－論文－

高岡短期大学紀要　第２巻　平成３年３月
Bull. Takaoka National College, Vol. 2, March 1991

コンピュータによる協働支援システムの基本的枠組（１）

—— メッセージ通信とサーバの設計 ——

小郷　直言・米川　覚

（平成２年11月１日受理）

要　旨

われわれは，集団における協働活動を支援するソフトウェアである「もんじゅ」と呼ぶシステムを開発した。もんじゅはＴＳＳ上にクライアント／サーバモデルをシミュレートする方法で構築されている。本論文では，システムを構成している基本的な枠組であるメッセージ通信とサーバの設計について詳しく述べる。

キーワード

協働システム，グループウェア，クライアント／サーバモデル，オブジェクト指向，プロダクションシステム

1　はじめに

われわれは協働を支援することを目指した「もんじゅ」と呼ぶコンピュータシステムをタイムシェアリングシステム（ＴＳＳ）上に構築し，実際に利用して評価を行っている[1)]。もんじゅは，共通の目的を持って集まった人間の集団の中で，コンピュータが単に個人的な道具として使われるというだけでなく，他人との情報・知識の交換，意見の発表，集団による学習・意思決定などの手段を提供し，問題意識を持つもの同士が継続的に，かつ間接的に双方向に対話できるような場を与えることを強く意図している。

もんじゅでは，人々の協働をコンピュータにより支援することを目的に，メッセージ交換，電子メール，電子掲示板，電子黒板，リアルタイムアンケート，オンラインテキスト，エキスパートシステムによる診断，文書の共同執筆・添削支援，マルチメディアによる情報の提示，共有メディアを用いての意見の発表と交換，ビジネスゲームによる学習，親しみやすい入力形式など様々な機能を提供する。

本論文では，もんじゅのシステム設計で中心となった基本的な枠組である，メッセージ通信とサーバの設計について詳しく述べる。もう一つの柱である双方向性とヒューマンインターフェースについては次号において述べる。

2　メッセージ通信

メッセージ通信と共有メモリは離れて存在する別々のコンピュータ（プロセス）間の通信を可能にする一形態である。離れたところにある別のコンピュータに対して処理の依頼をする方式の一つにクライアント／サーバモデルがある[2)]。ここではこれを基本的方式として検討する。さらにシステムではこの方式に対話性を導入する試みを行っている（これについては次号において述べる）。

産業情報学科

2.1 クライアント／サーバモデルの模擬[3)]

まず，モデル構築にあたって用いた実際のハードウェアについて少し触れておく。われわれは汎用コンピュータ上のＴＳＳを用いてシステムを構築した。協働という多人数を想定した活動を支援しようとするソフトウェアは，ある程度の端末台数がなければ，実際の使用実験を積み重ねることができない。システム開発の趣旨から，実際にシステムを利用して，そのシステムの動的な耐用性と柔軟性を見極めることが最優先の課題であった。そこで最初のシステム設計は，現在利用可能な汎用機によるＴＳＳ環境を利用して行うことにした。ただし，将来，システムはＬＡＮ(ローカル・エリア・ネットワーク）に移植することを考慮して設計されている。

ＴＳＳを分散システムと見立てることはできないので，正確な意味での分散ソフトウェアを設計することはできない。そこでシステム設計はＴＳＳ上でメッセージ通信を擬似的に行うという方法をとった。この方法を用いて，システムはクライアント／サーバモデルを基本的方式として模擬した。同様に，分散システムや並列プロセスに対して用いられる同期あるいは非同期という通信形態もＴＳＳにそのまま持ち込むことはできない。この理由から，われわれは模擬されたクライアント／サーバモデル上に，リモートプロシージャコール（Remote Procedure Call，遠隔手続き呼び出し，以下ＲＰＣと略す）を初め，いろいろな通信形態をソフトウェア的に工夫して取り入れている。

システムではＴＳＳを用いている制約から，各ＴＳＳ端末をネットワーク上のコンピュータに対応するものと見なす(図１参照)。そして，端末上のアプリケーションプログラムは分散処理の実体として位置付けられる。クライアント／サーバモデルでは，プログラムは分散している別々の処理装置で実行されるが，ＴＳＳでは，すべてホストコンピュータの一つの処理装置の上で実行される。この理由からシステム全体は分散処理のような並列処理ではなく，並行に処理が進行する（以後，本来なら並列という言葉を使うべきであるところも，われわれのシステムの制約上，並行という言葉を多用することになる)。

