オブジェクト指向型操作的仕様に関する一考察†

松本 吉弘††

ソフトウェアの開発初期段階において，そのまま実行することによって操作的意味を示すような操作的仕様を作成し，実行によって示された意味によって要求を間接的に定義する方法が研究されている．本稿では，筆者の開発したオブジェクトに基づいた操作的仕様記述法を示し，これによってオブジェクト指向操作的仕様のもつ本質について考察する．

1. まえがき

ソフトウェアを製作する場合には，そのソフトウェア（以下目的ソフトウェアと称する）に携わる多くの関与者が存在する．大まかに関与者を分類すると，つぎの2つに分かれる．

関与者1：発注責任者，契約責任者，発注側の計画遂行責任者，同設計者，運用責任者，操作員，保守員

関与者2：製作責任者，設計者，プログラマ，試験者，据え付け調整者

これら関与者は，それぞれ異なる視点からこれから作られようとする目的ソフトウェアに対して，独自の理解（意味，または解釈）をもっている．この理解を，意味世界における抽象であるとみなし，モデル（後に定義する）によって表現する．仕様やプログラムは，なんらかのモデルによって解釈されたときに真になるような論理的記述であると見なされる．例えば，卑近な例では，関数は論理記述であり，ラムダ計算はこれを解釈するためのモデルのひとつである．人は一般に，実世界に対応して，意味世界でのモデルをまず頭に描き，このモデルによって解釈されたときに真になり得るような仕様やプログラムを作成する．合理的に目的ソフトウェアを作成するためには，関係する関与者が目的ソフトウェアに対して抱く解釈（モデル）を事前に分析，評価，選択し，これらを統合することが必要となる．統合された解釈のことを，ここでは統一モデルとよぶことにする．統合とは，複数のモデルによって別個に表現されていた意味を，ひとつのモデル（統一モデル）を用いてまとめて表す操作のことである．統合は，なんらかの手順に従って行われる．たとえば，次のような統合の例がある．関与者1の抱く意味（モデル）が構造化されたデータフローモデルによって表されているとき，これをまず関与者2の提案する CSP[9] を用いて，複数の並行して動作するモデルに分割し，分割された個々のデータフローモデルをさらに関与者2の提案する有限状態機械に変換する．この結果できる communicating automata とも称すべき新しいモデルは，データフローモデル，CSP，有限状態機械という3つのモデルによって表されていた関与者 1，2 のもつ意味を統合したものとなる．

従来のライフサイクル論では，これから製作しようとする目的ソフトウェアに関する関与者1の解釈（要求モデル）が，関与者2に影響されることなく，独立に作られるべきものであるとされていた．このことは，要求モデル，ないしはその形成過程に対して，実現手段からの制約を加えることがないという点ですぐれていたが，同時に要求モデルから実現モデルへの写像を困難にするという欠点への要因のひとつともなっていた．

そこで，ライフサイクルを通して両方の関与者が協力し合って，共通なただひとつの解釈（統一モデル）を形成することとし，それが問題領域，実現領域いずれに対しても写像できるようにし，このモデルを通じて両者が解釈を一致させるような方式が論じられるようになった．このような考えに基づいて設定される解釈（統一モデル），これに基づいて体系化されたソフトウェア形成方法，および形成過程のことをソフトウェア開発パラダイムとよぶ．

モデルという語は，きわめて広い範囲で用いられているので，ここでは論理学の定義[1]に従う．ある目的をもった論理表現を解釈し，その真偽を論ずるために設定する概念の記述や図式のことを，モデルとよぶ．すなわち，設定されたモデルに従って論理表現を解釈することによって，その論理表現の真偽が論じられ

† A Study on an Object Oriented Operational Specification by YOSHIHIRO MATSUMOTO (Heavy Apparatus Engineering Laboratory, Toshiba Corporation).

†† (株)東芝重電技術研究所

る．たとえば，実行可能な計算機コードを一種の論理表現と見なすと，これがアーキテクチャというモデルに従って解釈されて真であるとき，このコードはこの計算機によって正しく実行されると考える．このような考え方を拡張すると，仕様書は，関与者が抱く概念モデルによって解釈されたときに真であるような論理表現であると見なすことができる．

新しいパラダイムでは，第1の段階で問題に対する解法を表明する統一モデル，およびそれを実行する抽象機械を作成し，第2の段階でその解法に対する資源割り付けを行う．ライフサイクルを通して，形成するモデルは統一モデルのみであり，このモデルに従う解釈によって真になるような仕様を作成する．関与者1，2は，この仕様記述，またはその実行を通じて交信し合い，両者の満足がいくまで統一モデルおよびその仕様を改善する．新しいパラダイムでは，従来におけるような要求モデルや要求仕様の作成はせず，統一モデルの仕様，またはその実行によって得られる間接的効果によって，間接的に要求を定義する．統一モデルの仕様として，操作的仕様 (operational specification)[2]，プロトタイプ[3]，形式的仕様 (変換可能仕様)[4] などがある．

