TECHNOLOGY REVIEW

UNISYS 技報 通巻 36

1993年2月発刊 Vol. 12 No. 4

論 文

日本ユニシス

ISSN 0914-9996

ソフトウェア開発の形式的方法のひとつであるウィーン開発技法VDMの仕様記述言語VDM-SLの規格化が進行している．山崎利治は，**ウィーン開発技法VDM**の中で，その紹介を兼ね，OR関係のソフトウェアを手掛けているチームへ形式的方法の導入を企画して行った実験として，ヒッチコックの輸送問題を解くプログラムの仕様を書いている．その結果，VDM-SLは，それが持つ型と式の構成演算が適切で強力な表現力を持ち，文が実現に向けての設計仕様作成に役立つ等の極めて書きやすい言語で，簡潔で明瞭な仕様が作成できたとしている．

形式的ソフトウェア開発法の利点は一部の研究者の間では認められているが，実務に応用されてはいない．一方，産業界で比較的取り組みやすいソフトウェア開発法にJSDがある．JSDは形式的仕様記述言語を持たないため，より厳密なスタイルと支援系を与えるために，適切な仕様記述のための形式的な枠組みを調べた．その結果，JSDの仕様を形式的仕様記述言語で記述するには，RAISEが適切であることが解った．峰尾欽二の**JSD仕様のRAISE記述への変換**は，小さな課題を例として，JSDの仕様をRAISE仕様記述言語で記述し直し，本稿で提案する形式指向JSDが十分実用的であることを示している．

筆者らは，通信システム設計者でなくても，通信サービスを容易に記述できる形式的通信サービス仕様記述言語STRを提案してきた．STRでは通信サービスを広域状態遷移を表現するものと定義し，広域状態遷移をプロダクション制御の集合で記述する．STRのプロダクション規則は，グラフの書き換え規則とみなすことができる．STR記述を実現する通信ソフトウェアは，通信することによってグラフの同形判定を行う．河田慶三・田倉昭・太田理は，**宣言型通信サービス仕様記述言語からプロセス仕様を生成する手法**の中で，グラフの同形判定を行えるような通信ソフトウェアのモデルを提案し，STRからそのモデルに従った通信ソフトウェアを自動生成する手法について論じている．

近年の計算機に関する環境の飛躍的な発展により，プログラミング言語システムにおけるユーザインタフェースの系統的な記述方法論の確立が重要な研究課題となっている．「御結び」(OMS/B)は，直接操作型のインタフェースを構築するためのシステムであり，Common Lispのプログラミングにおけるオブジェクトブラウザとして有効である．また，その基本概念は他のプログラミング言語システムにおける視覚的ユーザインタフェース生成系を実現するための基盤技術である．溝上昌宏の**ユーザインタフェース生成系「御結び」については**，「御結び」の概要を説明し，実装における問題点の一つである表示の更新について述べている．

要求仕様書を理解し，プログラムとして実現すべき機能を明確にする作業は，重要な設計作業の一つである．しかし，設計作業に不慣れな者にとっては容易な作業ではない．そこで，筆者等は要求仕様に含まれている意味内容を段階的に整理していくためのモデル（自然語理解モデル）を提案し，モデルに基づく設計作業支援システムの開発を行ってきた．熊本厚志の**要求仕様理解モデルのプログラム設計への適用と評価**は，設計作業支援システムを大学の学生実験に適用することにより，モデルと支援システムの有効性を実験的に評価した結果について述べている．

教育内容の充実のためには，目標にそった授業の計画・実施・評価を行い，教育内容・指導方法を改善し，教材等の精練化を行う必要がある．これら教育カリキュラムを作成・評価する上で基礎となる情報を提供するものに，教授目標・教授目標ネットワークがある．しかし，教授目標ネットワークを構築・改善する方法はまだ確立されていない．内田修市は，**授業の開発——教授目標の抽出と構造化**の中で，教授目標ネットワークについて紹介し，教育カリキュラム全般に亘って教授目標ネットワークを活用する上での利点，ネットワークを構築・評価できるように教授目標を記述する上での考え方，教授目標の抽出方法，抽出された目標の構造化方法・評価方法等を述べている．