端末上の応用プログラムは利用者である人間とコンピュータとがインタラクティブに対話できる環境を提供する。この意味で端末はヒューマンインターフェースとしての役割を

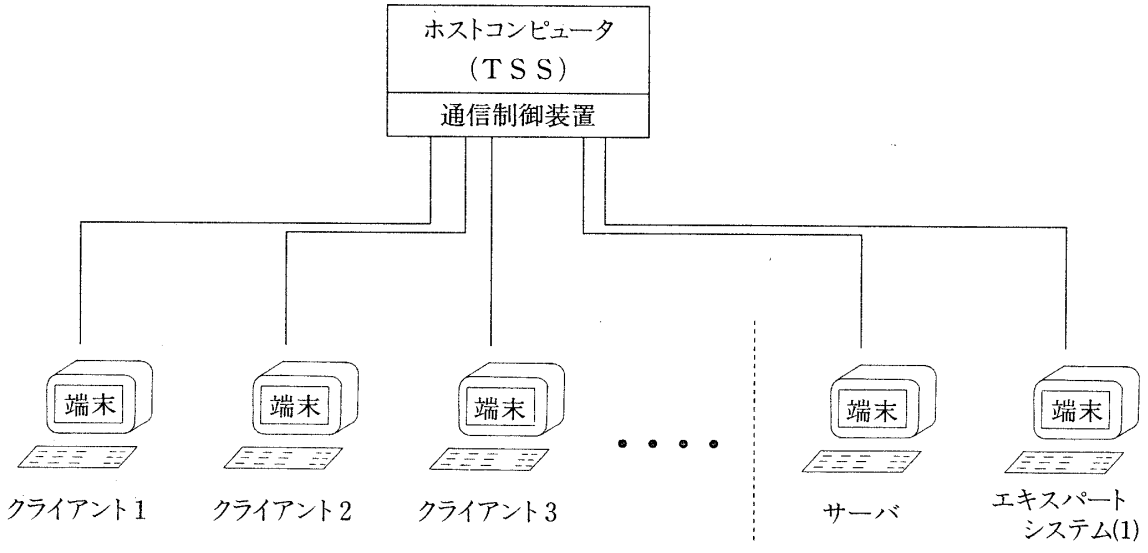


図１　ＴＳＳ上のクライアントとサーバ

担う重要なサブシステムである。以下，ここでは，利用者である人間あるいは端末上のヒューマンインターフェースを担う応用プログラムのことを，クライアントと呼ぶことにする。

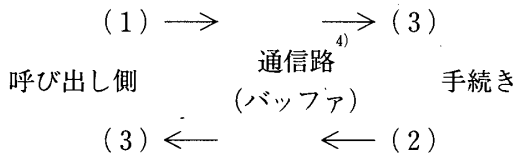
クライアント／サーバモデルの本質的な課題の一つは，分散したプログラムの間で，いかにして通信を行い，またいかにして同期をとるかということである。これにはＲＰＣの技法を採用した。

2.2　リモートプロシージャコール（ＲＰＣ）

ＲＰＣは一つのプログラム（正確にはプロセス）が，別のコンピュータ上のプログラムの中にある手続きを，あたかも自分の手続きであるかのごとく呼び出す機能である。手続きを実行するのは，手続きを呼び出したプログラムではなく，手続きを持っている別のプログラムである。これが本来のＲＰＣの意味である。しかし，ＴＳＳを利用しているわれわれのシステムでは，別のコンピュータ上のプログラムとはＴＳＳの他の端末上で実行される別のプログラムということになる。

まず，手続きを呼び出すプログラムは，手続きの実引数を，手続きを持っているプログラムへ送り，結果を受け取るまで実行を中断する。手続きを持っているプログラムは，受け取った実引数で手続きを呼び出し実行する。手続きの実行が終わると，その結果の値（返り値）を，元のプログラムへ送り返す。手続きを呼び出したプログラムは，結果の値を受け取ると，中断していたプログラムの残りの部分の実行を再開する。以上の流れを図示すると図２のようになる。

ＲＰＣは，プロセス間の相互作用を可能にするための一つの手段である。ＲＰＣではメッセージパッシングによる方法と同じように共有領域を必要としない相互作用が行える。図２に示した通信路にバッファを用いるのは共有領域のように見えるかもしれないが，このバッファはパケットを受け渡す中継地の役割を果たすに過ぎない。このような構成をとらざるをえなかった理由は，ＴＳＳにはコンピュータ同士を結ぶＬＡＮのような通信路が存在しないからである。



1：手続き名と実引数を送り実行を中断して待つ
2：結果の値を返す
3：受信[5]

図２　リモートプロシージャコール

システムでは図２の呼び出し側をクライアントとして，また手続き側を拡張し，サーバとして設計する。通信路を行き来する情報は通常パケットの形をとる。クライアント／サーバモデルでは複数のクライアントから要求が出され，サーバがそれに答えるというのが最も典型的な型である。このとき，主導権は常にクライアント側が持ち，サーバは受け身に，依頼された仕事を処理する。２つ以上の依頼がほぼ同時に到着した時には，決められたスケジューリング方式で順次処理される。クライアント／サーバモデルでサーバが一つだけの構成を取ると，クライアントはサーバを介して間接的に結びつけられることになる（図３参照）。サーバに，共有する情報を記憶し，それを他のクライアントが取り出すということが容易に行えるので，このタイプのクライアント／サーバモデルでは共有メモリとしての機能を実現することができる。また，通常ではあるクライアントからの要求はサーバによって処理された後，当のクライアントへ結果として返事が返されるが，場合によっては別のクライアントに返事を送ることもある。これを利用してメッセージ交換や電子メール（メールサーバとしての機能）が簡単に

実現できる。

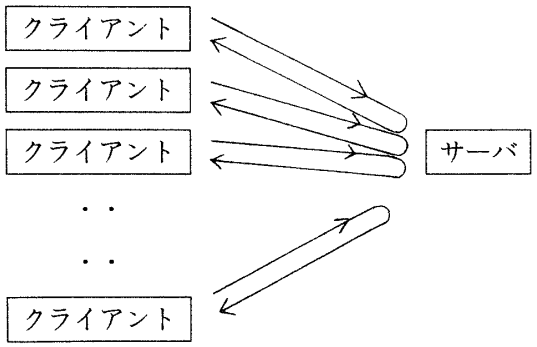


図3 クライアントとサーバ

3 サーバの設計

図3はサーバが一つであるが，サーバが複数になれば，どのサーバに対してメッセージを送るかを指定する必要が出てくる。一つのサーバを一つのオブジェクトと見なすと，これは並行オブジェクト指向の計算モデルに近いものになる。世界を並列・並行に動作できるオブジェクトの集合と見なし，その並行オブジェクトが互いにメッセージを送り合うことによって，計算が行われるとするのが並行オブジェクト指向である。ＴＳＳ上でも，サーバを複数用意して，並行オブジェクト指向のモデルを作ることも可能ではある。しかし，われわれのシステムでは，サーバが一つの構成をとった。