新しいパラダイムの特徴は，統一モデルが具体的資源割り付けから離れた抽象機械として表現されるために，関与者1がこれを理解しやすく，その結果として要求モデルを作らなくても，仕様の正当性（要求を満たす度合）を確認できることである．

新しいパラダイムのもうひとつの特徴は，作られた統一的仕様から後の過程，すなわち資源割り付けの過程（変換過程）を半自動，または機械的に行わしめ，新しい要求や修正に起因した途中からの人為的改変を許さない仕組みを考えている点である．これによって，プログラム文面だけで保守が行われ，仕様書の保守が取り残されるようなことを防ぎ，製作と保守の一元化をはかろうとする．

2. 統一モデル

前章でモデルについて述べた．問題が与えられ，モデルを構築する段階では，汎用的な既成モデルが参照される．以下，主なモデル名を掲げておく．

(1) E-R モデル (entity-relationship model)[5]
(2) データフローモデル[6]
(3) コントロールフローモデル[7]
(4) 有限状態機械モデル[8]
(5) ペトリネットモデル
(6) 刺激応答モデル[7]
(7) CSP (Communicating Sequential Model)[9]
(8) 抽象データ型モデル
(9) オブジェクトモデル[10]

一方，関与者2の解釈を記述するに際して参照しやすいモデルとしては，次のようなものがある．

(a) Structured model[11]
(b) Concurrent Processes[12]
(c) Distributed Processes[13]
(d) 関数モデル，論理モデル，オブジェクトモデル

統一モデル構築のためには，これらモデルを参照しつつ，これらのなかから与えられた問題記述に最も適するものを選択する必要がある．統一モデル構築のためには，次に述べるような手順が考えられる．

(1) 各関与者がそれぞれ自分の思想を表すのに適した参照モデルを用いてモデルを構築し，作られたものを統合する．

(2) あらかじめ関与者1，2が参照する唯一のモデルを決めておき，両者がそれぞれこの形式に従ってモデルを製作し，結果を照合することによって統一を図る．

(3) 関与者1が合意する参照モデルに従って関与者2がモデルの作成を行い，結果の正当性を両関与者が共同で検証することによって統一モデルに仕上げる．

以下に述べる操作的仕様は，刺激応答およびオブジェクトモデルに基づいており，(3)の方式に従って作成されることを前提として計画されたものである．

3. 操作的仕様による統一モデルの意味表現

操作的仕様 (operational specification) とは，次のような性質によって説明される仕様のことである．

a) 与えられた問題（要求）に対する解を生成するための機構を抽象的に表現する．

b) その記述にあたっては，実現のための資源は抽象的に表現され，実現のための具体的な資源割り付けを規定しない．

c) インタプリタなどによって実行させることができ，実行によって実現モデル，すなわち想定している目的プログラムに対する設計者の解釈（意味）を表明することができる．

d) 操作的仕様そのものには，それが外部に対して与える効果に関しての明白な記述が行われず，それを実行させることによって間接的にその効果を外部に示

すことができる．

操作的仕様は，統一モデルの操作的意味を記述する．実時間制御システムに対して採用される可能性の高い種々の統一モデルを経験的に調査した結果，本研究では，刺激応答モデルとオブジェクトモデルの統合を採用した．もちろん，これが実時間制御に対して最も適した統一モデルになり得る保証はないが，両モデル，特にオブジェクトは元来統合的な性質をもち，統一モデルに適したもののひとつであると考え，採用した．記述されるものは，（1）モデルのもつ状態，すなわち変数の値のベクトルと，（2）モデル実行の途上で，上記変数に関して値の列を作り出すような状態推移関数，すなわちプログラムである．

操作的仕様は，次に示す4つ組の $i=1,2,\cdots,N$ に関する集合によって表す．ただし，N は刺激応答対の総数であり，各4つ組は互いに独立であるものとする．

$$\{\mathrm{PreCondition}(i)\ \ \mathrm{Stimulus}(i)\ \ \mathrm{Response}(i)\ \ \mathrm{PostCondition}(i)\}$$

各4つ組の操作的意味 (operational semantics) は，外界から目的ソフトウェアに加えられると予想される刺激 Stimulus(i)，それに対して目的ソフトウェアが果たすべき応答 Response(i)，刺激が加えられる前に状態 S がとる初期状態 PreCondition，応答が終了した後に状態 S がとるべき最終状態 PostCondition，およびプログラム OpProg の実行によって与えられる状態 S の変化列によって表されるものと考える．