ウィーン開発技法 VDM

The Vienna Development Method ——A Degustation of VDM-SL

山 崎 利 治

要 約 ソフトウェア開発の形式的方法のひとつであるウィーン開発技法 VDM の仕様記述言語 VDM-SL の規格化が進行している．その紹介をかねてヒッチコックの輸送問題を解くプログラムの仕様を書いてみた．これはオペレーションズ・リサーチ関係のソフトウェアを手掛けているチームへ形式的方法の導入を企図して行った実験であったが，VDM-SL は極めて書きやすい言語で簡潔で明瞭な仕様が作成できたと考える．VDM-SL のもつ型と式の構成演算が適切で強力な表現力が得られ，また文が実現へ向けての設計仕様作成に役立つなどによるもので，大方の実用を期待したい．

Abstract The Vienna development method is one of the foremost formal methods for software development, and its specification language VDM-SL is now at the final stage of international standardization process. This report, derived from the author's degustation of VDM-SL, is intended as a VDM-SL guide for busy working programmers who have wanted to use VDM for their everyday activities.

A small experiment in writing formal specifications of the code to solve Hitchcock transportation problems has been conducted in the hope of formal methods being more widely accepted by people working with operations research software. It has turned out that VDM-SL's high descriptivity —— a result from its well-established concepts—— supported by a model-oriented approach, the implicit method of defining functions and operations, good selection of an adequate repertoire of type constructors and expression building operators, incorporated pattern-matching features and preparation of statements—— enables us to write a clear, precise specification.

The result of the experiment has made it clear that VDM-SL is already at the level of achieved maturity, awaiting our use for a wider variety of applications.

1. はじめに

ソフトウェアの形式的開発方法のひとつであるウィーン開発技法 VDM (Vienna Development Method) のうち，その仕様記述言語 VDM-SL を中心に紹介したい．それは VDM が Z とともに欧州の産業界での利用実績をもち，また，英国規格，さらには国際規格にしようとの動きもあるからである．

VDM の起源は 1960 年代初期に遡る．VDM の前身はウィーン定義言語 VDL (Vienna Definition Language) である．1965 年にその姿を現わした PL/I は，効率を重視し言語を単純にするよりも便宜を追求し特例の多い雑然とした大きな仕様のプログラム言語であった．このような言語の出現に当時の IBM ウィーン研究所の H. Zemanek を中心とするグループは，今後の計算機の安全な利用のためにはプログラム言語の厳密な意味定義が不可欠であると考えた．そして PL/I に対してその意味を形式的に記述し，この記述言語と記述方法が VDL である．VDL による意味定義は，

当時の技術の現状からプログラムを模倣する抽象機械を考え，プログラム言語の構文定義に対応する機械の状態変化を記述する方法であった．具体的にはつぎを与えたのである．

1) プログラムの字面を厳密に定めた具象構文
2) 意味記述のために字面の詳細を無視した抽象構文
3) 具象構文から抽象構文への変換方法
4) 抽象機械の状態定義
5) 抽象構文によるプログラムの実行に伴う機械の状態変化の規則

ほぼ完全なPL/Iの意味定義が1969年に報告され，操作的意味論の典型となった．改訂が頻繁に繰り返されるPL/Iに対して，言語設計者も処理系作成者も実際にこの定義を参照し議論の基礎となったというから，その意味ではこの試みは成功したといえた．しかし，言語の意味を形式的に推論しようとすると，この抽象機械では不必要な煩雑さをもたらすものでもあった．新計算機とそのPL/Iコンパイラの開発計画があったから，このVDLを見直すことになる．VDMの萌芽である．

プログラムの意味を数学的に集合・関数・関係などによって与える考えもあった．つまり，構文要素から構成する構文領域，関数や関係から構成する意味領域と前者から後者への写像である意味関数を与えることによって意味を定義しようというのである．これを表示的意味論というが1960年代ではこの考えにまだ技術的問題が残っていた．1970年にD. Scottがこの問題を克服しその基礎を整備したのでVDLグループもこの方法へと転換を企てる．実際にAlgol 60やPL/Iを再定義しこの実用性を確認する．ここで使用した記述言語META-IVと表示的意味記述方法をVDMというようになったのである．ところが新計算機計画が中止されVDMグループはウィーンを離れることとなるが，1973年から78年にかけてのこの時期をVDMの古典ウィーン期と言ってよい[2][3]．