一般には，複数の機能を果たすサーバは送られてきたパケットを読み込み，そのパケットを分解して，呼び出し手続きの指定と，手続きへの引数に組み立て直し，処理を実行する。ただし，ここでは手続き名の指定という方法ではなく，一つのサーバ中にオブジェクトをすべて埋め込むという方式を採ることにした。クライアント側からみると，サーバの中にあるたくさんのオブジェクトに自由に依頼を出せる環境として映ることになる。先の図2で示すと，手続き名ではなしにオブジェクト名を指定する。サーバは一つのプログラムで構成されるが，プログラムはオブジェクト指向プログラミングで実現する。システムでは，オブジェクトの指定，手続き名（メソッドと呼ぶ），引数からなったパケットをメッセージと呼ぶ。

われわれの設計した1サーバの方式では，すべてのオブジェクトを用意しておいて，オブジェクトへのメッセージ送信あるいは関数実行の形で処理が進行する。もっともこれにより，並行オブジェクト指向モデルのようなオブジェクトの並行実行は望めなくなるが，オブジェクト指向プログラミングによるモジュール的なプログラミングのメリットは享受できる。

3.1 オブジェクト指向

3.1.1 オブジェクト

サーバモデルは，メッセージを受理することにより一連の動作（処理）が起動される，「オブジェクト」と呼ぶ抽象的なものの集まりからなっている。「各オブジェクトは，それ自身の中に自立した演算能力を持ち，かつそれ自身のみが直接アクセスし，参照・更新できる局所的な記憶を保持することができる。」[6]

メッセージのやりとりは，系の中で複数あるクライアントとサーバとの間で行われる。しかも，このメッセージ通信がシステムの中で並行的に行われる。

一般にオブジェクトは到着したメッセージを受理することにより，次のような動作・処理の組み合わせが実行される。

(1)局所記憶に格納可能な値に対する種々の演算
(2)局所記憶の内容に対する参照・更新
(3)システム全体に対する手続き的な処理・関数実行
(4)逐次的なメッセージのやりとり
(5)オブジェクトの動的生成

など。[7]

以下の記述では，Lisp（東京大学で開発されたUTILispを使用した）上にオブジェクト

指向のプログラミング機能を追加したものを使用する。記法はフレーバシステム（Flavor system）に準拠してはいるが，まったく同じというわけではない。[8)]

3.1.2 オブジェクトの定義と記述

オブジェクトを定義するには，まずオブジェクトが属するクラスを宣言する（フレーバシステムではクラスに当たるフレーバを def-flavor で定義する）。クラスの定義ではクラス変数とインスタンス変数という二つの局所記憶を決める。さらに，そのクラスがさらに上位のクラスの性格を引き継ぐときには上位のクラス（スーパークラスと呼ぶ）を複数指定できる。

クラスを定義する defclass は次のような形式のマクロである。

```
(defclass クラス名
        (継承するスーパークラス名のリスト)
        (〈インスタンス変数　デフォルト値〉の
        リスト)
        (〈クラス変数 値〉のリスト)
        )
```

スーパークラスはリストで書けるようにして，クラス階層の多重継承（multiple inheritance）を許している。多重継承の優先順位は depth-first-left-to-right で行う。インスタンス変数は各オブジェクトの内部状態を表わすための変数である。インスタンス変数はそのクラスで新たに定義されたものだけを書く。あとはスーパークラスで定義されたインスタンス変数をすべて受け継ぐ。クラス変数はクラス全体に共通した変数とその値を指定する。

オブジェクトの実体は，クラスより生成されるインスタンスによって実現される。インスタンス同士の差は，システム内では唯一の名前と，そのインスタンス変数の値による。

インスタンスの生成は，大域変数名（＝インスタンス名）に次の式を代入して行う。

```
インスタンス名：＝
    (make-instance クラス名
        〈インスタンス変数名1 初期値1〉
        〈インスタンス変数名2 初期値2〉
        ・・・ )
```

各インスタンスは，メッセージを受理すると，受理した各メッセージに対して，どのような一連の仕事を実行するかを決める。この実行を規定しているのは，メソッド定義と呼ばれるプログラムである。メソッドはそれが属するクラスに対して定義される。クラスに属するインスタンス群と当のクラスを（間接的にでも）スーパークラスとするすべてのインスタンスからのみ実行指令を受け入れる。

メソッドの定義は，次のようにして行う。

```
(defmethod　(クラス名 メソッド名)
            (〈引数リスト〉)
             処理本体の式　・・・
            )
```

処理本体は Lisp 式あるいは 3．2 で述べるメッセージ伝達形式で記述する。

3.1.3 フレームモデルとしての利用

フレームとは，ある一定の概念対象を表現するために考え出されたデータ構造の一種であり，M. ミンスキーのフレームに基づく知識表現に端を発する考え方である。[9)] オブジェクトに係わる情報はフレーム内のスロットに持たせられる。スロットには，スロット間の関係のみならず，そのスロットに関する宣言的知識，さらには手続き的な知識を表現できることから（それぞれデフォールト情報がもてる），柔軟でかつ強力な知識表現手法として注目を浴びた。