4．操作的仕様言語

本稿で述べる操作的仕様は，OKBL (Object oriented Knowledge Based Language)[14] という言語によって記述される．これは，Franz Lisp で書かれたオブジェクト指向型言語であり，ABCL[15] をその源とする．OKBL によるオブジェクトの記述形式を説明するために，簡単な例をまず掲げる．数を数えるカウンタという概念は，整数の記憶とこれに対する操作（たとえば，カウントアップ，カウントダウン，カウント問い合わせ）が複合されたものである．OKBL によるこのカウンタの記述はつぎのとおりである．

```
[ : Class カウンタ
    ( : InstanceVariables [回数 := 0])
    ( : InstanceScripts
        ( : receive [ : カウントアップ]
            [回数 := (1+回数)])
        ( : receive [ : カウントは？]
            ^回数))
]
```

もっとも外側にある［‥］に囲まれた部分がオブジェクトを表し，このオブジェクト名はカウンタである．第1行目では，さらに“カウンタ”をクラス名として定義する．カウンタは種々の目的に対して特定化することができる．たとえば，“投球数カウンタ”のようなオブジェクトを生成することができる．クラスを特定化することをインスタンス生成といい，特定化の結果できたオブジェクトのことをインスタンスという．

カウンタから投球数カウンタというインスタンスを生成するためには，つぎのような文を実行すればよい．

```
[投球数カウンタ == [カウンタ <= [ : new]]]
```

この結果生まれた投球数カウンタというインスタンスには，上のカウンタ記述の 2-6 行目がそのまま写される．第2行目にあるのは，オブジェクト変数の宣言である．“回数”がオブジェクト変数名で，この値の初期値は0であるとしている．第3行目からがスクリプトであり，オブジェクトの機能を記述する．“カウントアップ”というメッセージに対しては，回数を1だけ増加して送信元へ終了を報告し，“カウントは？”というメッセージに対しては，回数の値を送信元へ返す．この例では，オブジェクト変数としてインスタンス変数，スクリプトとしてインスタンススクリプトの記述しかないが，クラス変数およびクラススクリプトの記述をこれと同じように併記することができる．クラスオブジェクトがインスタンスを生成した際に，クラス変数およびクラススクリプトは生成されたインスタンスへ写されない．インスタンススクリプトの中でクラス変数が参照，書き換えられる場合には，クラスへ戻ってクラス変数が操作される．

クラスの上位概念を表すクラスをスーパクラスとして宣言することができる．スクリプトに関した継承関係は次のとおりである．

*インスタンスがメッセージを受け，そのメッセージパターンに適合する receive 文がそのインスタンスにない場合に，スーパクラスのインスタンススクリプトの中から探される．もし，そのスーパクラスにも適合するものがない場合，そのスーパクラスになおスーパクラスが宣言されていれば，スーパクラスのスーパクラスのインスタンススクリプトの中から探される．

*インスタンススクリプトの中でスーパクラスのインスタンス変数を参照したり代入したりすることができる．これはクラスのインスタンス変数がインスタンスに写されるだけでなく，クラスの階層関係をたどって，スーパクラスのインスタンス変数もインスタンスに写されるからである．

スーパクラスは複数個宣言することができる．この場合には宣言の順序により継承順序が決定される．

スクリプト内の：receive は，その直後に受信可能なメッセージパターン，そのあとに一時変数の宣言，さらにその後に値をもつ OKBL 式をもつ．メッセージがこのパターンとマッチすると，その後ろにある OKBL 式が順次評価されて，最後に評価された式の値がスクリプト実行結果となる．メッセージがインスタンスのメッセージパターンとマッチしないときには，その上位にあるクラスのスーパクラスへと遡って，マッチするスクリプトを探す．

receive 文のあとに続き得る OKBL 式には，オブジェクト結合式，代入式，メッセージ送出式，SWITCH 式，Lisp 式がある．メッセージ送出式は

[カウンタ <= [：new]]

のようなもので，オブジェクト結合式は

[投球数カウンタ == [カウンタ <= [：new]]]

のように，カウンタから生成されたインスタンスの実体を投球数カウンタにバインドする．代入式はシンボルに値をバインドするためのもので，setq と同じ役割をもつ．たとえば，[Obj := [%Self <= [：new]]] とあれば，クラスが自分自身にメッセージを送ってインスタンスを生成し，その実体を Obj というシンボルにバインドする．SWITCH 式はつぎのようなものである．

```
(: switch '(: a b c)
          (: case [: a *B *C] ...)
          ......
)
```