ウィーンを離れたVDMは英国とデンマークで活躍する．英国ではC. Jonesが産業界への普及を精力的に図る．英国流VDMは抽象データ型を定義し段階的に詳細化することを教える．ここでの抽象データ型の定義はモデル指向・論理型で，定義対象を具体的に直接構成表現し，それらの上の関数や演算を作用前の条件と作用後の条件とを述べる間接的な手段によって行う．詳細化に際しては，その正しさを必ず証明しなければならないという義務を負わせる[11][12][13]．デンマーク流VDMの代表はD. Bjørnerである．ここではプログラム言語の仕様記述とコンパイラの設計を盛んに行う[4]．抽象機械を表示的意味記述に変えたもののVDL以来の伝統に忠実で，具象構文，抽象構文，前者から後者への変換，静的意味論（文脈条件），動的意味論（意味関数）などを与える定義方法を確立している．英国流のモデル指向・間接的論理型VDMに対してデンマーク流は，直接的作用型VDMであった．この時期，1979年から86年は英国とデンマークというVDMの併立期であった．

VDMが産業界に普及するにつれ，そのVDMに対してさまざまな希望や要求が表面化してくる．英国流とデンマーク流の記法を統一したい，大きな仕様のためにモジュール化機構を用意したい，詳細化に伴う検証用論理体系を設定したい，特殊な記号を使うのでエディタなどのソフトウェア・ツールがほしい，教育資料や教育プログラ

ムを充実したい，などである．1987年には欧州共同体委員会が後援してVDM利用者団体であるVDM欧州グループが結成され，年3，4回の会議や一週間におよぶ研究集会をもつようになる．実際そこが，VDMに関する経験，教育，ツール，方法論，数学的基礎，規格化などの情報交換や討議の場になった[5][6][7][15]．VDM会議はやがてVDMにとどまらず，Zやそのほかの形式的開発方法も検討するよう拡大することになる．

現在のVDMに関する最大の話題は，その仕様記述言語VDM-SLの規格化である．産業界からの要望が英国標準局（BSI）にVDM規格化パネルをつくらせ，VDM方言の整理統合，構造化機構（モジュール）の設定，関数と演算に多様型の導入などを狙いとして規格化を図る．当初は英国規格として87年中に原案作成，88年夏に規格成立と考えたが，VDM欧州グループの誕生とともに国際規格化を企図し，歩みは遅れることになる．規格案は，デンマーク，オランダ，ポーランド，英国の分担作業で，素朴表示意味論の枠組で超循環定義，つまりVDM-SL自体で記述している．

2. VDM-SLのあらまし

現在進行中の規格案に従ってVDM-SLのあらましを述べる．VDM-SLは仕様記述とともに実現のための設計記述にも利用できる．

VDM-SLによる仕様は文書（document）と言い，型，値，状態，関数，演算を定義した区画から構成する．

型定義区画は仕様記述に必要な値の型をまとめて定義するためのものである．型とは値の集合とそれに許す演算の組に対する名前である．表1に示す組込型を用意している．

表1 組込型
Table 1 Built-in types

記号	値	説明
B	true,false	
N	0,1,2,…	
$\mathbf{N}_1$	1,2,3,…	
Z	整数	
Q	有理数	
R	実数	
char	VDM-SL文字	
token	トークン	異なる無限個の値をもつ．
quote	クォート	枚挙型の値として用いる．

表2 型構成演算子
Table 2 Type constructors

型構成子	優先順位	結合性	説明
T-*Set*	4	—	T型値の有限集合型
$T \xrightarrow{m} T'$	3	右	T型からT'型への有限写像型
$T \xrightarrow{m} T'$	3	右	1対1写像型
T^*	4	—	空列を含む有限列型
T^+	4	—	空列を含まない有限列型
$I :: I_1 : T_1$ ⋮ $I_n : T_n$	—	—	$T_1,\cdots,T_n$型から構成したレコード型
$T \times T'$	3	—	直積型
T|T	2	左	連合型（合併型）
〔T〕	—	—	選択型（T型の値かnil）
$T \rightarrow T'$	1	右	関数型
$() \rightarrow T$	1	右	関数型（引数なし）
$()\|T \xrightarrow{o} ()\|T'$	1	右	演算型