上で示したクラス，メソッド，インスタンスの記述は，このフレームに基づく考え方に

```
;;;
;;; monju-sys class

(defclass monju-sys () () ())

;;;
;;; monju class

(defclass monju (monju-sys) () ())

 (setq monju (make-instance 'monju))

 (defmethod (monju start) (nm)        ;REGSTRATION TO CLASS
   (let ((ms " " ))
     (cond ((numberp *id-no* ) (setq ms 'id-no-error))
           ((memq *id-no* *class*) (setq ms 'id-no-error))
           (t (set *id-no* (make-instance 'student
                 'name (car nm) 'rec  *rec-no* 'id *id-no*
                 'term (cadr   nm) 'tvno (pop *tv-no*)))
              (push *id-no* *class*)
              ms))))

 (defmethod (monju bye)            ; monju logoff
    (setq *class* (delq  *id-no*  *class*))
    (push ~(eval *id-no*).tvno *tv-no*)
    'ok)

 (defmethod (monju member)
    *class*)

;;;
;;; student class

(defclass student (monju-sys)(name id rec term tvno delay+)())

 (defmethod (student id?) (nil)
   (let (idnoe)
        (if (memq obj *class*)
            (progn (setq idnoe (eval obj))
                   (list ~idnoe.name ~idnoe.rec ~idnoe.id ~idnoe.term))
            'no-exist)))

 (defmethod (student delete) (nil)
    (setq *class* (delq obj *class*))
    (push ~tvno *tv-no*)
    (make-unbound obj))

 (defmethod (student delay)(nil)
    (cond ((neq  ~(eval *id-no*).rec *rec-no*) 'you-are-not-owner)
          ((null ~delay+) 'data-nothing)
          (t (pop* ~delay+ ))))

 (defmethod (student relay)(x)
    (if (car x) (setq x (car x)) (setq x 1))
    (cond ((null ~(eval *id-no*).delay+) 'data-nothing)
          ((null ~delay+) 'data-nothing)
          ((numberp x)
           (nth (1- x) ~(eval *id-no*).delay+ ))
          (t 'data-error)))

 (defmethod (student ojama)(data)
   (let (cag3)
    (string-amend d200 *d200_o*)
    (princ (car data) d200)
    (setq cag3 (make-string 44 " "))
    (string-amend cag3 "OJAMA" 23)
    (mb)
    ))
```

図4 クラス，インスタンス，メソッドの記述例

当てはめることができる。概念の階層性もクラスの継承機構で表現できる。図4は，システムにおける記述の一例である。

3.1.4 オブジェクトの創生

オブジェクトは動作の一つとして，動的にオブジェクトを創生することができる。クライアントはクラス，メソッド，インスタンスの各定義と初期化に必要な情報をメッセージの形で送る。送られたLisp式はサーバのインタープリタにより評価され，新しいオブジェクトが作られることになる。すなわちオブジェクトの一つの機能として，オブジェクト自らが別のオブジェクトを作り出すことができる。

3.2　メッセージの送信

オブジェクトに対して仕事を依頼するには，オブジェクトに対して一定の形式でメッセージを送信しなければならない。

パケットで送られてきたメッセージは内部で次の形式に組み立て直され，send関数を使ってインタプリタにより実行される。

(send　レシーバ　メソッド　引数　・・・)

この式は，レシーバ（メッセージの受け手でインスタンス名を指定する）となるオブジェクトがメソッド（セレクタとも呼ぶ）以降のメッセージ（メソッドと引数の並びを包括したものをメッセージと呼ぶ）を受け取ることを表わす。ここでレシーバと引数はすべて評価されるが，メソッドは評価されない。

図5は，メッセージがサーバの中で処理される流れを簡略化して示している。

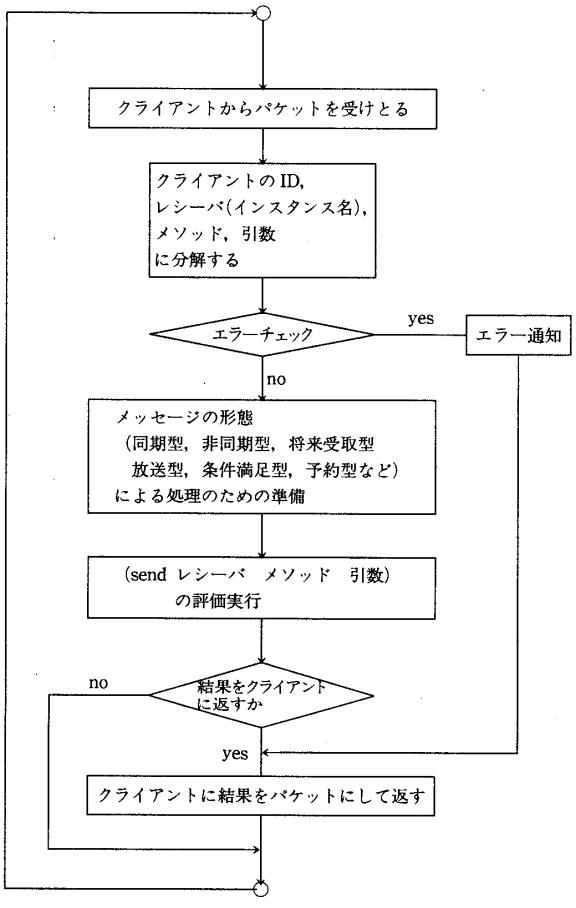


図5　サーバにおけるメッセージの処理

3.2.1 メッセージのやりとり

システムでは通常，端末利用者である人間がサーバにあるオブジェクトに対してメッセージを送るという形式を取る。

メッセージのやりとりの一般的な特徴は次のように要約できる[10)]。ただし，O(オー)は任意のオブジェクトを表わすものとする。

(1)メッセージを送る時点でOの存在を知っていなければ，Oにメッセージを送ることはできない。ただし，初めから知っている場合もあれば，新たに知る場合もある。また忘れてしまうこともある。

(2)Oにメッセージを送るとき，Oの状態やモードに無関係にメッセージを送ることができる。複数の送り手側は，これにより形の上では並行して仕事を進められる。

(3)送られたメッセージは宛先に有限時間内に必ず到着する。ただし，宛先のオブジェクトが存在しない，あるいは消滅していた場合，適当なエラー処理がされてそのことが通知される。

(4) オブジェクトに送られたメッセージは，概念的には，処理されるまで送り元ごとに用意された待ち行列に並ぶ。ただし，現在その長さは2つまでに制限されている。また，送り元ごとのメッセージの取り出し順序はＦＩＦＯではなくポーリング形式で行う。