この switch 式が実行されると，初めに '(：a b c) を評価し，つぎにその評価値と：case のあとにある case パターンのマッチングを行い，マッチした場合にはそれに続く OKBL 式の評価を行う．

Lisp 式は，Franz Lisp のそれをそのまま用いる．

5. 例題による操作的仕様の説明

図1に，簡単化して表した電力系統の例を示す．このなかのひとつの設備に事故が起きたときに，需要家が停電によって受ける被害をできるだけ極小化するために，事故を起こした設備を切り離し，健全設備だけで給電を続けるような系統再構成ソフトウェアが要求されたとする．ここで設備と称するものは，変圧器，送電線，母線のようなもので，それぞれ，属性（たとえば許容容量），外界刺激によって変化する状態（たとえば電流値や開閉値）をもっている．そこで，設備の果たすべき機能を設備クラス，個々の設備の果たすべき機能を設備インスタンス，属性や状態をクラス変数，またはインスタンス変数で表すことにする．

個々のインスタンスに関する要求は次のとおりである．

（1） 各設備において，入力される電力値の総和と，出力する電力値の総和は等しくなる必要がある．

（2） 入力出力が平衡した状態にあって，入力または出力の総和は，自分に固有の許容容量を越えてはならない．

（3） 自分以外のいずれかの設備の事故が発生した後，各設備において，入力の総和が出力の総和より小さくなったときには，入力側の健全な設備に対して不足分の電力を要求する．要求は入力端のもつ優先度に

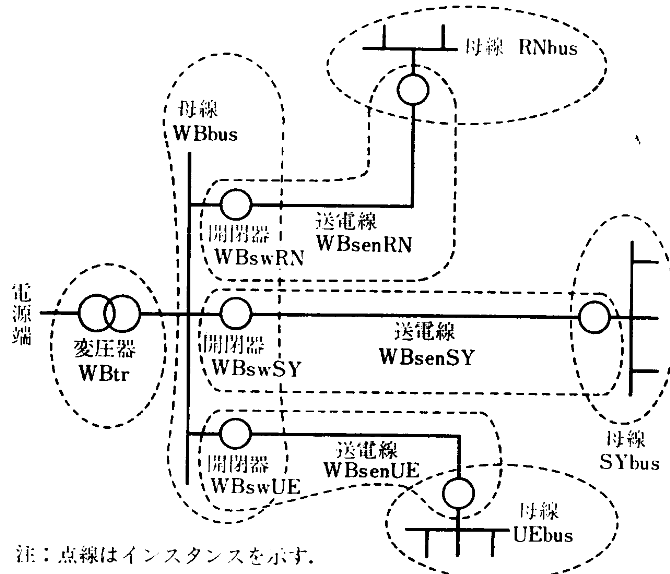


図1 例として用いた電力系統の一部
Fig. 1 A part of electric power transmission system which is used to explain instances for the cited example.

```
[WBtr == [設備 <= [: new 'WBtr 250 'y 'n
            '((WBbus 237 nil nil nil Closed nil
              nil n LoadEnd nil 1 n n))]]]
[WBbus == [設備 <= [: new 'WBbus 250 'n 'n
            '((WBsenRN 58.6 nil nil WBswRN Closed
              nil nil n LoadEnd nil 1 n n)
             (WBsenSY 133.8 nil nil WBsenSY
              Closed nil nil n LoadEnd nil 2 n n)
             (WBsenUE 44.6 nil nil WBswUE Closed
              nil nil n LoadEnd nil 3 n n)
             (WBtr 237 nil nil nil Closed nil nil
              y OldSource 1 nil n n))]]]
;(以下省略)
```

図 2 インスタンスの生成と状態（インスタンス変数）の定義の一部
Fig. 2 A part of description which aims to create instances for the cited example.

従って行う．

(4) 入力増加を要求した結果，入力の総和が出力の総和に及ばないときには，出力端の設備に対して優先度の低い順に電力抑制を行って平衡を保つ．

以上のような要求に対する操作的仕様を説明する．

まず，状態変数を決める．**図2**は，そのごく一部を示している．[WBtr==[設備<=…] は，設備というクラス（**図3**に示す）に対する [: new…] メッセージの送出を示す．このメッセージのなかに含まれる引数，すなわち，'WBtr 250… 以下が状態を表す値で，これがクラスに渡されると，クラスは，これらをインスタンス変数とするインスタンス WBtr を生成する．これら引数値は，図3の LINE 3 において*印のついた変数を束縛し，さらにこれらは LINE 6, LINE 11-12 においてインスタンス変数を束縛する．インスタンス変数それぞれの意味は LINE 8-9 に記載されている．たとえば，'WBtr は設備名に，250 は許容容量に，それぞれ当てはめられる．これら，インスタンス変数が操作的仕様における状態，言い替えれば，操作的仕様実行中の各設備の動的状態を表す．図2では，WBtr, WBbus という2つのインスタンスだけの生成と状態変数定義しか示していないが，他のインスタンスに対しても同様のことを行う．

