-MDP-31-2, 9月,1986.

[Yone 85] 米澤 明憲: "オブジェクト指向並列モデルとその記述言語ABCL", 昭和60年電気・情報関連学会連合大会, 35- 6,1985.

[Wata 87] Watanabe, K., Ishikawa, A., Yamada, Y., and Hibino, Y. : A 32b List Processor,IEEE International Solid-State Circuits Conference, pp.200~201. 1987.