(5)Oに対してメッセージＭ1とＭ2をこの順に送ると，Ｍ1とＭ2はこの順に到着し，処理される。これにより送り手側は逐次的な依頼による仕事を遂行できる。

3.2.2 メッセージの形態

システムではメッセージのやりとりを通して，人間とコンピュータとがコミュニケーションを行う。さらにシステムでは人間同士を間接的に結び付ける形態をとっている。

オブジェクトへのメッセージの送信として，システムでは次の同期型，非同期型，将来受取型という三つの基本型が用意されている

(図6左側参照)。

1) 同期型

同期型は，コンピュータに仕事を依頼し，その依頼に対する返事や，依頼した仕事の結果が返送されるまで自分の仕事を中断して待つという通信の形態である。これは主にコンピュータと同期をとって仕事を進める場合に使われる。

同期型のメッセージのやりとりは，いわゆるＲＰＣと同じものである。ただし，返事を返すことが，オブジェクトの一連の仕事の終了とは必ずしも言えないという意味でＲＰＣより一般性がある。

2) 非同期型

非同期型のメッセージのやりとりは，人がコンピュータに（メッセージを送って）仕事を依頼するが，その依頼に対する結果を必要とせず，すぐに自分の仕事の続きを行いたい場合に使用する。あるいは単純に情報を伝えるだけの目的にも使う。

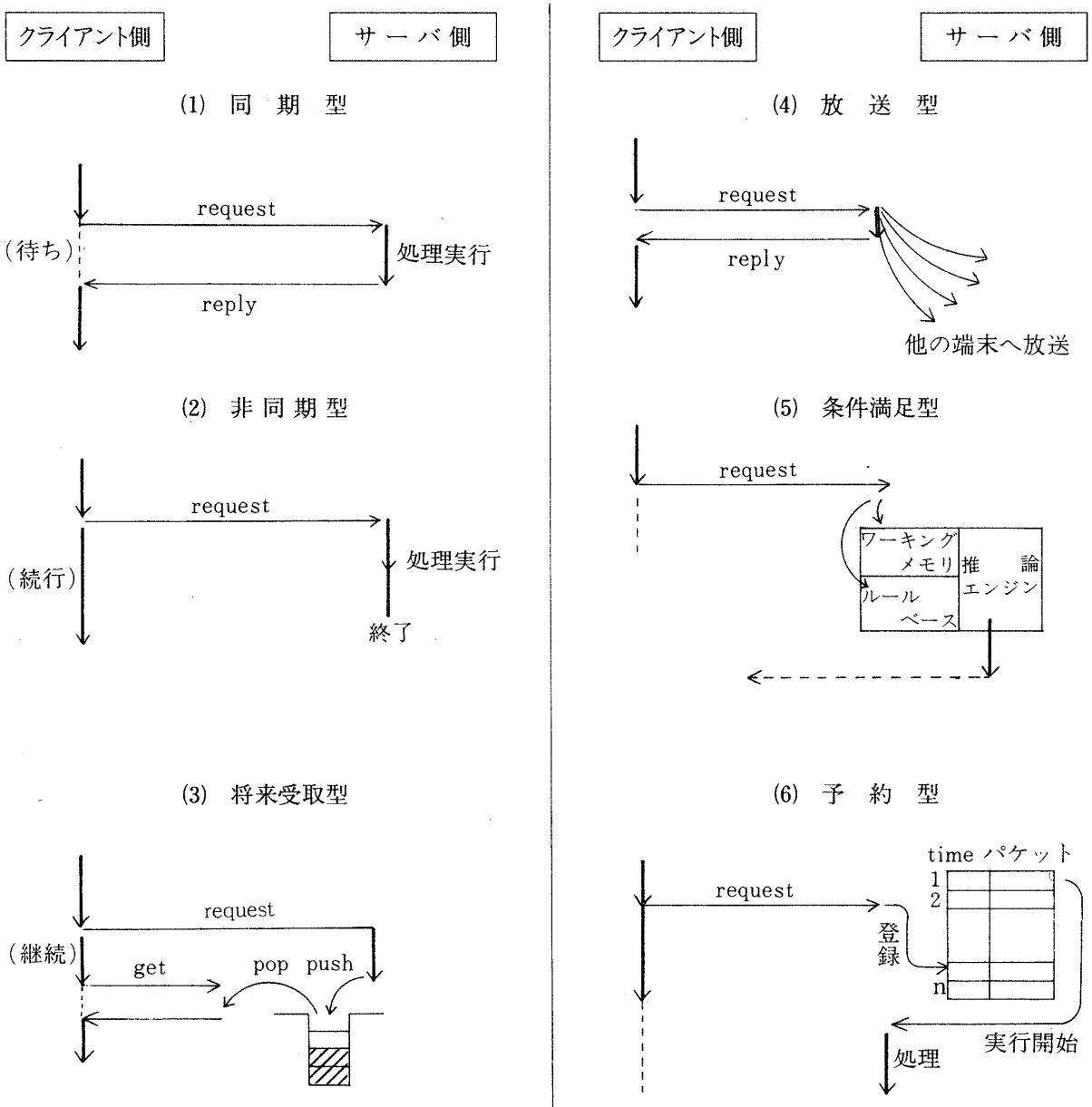


図6　メッセージの形態

3）将来受取型

将来受取型というのは，人から依頼された仕事はコンピュータにより処理実行されるが，その結果は今すぐには必要としないので，その人だけが中を調べることができる特定の場所（スタック形式の記憶）に仕事の結果を保持しておくという方式である。後で，その結果が必要になったときに，それをLIFO(Last In First Out)方式でreply命令を使って取り出すことができる。人はオブジェクトにメッセージを送って仕事の処理を依頼はするが，その返事をその時点では受け取らずに，自分の仕事を直ちに継続して行う。