図3には，設備というクラスから主な記述を抽出して示した．LINE 2-7 がクラスに関する記述で，それ以下は，このクラスが生成するインスタンスへ継承されるインスタンス変数とインスタンススクリプトである．第4インスタンススクリプト (LINE 14) 以下のスクリプトに関する内部記述は省略した．

操作的仕様は，3章で述べたように N 組の4つ組から成るが，ひとつの4つ組を以下に示す．以下の4つ組は次のように読む．

事故が発生していない状態 (PreCondition) にある電力系統に対して，Stimulus によって与える事故が発生したときには，Response に記述するプログラム OpProg を実行することによって電力系統を PostCondition に示す状態に至らしめる．以下の記述では，X を設備クラスとし，x をこのクラスに属する設備のインスタンスとする．また，Y を需要家端供給設備クラスとし，y をそのインスタンスとする．

```
PreCondition:
    (P-is-true-for-any-member-in '(P 1 X))
        ; この関数は，集合 X に含まれる設備イ
        ; ンスタンス x すべてに関して，P 1(x)
        ; の評価値が真であることを要求する．
        ; ただし，P 1(x) は次のような関数であ
        ; る．

    (and
      ([回答 := [x <= [: 状態問い合わせ  '事
       故中]]])
       (members-are-all 'no 回答)
    )
  ; 説明：すべての設備インスタンスに対して，
  ; 隣接設備のなかに '事故中と表示されたもの
  ; がないか問い合わせて，メンバがすべて 'no
  ; となっている回答（リスト）を得なければな
  ; らない．

Stimulus:
    (P-is-true-for-any-member-in '(P 2 X))
        ; この関数は，集合 X に含まれる設備イ
        ; ンスタンス x のすべてに関して P 2(x)
        ; の評価値が真であることを要求する．
        ; ただし，P 2(x) は次に示すような関数
        ; である．

       (do [x <= [: 故障発生]])
  ; 説明：個々の設備インスタンスに故障発生と
  ; いう状態の変化を与える．
  ; この結果，次の Response によって与えられ
  ; るプログラムを実行して，状態 S に操作を
  ; 加える．

Response:
    (execute OpProg)
```

```
#include jdict.ss
   ;日本語変換のためのソフトウエアを組み込む
[:temporary N B then else Found
          電源側設備  一時隣接設備リスト  隣接設備 回答 事故前負荷
          要求電力 供給可能電力 新電源あり 該当あり 不足電力 新電源設備
          確定電力 停電前負荷 余裕電力 事故前電源設備 新要求電力
          回答電力 ブスタイあり 停電前負荷]
[:Class 設備                                              ;LINE1
 (:ClassScripts                                           ;LINE2
  (:receive [:new •設備名 •許容容量 •電源設備かどうか          ;LINE3
              •再入かどうか •隣接設備リスト]
     (:var OBJ)                                           ;LINE4
     [OBJ := [%Self <= [:new]]]                           ;LINE5
     [OBJ <= [:初期化 •設備名 •許容容量 •電源設備かどうか      ;LINE6
                 •再入かどうか •隣接設備リスト]]
     ^OBJ))                                               ;LINE7
 (:InstanceVariables                                      ;LINE8
    設備名
    許容容量
    電源設備かどうか ;(y,n)
    再入かどうか ;(y,n)
    隣接設備リスト      ;この変数は各接続子のつぎの値からなるリストである.
               ;00隣接設備名,
               ;01事故前電力,
               ;02仮定電力,
               ;03確定電力,
               ;04開閉器名称,
               ;05事故前開閉器状態(切,入,ロック),
               ;06仮定開閉器状態(切,入),
               ;07確定開閉器状態(切,入),
               ;08電源側資格性(y,n),
               ;09現在の側(電源側,負荷側,事故中,停電中,新電源),
               ;10電源側資格優先度,
               ;11負荷側資格優先度,
               ;12局配負荷かどうか(y,n)
               ;13母線連絡かどうか(y,n)
 )                                                        ;LINE9
 (:InstanceScripts                                        ;LINE10
  (:receive [:初期化 •設備名 •許容容量 •電源設備かどうか       ;LINE11
               •再入かどうか  •隣接設備リスト]
     [設備名 := •設備名]
     [許容容量 := •許容容量]
     [電源設備かどうか := •電源設備かどうか]
     [再入かどうか := •再入かどうか]
     [隣接設備リスト := •隣接設備リスト]
  )                                                       ;LINE12

;;;;;;;;;;;;;;;;;;;;;;; ;Chapter1 ;;;;;;;;;;;;;;;;;;;;;;;;;
;;設備インスタンスに宛てた[:事故発生]メッセージを処理するためのスクリプト
  (:receive [:事故発生]                                     ;LINE13
     (:var 電源側設備  一時隣接設備リスト  隣接設備 回答) ;一時変数宣言