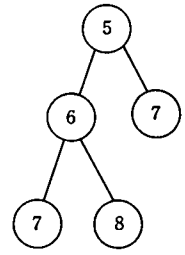
また，表2に示す型構成子をもち，これによって新しい型を構成できる．一般に再帰的な型構成ができる．また，型定義に不変式を付けて部分型も定義できる．たとえば，つぎは再帰的定義の例で以下に説明する型を定義している．

types

Bt = [Bt]×**N**×[Bt]

inv mk-(lt, n, rt) ≜lt=*nil*⇔rt=*nil*

N は組込型で 0 を含む自然数型である．[Bt] はいま定義している型 Bt か，あるいは *nil* である．*nil* は存在しないを意味する型の値の表示である．型 Bt は，したがって [Bt]×**N**×[Bt] という直積型である．*inv* 以下は，この型の値について要請する不変式で，第 1 要素 lt が *nil* なら，第 3 要素 rt も *nil* で，その逆も成立しなければならないと主張している．つまり Bt は節に自然数が付随した 2 分木で，ある節の左右の子が同時に存在するか，あるいは同時に存在しないようなものを値とする型を定義している．



bt：Bt…Binary tree

図 1

値定義区画は仕様記述に必要な定数を定義する．つぎはこの一例である．

values

π：**R**=3.1416；

ε：**R**=0.0003；

bt：Bt=*mk*-(*mk*-(*mk*-(*nil*, 7, *nil*), 6,

mk-(*nil*, 8, *nil*)), 5, *mk*-(*nil*, 7, *nil*))

第 1 の定義は π は **R** 型（実数型）の値で，3.1416 である定数であることを定義している．VDM-SL ではギリシャ文字が使える．第 3 の定義は図 1 に示す 2 分木を定数として定義している．*mk*-(−, −, −) は三つの型の値から構成する直積型の値を構成する．

状態定義は状態を定義する．状態は命令型プログラム言語でいう大域変数に相当し，演算による累積効果を利用するための装置である．演算定義区画で定義した演算がこの状態を参照，更新する．初期値が設定でき不変式も書ける．つぎはこの定義の一例である．

state Tree *of*

bt：Bt

init bt≜*mk*-(*nil*, 0, *nil*)

end

関数定義区画は仕様の中で参照する関数をまとめて定義する．ここで関数とは，いくつかの値の順序をもった組（引数）に対してひとつの値（結果）の対応付けを言う．結果を直接に式によって示す直接的な関数定義と，引数が満たすべき条件（前件）と

結果が満たすべき条件（後件）とを与える間接的な定義がある．関数は副作用をもたない．つまり，状態を参照することはできない．高階関数（カリー化）や多様型の関数も定義できる．

つぎは関数定義の例である．

functions
sq : **R**→**R**
sq (x) ≜ x×x ;
sqrt (x : **R**) y : **R**
pre x≥0
post *abs*(x−y×y)<ε ;
Σ[@A] : @A-*set*×(@A$^{m}_{\to}$**R**)→**R**
Σ(a, f)≜ *if* a={} *then* 0
else *let* e∈a *in* f(e)+Σ(a−{e}, f)

関数 sq は直接的定義の例で，その型が **R**→**R** で，x に対して引数 x×x を結果とすると定義している．

関数 sqrt は間接的定義の例で，引数 x は実数型の値，結果 y はやはり実数型の値である．*pre* …は前件であり，x≥0 であれば，この関数が適用でき，結果が y として得られ，*post* …を満たすと主張している．この場合 |x−y×y|<ε，つまり，y は x の平方根を近似する．

関数 Σ は多様型の関数を定義している．VDM-SL の場合，型の変化する部分を型変数を使って定義し，関数適用時に型を型変数に与えて具体化する方式（parametric polymorphism）である．型変数は@T の形で書く．関数 Σ の場合，@A がこの型変数で，これを媒介して定義している．引数の型は @A-*set*×@A$^{m}_{\to}$**R** であり，結果は **R** 型である．これは @A 型の値の有限集合 a と @A 上で定義し値域を **R** とする有限写像 f に対し，a 上の f の値の和 $\sum_{e\in a}f(e)$ を対応させる関数の定義である．*if* 式と *let be* 式を利用しているが，これについては式のところで説明する．

演算定義区画は状態を参照しまた更新できる演算をまとめて定義する．式を主体とした前件と後件による間接的定義と，文を主体とした直接定義の二つの定義方法がある．例はあとで示す．