以上の基本形態の他にも，メッセージのバラエティを増やして利用の便利さを支援する意味から，次のものが用意されている（図6右側参照）。

4）放送型

これはメールの機能を拡張したものであり，放送のように全員に，あるいは特定のグループのメンバーに，同じメッセージを伝えたいときに用いる。

5）条件満足型

オブジェクトの中には，条件が満たされた時にだけ実行されるイベント駆動型のプログラムを組み込める。処理条件を設定しておけば，条件が満たされた時にのみ手続きが実行されるような方式をとれる。この実現には，4.2で説明するプロダクションシステムを用いる。

6）予約型

予約型では，システムに組み込まれたタイマーを用いて，処理時刻（現在から何時間何分後）を指定した形で，手続きを依頼する。これは非同期型の特殊な用途として使われる。

3.2.3 返答の宛先

同期型によってメッセージを送り仕事を依頼する場合，メッセージの受け手であるサーバは受け取ったという知らせ(acknowledgement)，あるいは依頼された仕事の結果を送り返さなければならない。このため，どの依頼者からのメッセージであったかをサーバは当然知っていなければならない。

この返答の宛先(reply destination)は，通常メッセージを送ってきた依頼者本人になっている。システムではパケットに一緒についてくる依頼者名が返答先変数（大域変数）に束縛されているので，通常これを利用する。しかし，意識的に返答の宛先を別の依頼者(端末利用者あるいはエキスパートシステムのどれか）にすることもできる。返答の宛先が他の端末利用者である時には，メッセージ伝達や電子メール的な利用に相当する。この場合にも，依頼者には処理が実行されたという知らせが通知される。もっとも，この通知が必要ないなら，非同期型でメッセージを出せばよい。

3.3 設計方針

本システムにおけるオブジェクト指向プログラミングの実現は，Smalltalkなどに見られるように[11]，計算・情報処理に関するすべての概念をオブジェクトとして捉えるという純粋な立場は採っていない。ここで採ったアプローチでは，オブジェクトの挙動の定義における基本的な値やデータ構造(数やリストなど)はLisp構造をそのまま用い，数やデータ構造への操作は関数呼び出しの手法を用いている。また制御構造もメッセージのやりとりに還元せず，if-then-else, while, and, orなど手続き型のものを利用する。

システムの記述はLispを基礎に，3.2で示したオブジェクト指向とプロダクションシステム（4.2で述べる）をLisp上に作成したものを用いている。このためプログラミングする上で，Lispの諸関数がオブジェクト内の動作としてそのまま記述できることは実用上極めて有利である。

複数の依頼者の活動はコンピュータの処理

能力にも依存するが，基本的には並行的な活動と見なしてよい。各依頼者はサーバのオブジェクトに対してめいめい気儘にメッセージを送るが，各依頼者の依頼は時間的に重なってもよく，他を気にせずにメッセージを送信できる。

分散システムの純粋な枠組では，大域的な時間や，大域的な情報の存在を否定する場合もあるが，システムでは現在これを厳密に守るという立場には立っていない。

4 プロダクションシステム

4.1 プロダクションシステムの利用

プロダクションシステムは，知識を＜IF-THEN(認知－行動)＞形式で表現するプログラミングシステムの一つである。

システムの中でプロダクションシステムは，サーバとエキスパートシステムの両方で利用される。サーバの中では，トリガーあるいはデーモンとして機能する。エキスパートシステムは一つの独立したシステムとしてプログラミングされる（この点については次号で述べる)。

次に述べるプロダクションシステムは，UTILisp 上に作成された。記法はＯＰＳ系のプロダクションシステム[12]にしたがっているが，フレーバのクラス構造 (3.1.2を参照) をそのまま利用するという特徴を持っている。

プロダクションシステムはエキスパートシステムにおいて最もよく利用されてきた知識表現用言語である。プロダクションシステムは，IF-THEN ルールの形式で知識を表現する知識ベースシステムであるが，仮説の評価を行うのに適した後ろ向き推論型のシステムと，目標の合成に適した前向き推論型のシステムとがある。

プロダクションシステムがエキスパートシステムに多く採用される理由は，知識の表現が「もし～ならば，～である」という形式であるため，人間の思考様式に近く，人間にとってわかり易いということがまず第一に上げられる。また，人間には環境・状況の可変性に対応し適切な行動をとる能力がある。その意味でルール型表現は，状況対応的な知識を表現するために最適な形式といえる。いつどこでその知識が使われるかわからないような状況でも，実行の順序に気を使う必要がなく，ルールの形で記述しておけばよい。明確なアルゴリズムの実行を記述するのに比べ，柔軟性が高い表現といえる。ルール表現は元来モジュール性が高いので，新しいルールの追加，修正，削除が容易であり，保守がし易い。

ルール表現は状況の場を通じての相互作用を意図したものであるから，協働を支援するシステムにとって願ってもない記述の手段を与えてくれる。

4.2 プロダクションシステムの構造と記法

プロダクションシステムの構造とわれわれの記法について説明する。

プロダクションシステムの最も基本的な構成要素はルールベースとワーキングメモリー(これらを併せて知識ベースと呼ぶこともある)，それに推論機構であるインタープリタからなっている。

1）ワーキングメモリー

ワーキングメモリーで使う変数はエレメント型と呼ばれる。エレメント型とは，Lisp の属性リスト，フレームのスロット変数，レコード型と同じものである。ここでは，インスタンスのインスタンス変数を用いる。このため，クラスの多重継承の性質をそのまま引き継ぐことになる。エレメントはインスタンスの生成とまったく同じ方法で作り出される。記法について説明するために，例としてよく引用される animal ゲーム[13]を用いることにする。

```
(defclass animal ( ) (species color count) ( ))
```

これはクラス animal の定義であり，インスタ

ンス変数として，動物種，体色，動物の数をもたせる。スーパークラスとクラス変数はない。

エレメントをワーキングメモリーに登録，記憶させるには，make文を使う。例えば，

```
(make animal 'species '狼 'color '灰色 'count 8)
```