     (msg N 設備名 B "received message" B "[:事故発生]")
     :(break)
     (if (eq 電源設備かどうか 'n)
      then
       [該当あり := 0]
       (do ((隣接設備リスト 隣接設備リスト (cdr 隣接設備リスト)))


          ((or (eq 該当あり 1) (null 隣接設備リスト)))
        (if (and (eq (fetch 9 隣接設備リスト) '電源側)
                 (eq (fetch 5 隣接設備リスト) '入))
         then
          [該当あり := 1]
          [電源側設備 := (caar 隣接設備リスト)]
        )
       )

     ;;まず電源側の設備にメッセージを送る
      [回答 := 'nil]
      [回答 :=
        [電源側設備 <= [:故障した :頼むよ  設備名]]]
      (if (eq 回答 '処理完了)
        then
         (write 3 隣接設備リスト 0)
         (msg N "In" B 設備名 B ", now finally set the power of branch:"
              B 電源側設備 B "to" "0")
         (write 7 隣接設備リスト '切)
         (msg N "In" B 設備名 B ", if there is a switch of branch:"
              B 電源側設備 B ", please open it")
      )
     )

     ;;優先度順に負荷側の隣接設備にメッセージを送るためその順にならべかえる
     ;;"負荷側資格優先度によりならべかえ"はそのための関数である
     [一時隣接設備リスト := (負荷側資格優先度によりならべかえ 隣接設備リスト)]
     ;;一時隣接設備リストの各要素に対して処理を行うdoループ
     (do ((一時隣接設備リスト 一時隣接設備リスト (cdr 一時隣接設備リスト)))
         ((null 一時隣接設備リスト))           ;要素が尽きたらループから抜ける
         (if (and (eq (fetch 9 一時隣接設備リスト) '負荷側)
                  (eq (fetch 5 一時隣接設備リスト) '入)
                  (eq (fetch 12 一時隣接設備リスト) 'n))
          then
            [隣接設備 := (caar 一時隣接設備リスト)]
            ;メッセージを送る
            [回答 := 'nil]
            [回答 := [隣接設備 <= [:故障した :頼むよ  設備名]]]
            (if (eq 回答 '処理完了)
             then
              (do ((隣接設備リスト 隣接設備リスト (cdr 隣接設備リスト)))
                ((null 隣接設備リスト))
               (if (eq (fetch 5 隣接設備リスト) '入)
                then
                 (write 3 隣接設備リスト 0)
                 (msg N "In" B 設備名 B ", now finally set the power of branch:"
                      B (fetch 0 隣接設備リスト) B "to" 3 "0")
               (write 7 隣接設備リスト '切)
               (msg N "In" B 設備名 B ", if there is a switch of branch:"
                      B (fetch 0 隣接設備リスト) B ", please open it")
               )
              )
            )
         )
     )
   ^'処理完了 ;処理完了という記号を返す
  )

;;;;;;;;;;;;;;;;;;;;;;;;; ;Chapter2 ;;;;;;;;;;;;;;;;;;;;;;;;;
;;設備インスタンスに宛てた[:故障した...]メッセージを処理するスクリプト
  (:receive [:故障した :頼むよ •故障設備名]                   ;LINE14
   ;....(省略)
   ^'処理完了 ;処理完了という記号を返す
  )


;;;;;;;;;;;;;;;;;;;;;;;; ;Chapter3 ;;;;;;;;;;;;;;;;;;;;;;;;;
;;設備インスタンスに宛てた[:停電した...]メッセージを処理するスクリプト
  (:receive [:停電した :問い合せ •要求電力 •停電設備名]          ;LINE15
   ;....(省略)
   ^回答電力 ;供給可能な電力値を回答する
  )
;;;;;;;;;;;;;;;;;;;;;;;; ;Chapter4 ;;;;;;;;;;;;;;;;;;;;;;;;;
;;設備インスタンスに宛てた[:供給頼む..]メッセージを処理するスクリプト
  (:receive [:供給頼む :確定通知 •確定電力]                    ;LINE16
   ;....(省略)
   ^'処理完了
  )
;;;;;;;;;;;;;;;;;;;;;;; ;Chapter5 ;;;;;;;;;;;;;;;;;;;;;;;;
;;設備インスタンスに宛てた[:停電した :給電継続不能..]を処理するためのスクリプト
  (:receive [:停電した :給電継続不能 •停電設備名]               ;LINE17
   ;....(省略)
   ^'処理完了
  )
;;;;;;;;;;;;;;;;;;;;;;; ;Chapter6 ;;;;;;;;;;;;;;;;;;;;;;;;
;;設備インスタンスに宛てた[:状態問い合せ ....]を処理するためのスクリプト
  (:receive [:状態問い合せ  •問い合せ内容]                     ;LINE18
   ;....(省略)
   ^回答
  )
 )                                                        ;LINE19
]                                                         ;LINE20
```

図 3 "設備"クラスの記述

Fig. 3 A part of program describing class "Facility."