式について一言する．式は値を表示する．式はプログラム言語で馴染みであるから多言は無用だろう．ここでは，*if* 式，*case* 式，*let* 式，*let be* 式とパターン照合について注意をする．

if 式はつぎの形をしているが意味は自明だろう．[…] はあってもなくてもよい．

if B_1 *then* E_1
[*else if* B_2 *then* E_2
…
else if B_n *then* E_n]
else E

cases 式はつぎの形をしている．

cases E：
$P_1^1, P_1^2, \cdots \rightarrow E_1,$
…
$P_n^1, P_n^2, \cdots \rightarrow E_n,$
[*others*→E′]
end

この意味は「式Eの値がちょうどパターンP_k^lに一致すれば，E_kの値となり，どのパターンとも一致しなければ未定義となる．ただし，*others*→E′があればE′の値となる」である．ここでパターンについて説明しなければならない．

パターンはある型の値，あるいは，その値の一部が定まっていないとき，その値を決定するための型板となるものである．つまり，未束縛変数をパターン照合によって束縛しようというもので，関数型や論理型のプログラム言語で普及している機構である．パターンとしては，名前，－，(式)，リテラル，p_i, p, p′をパターンとして，$\{p_1, \cdots, p_n\}$, $p \cup p'$, $[p_1, \cdots, p_n]$, $p \frown p'$, *mk*-$(p_1, \cdots, p_n)$, Id$(p_1, \cdots, p_n)$などがある．たとえば，p_1, p_2を$\mathbf{R}\times\mathbf{R}$型の値として，

let *mk*-$(x_1, y_1)=p_1$,
mk-$(x_2, y_2)=p_2$ *in*
sqrt(sq(x_1-x_2)+sq(y_1-y_2))

は，2次元ユークリッド空間における点p_1とp_2との間のユークリッド距離を与える式となる．つまり，*mk*-$(x_1, y_1)=p_1$によって，未束縛変数x_1, y_1がp_1と照合することによって束縛され，x_1, y_1の値が定まるわけである．x_2, y_2についても同様である．

Σ：**N**-set×(**N**→**N**)→**N**
$\Sigma(a, f) \triangleq$ *cases* a：
{}→0,
{e}∪b→f(e)+Σ(b,f)
end

上の例は**N**-*set*型のaの場合分けによる関数Σの定義である．aが{}に一致したとき，つまり，$\Sigma(\{\}, f)$は0であり，aが{e}∪bに一致したとき，$\Sigma(\{e\}\cup b, f)=f(e)+\Sigma(b, f)$であると定義している．パターンは上のように，

1) 値による場合分け
2) 注目対象である部分の値を選択する．
3) 選択抽出した成分値への演算
4) 複雑な型の値の構成

などに利用する．

let 式は上に挙げたように，
let 値定義，関数定義，… *in* 式
の形で用法は自明だろう．

let *be* 式はつぎの形をしている．
let p∈集合 [*be* *st* 式1] *in* 式2
let p：型 [*be* *st* 式1] *in* 式2

これは，式2において出現するpは集合の要素（あるいは，型の要素）で（もし書いてあれば）式1を満たすものである．つまり，そのようなpとして，式2を評価するわけである．式についてその書き方を表3，4，5，6にまとめておく．

文については省略する．

表3 集合体

Table 3 Set type notations

記 法	説 明
{}	空集合
$\{e_1,\cdots,e_n\}$	$e_1,\cdots,e_n$を要素とする集合
{e | p : T, p′∈S, …•e′}	pをT型の値，p′をSの要素，…としてe′を満たすようなeのつくる集合
S ∪ S′	SとS′の合併
S ∩ S′	SとS′との共通部分
e∈S	eはSの要素である．T×T-*set*→**B**
e∉S	eはSの要素ではない．
S⊂S′	SはS′の真部分集合である．T-*set*×T-*set*→**B**
S⊆S	SはS′の部分集合である．
card S	Sの濃度．T-*set*→**N**

表4 有限列型

Table 4 Finite sequence type notations

記 法	説 明
[]	空列
$[e_1,\cdots,e_n]$	$e_1,\cdots,e_n$の順の有限列
[e | i∈S•e′]	Sを順序集合とし，その要素をiとしe′を満たすeの列（Sの順）
hd l	有限列lの先頭要素．l=[]なら未定義
tl l	有限列lの先頭要素を除いた列 空列なら未定義
len l	lの長さ，空列は0
elems l	lの要素の集合．
inds l	l=[]なら空集合，そうでない時は{1,…, *len* l}
l⌢l′	有限列lとl′との連接
l(i)	有限列lのi番目の要素