とすると，これは動物の一種である狼（種名は狼，色は灰色，数は8匹という属性を持つ）をワーキングメモリーに登録したことになる。[14]

2）ルールベース

ルールベースはプロダクションルール（単にルールとも呼ぶ）の集合からなっている。プロダクションルールは条件部と行動部から構成されている。通常，条件部のことをＬＨＳ（Left-hand sideの略），行動部のことをＲＨＳ（Right-hand sideの略）と呼ぶ。

ルールは，

ＬＨＳ ·····> ＲＨＳ

の形式をとる。ルールの記述は次のようにして行う。

```
(rule ルール名
    ラベル1 (クラス名 式 ・・・ )
    ラベル2 (クラス名 式 ・・・ )
        ・・・
    ⟶
        式 ・・・
    )
```

例をあげて説明しよう。

```
(rule example
  G (animal (= ~species '狼) (<> ~color '銀色))
  -->
    (if (> ~G.count 20)                    ;条件部
        (remove G)                         ;ｔｈｅｎ部
        (modify G ~count (1+ ~count)))     ;ｅｌｓｅ部
        )
```

これはＬＨＳでワーキングメモリの中に，種が狼でかつ体色が銀色でないという条件を満たす事実があるかどうかを調べる。もしも，この条件が満たされたならば，ＲＨＳで群れの数が20を超えているかどうかによって二つの処理が考えられる。数が20を超えているときには該当した事実がワーキングメモリから消される。さもなければ，該当した事実の群れの数を1だけ加えて事実を修正する。これがこのルールの内容である。

~species, ~color はエレメント型のフィールド値（属性値）を参照する。これにより，animalクラスのインスタンス変数である「種」と「体色」の値を参照できる。この条件に対応するＬＨＳのラベルである‘Ｇ’は，条件が満足された事実をルール内で一時的に拘束（バインド）する。[15] これにより，ＲＨＳの~G.countはGに拘束された事実のインスタンス変数countの値を指している。

remove文はエレメントをワーキングメモリから除去するのに使う。一方modifyはワーキングメモリに蓄えられているエレメントの属性値を書き換えるのに使う。

3）インタープリタ

インタープリタは，照合(Match)，競合解消（Conflict Resolution），動作（Action）の繰り返しをコントロールする推論機構である。照合では，ワーキングメモリのエレメントとルールの条件部が満たされるかを調べ，具体化(instantiation)集合を作成する。[16] 競合解消はこの具体化集合（＝競合集合）の中からひとつのインスタンシエーションを選び出し，ルールの行動部を実際に実行する。この際，競合を解消するのにＭＥＡ（Means-Ends-Analysis）という戦略を用いた。[17]

4.3 条件適合，デーモン的な利用

サーバ中のプロダクションシステムの利用は，柔軟性のある共有メモリの一種として用いることができ，オブジェクトの一つの性質として組み込める。利用者がプロダクション

ルールあるいは事実を登録したことがきっかけとなって，条件や事実が満たされ，なんらかの処理の発動がなされることになる。[18] システムではこのような利用法を考えている。

プロダクションシステムの長所は個々のルールが作り易く，わかり易いことである。また，ルールの追加や変更が容易であり，推論メカニズムが簡単であることなどである。一方短所は，ルールの相互関係が不明瞭であり，知識の全体像がつかみにくく，処理効率が悪く，人間の知識構造と異なり推論に柔軟性がないことなどがある。[19] 従って，対象とする問題が小規模なものであれば，長所が生かされ短所は目立たなくなる。知識量が大規模化する場合，対象とする問題が複雑な場合，柔軟な推論を必要とする場合，あるいは推論速度を上げたい場合などには，知識を構造化する努力が必要である。

5 まとめ

ここまで，人々の協働をコンピュータにより支援することを目的に開発した「もんじゅ」について，その基本的枠組のうちメッセージ通信とサーバの設計について見てきた。もんじゅによって提供されるものは，間接的なコミュニケーションの手段と，いくつかのグループウェア的機能である。このためにメッセージ通信はクライアント／サーバモデルを模擬して利用した。また，コンピュータによる処理はオブジェクト指向の方法を採用し，合せて，Lispによる動的特性とプロダクションシステムによる状況の場を通じての相互作用を可能にした。

引用文献・脚注

1）小郷直言，米川覚：“グループ思考支援システムもんじゅの開発とその利用”，全ＮＥＣコンピュータユーザー会，第13回シンポジウム論文集，149-167（1988）.
小郷直言：“教育を援けるコンピュータ”，高岡短期大学放送公開講座『身近なコンピュータ』，131-168，1989.
小郷直言：“コンピュータによる授業支援システム—もんじゅシステム—”，信学技報，**90**-228，ET90-82，87-94(1990).
小郷直言：“コンピュータによる協働支援システム(もんじゅ)の教育環境への応用”，ＣＡＩ学会誌，**7**-4，148-161(1990).
など。

2）Coulouris,G.F. & Dollimore J.: *Distributed Systems : Cocepts and Design*, Addison-Wesley, 1988.

3）クライアント／サーバモデルは，分散ソフトウェアの構成法の一つといえる。分散ソフトウェアとは，ネットワーク上につながれた複数のコンピュータを協力させて，一つの仕事をさせるためのソフトウェアのことである。このことから，システムのプログラムは，ネットワーク上のコンピュータに分散されて置かれることになる。(bit, Vol.20, No.10, pp. 59-67.を参照)

4）ＴＳＳを利用する関係から，通信路はファイル（ＶＳＡＭ形式の相対編成ファイル）をバッファとしてプログラムで共有する形をとった。

5）手続きを呼び出したＴＳＳ端末側では，結果の値が送り返されてきたかどうか，受信キーを押してたびたび調べる必要がある。ループしながら待つという方法もあるが，ＣＰＵ資源を消費するという点からこのような方式を取っている。

6）米澤明憲他：“オブジェクト指向に基づく並列情報処理モデルＡＢＣＭ／１とその記述言語ＡＢＣＬ／１”，コンピュータソフトウェア，**3**-3，9-23(1986).Ｐ10より引用。

7）ibid., p10.