; 説明：プログラム OpProg を実行して，状態
; S に関して，事故設備の切り離しと，復旧に
; 対応した操作を加える．

PostCondition :

(P-is-true-for-any-member-in '(P 3 Y))

; この関数は，設備集合 Y に含まれるす
; べてのインスタンス y に関して関数
; P 3(y) の評価値が真であることを要求
; する．ただし，P 3 は次に示すような関
; 数である．

```
(and
  ([回答 := [y <= [: 状態問い合わせ
                   '負荷値比較]]])
  (members-are-all '有意差なし 回答)
)
; 説明 : OpProg を実行した結果，最終の状態
; を各需要家供給端設備インスタンスに問い合
; せし，各設備の負荷側への電力供給が事故前
; と有意差があるほど減少しなかったことを回
; 答（リスト）によって確認する．
```

OpProg の内容およびその実行によって変化する状態の列が統一モデルの意味を表す．OpProg は，OKBS 上で動くクラスおよびインスタンスである．OpProg の一部は既に図3で示したが，その記述を自然文で説明すれば，おおよそ次のようになる．

（1） 故障が発生した設備に対応するインスタンスに故障したことがメッセージ［: 故障発生］によって告げられる．

（2） 告げられたインスタンスは，自分を系統から切り離すための開閉器操作を行い，自分が故障したことを隣接する設備インスタンスへ，重要順に，メッセージ［: 故障した　: 頼むよ …］によって告げる．

（3） 告げられたインスタンスは，この故障によって電力が貰えなくなった場合，不足分を隣接設備に要求する．要求できる順位に従って，メッセージ［: 停電した　: 頼むよ …］を送って不足分が充足されるまで，要求を繰り返す．不足分が充足されないときには，重要度の低い負荷端からメッセージ［: 停電した　: 給電継続不能 …］を送る．

（4） メッセージ［: 停電した　: 頼むよ …］を受けたインスタンスは，自分が発電所であれば出力を増加するなどの方法で対応するが，そうでない場合には，発電所インスタンスに到達するまで，隣接設備へメッセージ［: 停電した　: 頼むよ…］を送っていく．

（5） 要求に対する回答は，要求したインスタンスへ返される．回答されてきた電力供給値が適当である場合には，メッセージ［: 供給頼む　: 確定通知 …］を送ってその要求を確定する．すべての回答が故障した設備インスタンスに帰ってきたところで，OpProg の実行は終了する．

6. 考　　察

（1） 解の構造を事前に確定できないような問題を，独立した複数の知識を適宜使いながらヒューリスティックに解いていくような場合に，オブジェクト指向は有利である．5章では，設備事故に際しての系統構成という問題に対する解が，個々の設備に固有の操作知識の協同処理によって与えられる．ひとつのインスタンスがひとつの設備の操作知識を表し，メッセージ送受が発見過程での単位知識，すなわちインスタンス間の通信を表すので，解の記述が素直に行われる．

（2） 5章では，メッセージ送受の相手を，物理的に接続された隣接設備に関するインスタンスに限定した．これは，現在，人が行っている伝統的な手動操作における情報交換のアナロジイである．すなわち，5章の操作的仕様では，人が従来からこの問題を経験的に解いている過程からの類推（メタフォ）を試みたのである．このことは，仕様の理解容易度，保守容易度を高めるのに大きな助けになる．オブジェクトを指向した場合には，人が行っている解決過程からのメタフォを実現しやすいと考えられる．発達した並列型オブジェクト指向言語を用いて仕様を記述すれば，人の社会活動の基底にある分散型問題解決過程からのメタフォが実現可能と思われる．

（3） 図3の中に見られる (msg...) 文は，インスタンス変数の値，すなわち仕様実行中に状態値を表示する．電力系統構成図などと組み合わせて状態を表示することによって，モデルを表現できる度合はきわめて向上する．また，(msg...) 文において人との対話を行うことによって，希望に従った状態値のみを表示すれば，さらに効率的となる．