3. 例題——ヒッチコック問題

VDM-SLの味見としてヒッチコックの輸送問題を飛び石法（MODI法）によって解くプログラムを考える．この例題を選んだ理由は，オペレーションズ・リサーチ分野のソフトウェアを手掛けているチームに形式的開発方法の導入を計画して行った実験のひとつがこの問題であったからである．問題はつぎである．

「ある物資の生産地と消費地がそれぞれ複数個所あり，そこでの生産量と消費量が定まっている．どの生産地からもすべての消費地へ輸送路が通じていて，それ

表5　有限写像型
Table 5　Finite map notations

記　法	説　明
$\{\mapsto\}$	空写像
$\{e_1 \mapsto e_1', \cdots, en \mapsto en'\}$	定義域を $\{e_1, \cdots, e_n\}$, 値域を $\{e_1', \cdots, e_n'\}$ とする有限写像 ei に ei′ を対応させる.
$\{e \mapsto e' \mid p : T, p' \in S, \cdots \bullet e''\}$	P を 型Tの値, p′ を集合 S の要素, …とする時の e を e′ に対応させる有限写像, ただし, e, e′ は e″ を満たす.
dom m	有限写像mの定義域
rng m	有限写像mの値域
m(e)	mを有限写像, e ∈ *dom* mとするときmによる e の値
m†m′	m, m′ を T→T′ 型の有限写像とする. $m \dagger m' \triangleq \{e \mapsto e' \mid e : T, e' : T' \bullet$ $[e \in dom\ m' \Rightarrow e' = m'(e)]$ $\wedge [e \in dom\ m \wedge e \notin dom\ m' \Rightarrow e' = m(e)]\}$

表6　その他
Table 6　Notations for other types

記　法	説　明
$\forall p : T, p' \in S, \cdots \bullet e$	型 T のすべての値 p, 集合 S のすべての要素 p′, …が e を満たす.
$\exists p : T, p' \in S, \cdots \bullet e$	e を満たす型 T の値 p, 集合 S の要素 p′, …が存在する.
$\exists ! p \in S \bullet e$	e を満たす集合 S の要素 p が一意的に存在する.
$\exists ! p : T \bullet e$	e を満たす型 T の値 p が一意的に存在する.
$\iota p \in S \bullet e$	e を満たす集合 S の一意的な要素 p
$\iota p : T \bullet e$	e を満たす型 T の一意的な値 p
$\lambda p : T, p' \in S, \cdots \bullet e$	型 T の値 p, 集合 S の要素 p′, …を変数とする関数 e

ぞれの路には単位量当たりの物資輸送費が定まっている. 消費地での消費を満たすように生産地から消費地へ物資を送りたい. 総生産量と総消費量が等しいと仮定して, 総輸送費を最小にするためには, どの路にいくらの量を送ればよいか? もちろん, どの生産地からもその生産量を超える物資を送り出すことはできない.」

線型計画法の教科書ではこの問題をつぎのように定式化している.

「生産地, 消費地の集合をそれぞれ S. D とする. 生産地 i の生産量を s_i, 消費地 j の消費量を d_j, i と j を結ぶ路 (i, j) の輸送単価を c_{ij}, またその輸送量を x_{ij} とする. 以下の⓪, ①, ②, ③という制約条件のもとで, 目的関数 (objective function):

$$\sum_{(i,j) \in S \times D} c_{ij} \cdot x_{ij}$$

を最小にする x_{ij} を求めよ．

⓪　$\sum_i s_i = \sum_j d_j$（前提条件），

①　すべての $i \in S$ に対して，$\sum_j x_{ij} = s_i$，

②　すべての $j \in D$ に対して，$\sum_i x_{ij} = d_j$，

③　すべての $(i, j) \in S \times D$ に対して，$x_{ij} \geq 0$

条件①，②，③を満たす (x_{ij}) を可能解（feasible solution）と言い，可能解で目的関数を最小にする (x_{ij}) を最適解（optimal solution）と言う．」