8）Bromley,H. & Lamson,R.: *LISP LORE : A Guide to Programming The LISP Machine*, 2nd

Edition, Kluwer Academic Pub., 1986.
梅村恭司, 大里延康："Lisp上のオブジェクト指向プログラミング", 情報処理, **29**-4, 303-309(1988).
9) Minsky,M. : "A Framework for Representing Knowledge". In P.H.Winston (Ed.) : *The Psychology of Computer Vision*, McGraw-Hill (1975).
10) 米澤明憲他, op.cit.(41).
11) Goldberg,A.,and Robson D.: *Smalltalk-80 : The Language and its Implemetation*, Addison-Wesley, 1983.
12) 田中博, 下井優一：エキスパートシステム構築の方法, パーソナルメディア (1987).
13) 仮説駆動型推論のAnimalゲームの記法の一部を示す。

```
;;;

(defclass goal nil (op hyp status call) nil)
(defclass fact nil (data value rule) nil)


(defun initialize nil
   (setq *ps-wm* nil *ps-dn* nil)
   (make goal 'status 'inactive 'op 'reasoning 'hyp 'ツバメ)
   (make goal 'status 'inactive 'op 'reasoning 'hyp 'ペンギン)
   (make goal 'status 'inactive 'op 'reasoning 'hyp 'ダチョウ)
   (make goal 'status 'inactive 'op 'reasoning 'hyp 'シマウマ)
   (make goal 'status 'inactive 'op 'reasoning 'hyp 'キリン)
   (make goal 'status 'inactive 'op 'reasoning 'hyp '虎)
   (make goal 'status 'inactive 'op 'reasoning 'hyp 'チーター)
   (make goal 'status 'inactive 'op 'reasoning 'hyp '狼))

            :


;;;
;;; 推論用ルール
;;;

(rule wolf1
   g  (goal (eq ~status 'active) (eq ~op 'reasoning) (eq ~hyp '狼))
      (fact (eq ~data '哺乳類) (eq ~value 'yes))
      (fact (eq ~data '肉食獣) (eq ~value 'yes))
      (fact (eq ~data '黄色い眼を持つ) (eq ~value 'yes))
      (fact (eq ~data '丸い鼻を持つ) (eq ~value 'yes))
   -->
      (modify g 'status 'success))

(rule wolf2
   g  (goal (eq ~status 'active) (eq ~op 'reasoning) (eq ~hyp '狼))
      (fact (eq ~data '哺乳類) (eq ~value 'yes))
      (fact (eq ~data '肉食獣) (eq ~value 'yes))
   -->
      (make goal 'status 'active 'op 'ask 'hyp '黄色い眼を持つ 'call '狼)
      (make goal 'status 'active 'op 'ask 'hyp '丸い鼻を持つ 'call '狼))

(rule wolf3
   g  (goal (eq ~status 'active) (eq ~op 'reasoning)
            (memq ~hyp '(狼 チーター 虎)))
      (fact (eq ~data '哺乳類) (eq ~value 'yes))
   -->
      (make goal 'status 'active 'op 'hypothesis 'hyp '肉食獣 'call ~g.hyp))

(rule wolf4
   g  (goal (eq ~status 'active) (eq ~op 'reasoning)
            (memq ~hyp '(狼 チーター 虎)))
   -->
      (make goal 'status 'active 'op 'hypothesis 'hyp '哺乳類 'call ~g.hyp))
```

14) これによって, ワーキングメモリーに一つのインスタンスを登録したことになる。例えば,
(make animal 'species 'wolf 'color '銀色 'count 4)
とすれば, 色の違う別の狼の群れをワーキングメモリーに登録したことになる。
15) animalというクラスに属する事実の中で, 種が「狼」と等しく(=)かつ「体色」が銀色でない(<>)という条件を満たす事実があるかどうかが調べられ, もしも, この条件が満たされたならば, その事実がGというラベルに結びつけられる（拘束されるという)。
16) 照合の第2段階以降では, すべてのルールの条件部とワーキングメモリのエレメントを照合するわけ

ではない。エレメントに変化が起るとその部分だけ照合され，具体化集合の追加，削除を行っている。

17) 競合解消の戦略については次ぎの文献を参照。
鈴木宣夫："ＯＰＳ５文法入門"，*Computer Today,* **13**，11-19（1986）.
Cooper T.& Wogrin N.: *Rule-based Programming with OPS5*, Morgan Kaufmann Pub., 1988.

18) プロダクションシステムの起動は，クライアントからの要求，状況が変化して，ある事象が喚起されたとき，あるいはタイマーの指示など様々のことがきっかけとなって起こる。

19) また，アルゴリズムがはっきりしている，すなわち処理の順序が明確であるときには手続き的な言語で記述する方が当然効率的である。われわれのシステムでは，このような手続き的プログラミングは，Lispによる関数表現を用いている。

The Framework of the Computer-Supported Cooperative System (Part I)

—— Message Communication and the Design of the Server ——

Naokoto KOGOU and Satoru YONEKAWA

(Received November 1, 1990)

ABSTRACT

We have developed a computer-supported system for co-operative action in human groups. The system is called "Monju" and is designed to be used with Time Sharing System, in which the Client/Server Model is simulated.

In this paper (Part I), the configuration of the Monju, in paticular the basic framework, including message communication, and the design of the server, are discussed in detail.

KEY WORDS

Cooperative System, Groupware, Client/Server Model, Object-oriented, Production System