（4） 実時間システムの設計では，早い段階で，たとえば応答速度のような性能の予測や，メモリサイズのような資源の予測が強く要求される．OKBL で記述されたインスタンスは実時間マルチプログラミングオペレーティングシステムのもとで動くタスクへ変換する必要がある．現時点では，タスクは従来の手続き型言語で実現するほうが効率が良いためである．半機械的に手作業で変換する手順を定め，インスタンスを書き換える．予測に当たっては，このような変換のことを考慮する必要がある．

7. む　す　び

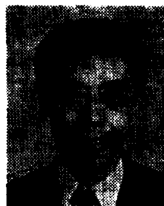
刺激応答およびオブジェクトモデルに基づき，オブジェクト指向言語によって記述されたプログラムの実行により操作的意味を表すようなソフトウェア仕様について述べ，実用のための問題を考察した．

謝辞 終わりに臨んで，ABCL に関して技術指導を賜った東京工業大学，米沢明憲助教授，OKBS の開発に当たった東芝，重電技術研究所の諸氏に感謝を表します．

参考文献

1) 長尾，淵：論理と意味，岩波講座情報科学 7，pp. 13-17，岩波書店，東京 (1983)．
2) Zave, P.: An Operational Approach to Requirements Specification for Embedded Systems, *IEEE Trans. Softw. Eng.*, Vol. SE-8, No. 3, pp. 250-269 (May 1982).
3) Gomma, H. et al.: Prototyping as a Tool in the Specification of User Requirements, *Proc. 5th ICSE*, pp. 333-339 (1981).
4) Balzer, R. M. et al.: On the Transformational Implementation Approach to Programming, *Proc. of 2nd International Conf. on Software Engineering*, pp. 337-344 (Oct. 1976).
5) Chen, P. P.: The Entity-Relationship Model: Toward a Unified View of Data, *ACM Trans. Database Systems*, Vol. 1, No. 1, pp. 9-36 (1976).
6) Ross, D. T.: Structured Analysis (SA): A Language for Communicating Ideas, *IEEE Trans. Softw. Eng.*, Vol. SE-3, No. 1, pp. 16-34 (1977).
7) Alford, M. W.: A Requirements Engineering Methodology for Real-time Requirements, *IEEE Trans. Softw. Eng.*, Vol. SE-3, No. 1, pp. 60-68 (1977).
8) Davis, A. M.: Formal Techniques and Automatic Processing to Ensure Correctness in Requirements Specifications, *Proc. Specification and Reliable Software Conf.*, pp. 15-35 (Apr. 1979).
9) Hoare, C. A. R.: Communicating Sequential Processes, *Comm. ACM*, Vol. 21, pp. 666-677 (1978).
10) Greenspan, S. J. et al.: Capturing More World Knowledge in the Requirements Specification, *Proc. 6th International Conf. on Software Engineering*, pp. 225-234 (1982).
11) Hamilton, M.: Higher Order Software—A methodology for Defining Software, *IEEE Trans. Softw. Eng.*, Vol. SE-2, No. 1, pp. 9-32 (1976).
12) Brinch Hansen, P.: The Programming Language Concurrent PASCAL, *IEEE Trans. Softw. Eng.*, Vol. SE-1, No. 2, pp. 199-207 (June 1975).
13) Brinch Hansen, P.: Distributed Processes: A Concurrent Programming Concept, *Comm. ACM*, Vol. 21, No. 11, pp. 934-941 (Nov. 1978).
14) 松本吉弘：オブジェクトモデルに基づいたソフトウェア設計，*bit*, Vol. 17, No. 10, pp. 19-30/Vol. 17, No. 11, pp. 67-72/Vol. 17, No. 12, pp. 70-82 (Oct./Nov./Dec. 1985).
15) Yonezawa, A. et al.: An Object Oriented Approach for Concurrent Programming, *Research Report*, C-63, Dept. of Information Science, Tokyo Institute of Technology (Nov. 1984).

(昭和 62 年 9 月 8 日受付)
(昭和 63 年 1 月 19 日採録)



松本 吉弘（正会員）

1932 年生．1954 年東京大学電気工学科卒業．同年（株）東芝入社．1974 年東京大学より工学博士の学位を受領，1979 年同社重電技術研究所技監，1985 年理事，現在に至る．この間，電力発生のための制御方式の開発，計算機制御システムの開発，ソフトウェアの生産技術の開発などに従事．日本電機工業会進歩賞，全国発明表彰，科学技術庁研究功績者表彰，IEEE フェロー表彰など受賞．現在 IEEE Transactions on Software Engineering の associate editor，筑波大学，慶應義塾大学大学院非常勤講師も務めている．「計算機制御システム」など著書 10 点．電気学会終身員，IEEE フェロー，日本ソフトウェア科学会会員．