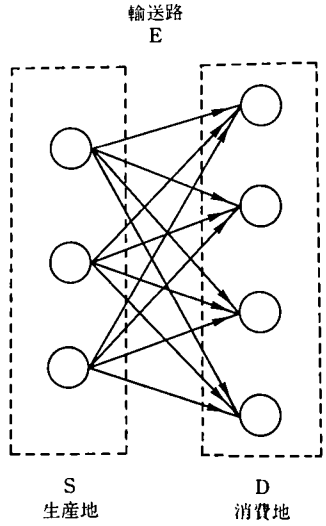


図 2

```
types
  St, Dt = token-set;
  Nt = St | Dt
       inv n ≜ is-St (n) ⇔ ¬ is-Dt (n);
  Et :: sup : St
        dem : Dt;
  B = Nt →m R
      inv b ≜ ∀n : Nt • b (n) > 0
            ∧ Σ [Nt] (S, b) = Σ [Nt] (D, b);
  Cost, X = Et →m R
```

図 3

以上を VDM-SL で記述する．VDM では問題を集合，関数，関係によって述べるから，まず必要な集合，つまり型を定義しなければならない．

図 3 がそれである．

生産地型 St と消費地型 Dt とを枚挙型と考えて定義する．具体的には生産地や消費地の名が与えられるのであろうが，ここでは，これ以上の詳細は述べないとして **token**-*set* とする．**token** は VDM の組込型で，無限個の相異なる値をもつ型で，その上に許す演算は，＝と ≠ だけである．あとで詳細を与える時などに使う型である．St と Dt の連合型を用意すると便利だから，Nt＝St|Dt と定義する．Nt の定義に，型 St の値集合と型 Dt の値集合が共通部分をもたないことを不変式として付加しておく．輸送路は生産地と消費地との順序対と考えなければよいが，ここではレコード型として定義する．すなわち，

Et :: sup : St
　　dem : Dt

型 Et はそれぞれ St と Dt 型の sup と dem という名をもつフィールドから構成したレコード型で，Et 型の値 e に対して，e. sup や e. dem は e の成分の値を表示し，また，s や d を St，Dt の値とすれば Et(s, d) は Et の値（レコード）を表示する．

さて，生産量と消費量はいまやNtから実数型**R**への有限写像として定義できる．つまり，B=Nt$\xrightarrow{m}$**R**である．生産量と消費量は正実数であるから，また，それらの値が等しいという仮定から，型Bに不変式として，これを付記する．それがつぎである．

$$inv\ b \triangleq \forall n : Nt \cdot b(n) > 0 \wedge \Sigma[Nt](S, b) = \Sigma[Nt](D, b)$$

連言の左辺は自明だろう．右辺は関数定義区画で定義する筈の関数Σを適用している．Σは多様型関数で，この適用に際して型変数に対して，型Ntを与えて具体化したものである．$\Sigma[Nt](S, b)$は$\Sigma_{s \in S} b(s)$のつもりである．ここにS, Dは型St, Dtの値の集合のつもりで，値定義区画で定義する筈の値である．

輸送路上の輸送単価，物資の割り当て量の型をCost, Xとすれば，ともにEtから**R**への有限写像型と考えればよく，Cost, X=Et$\xrightarrow{m}$**R**と定義できる．型定義は以上でよい．

つぎは値定義である．図4がそれである．

values

S : St-*set* ={s | s : St} ;
D : Dt-*set* ={d | d : Dt} ;
N : Nt-*set* ={n | n : Nt} ;
E : Et-*set* ={e | e : Et}

図 4

これらは型St, Dt, Nt, Etの値集合S, D, N, Eを定義したものである．これは単にそれらを参照したいためにすぎない．

つぎは関数定義である．仕様作成の目的はB型の値bとCost型の値cを与えて，X型の最適解xを求めることである．この関数optsolは間接的定義ですぐに書ける．

optsol (b : B, c : Cost) x : X
post is-optimal (b, c, x)

is-optimalはB×Cost×X→**B**型の関数として定義することにしたが，むろんこの関数定義を直接ここ，*post* …に書いてよい．しかし，随所で引用するだろうと考えて(実はあまりそうでもないが)，べつに定義することにしたのである．is-optimalとしては，xが可能解であることと，任意の可能解yに対して，xによる目的関数値がyのそれより大きくはない，つまり，xによる目的関数値が最小であることを言えばよいわけである．

関数is-feasibleはB×X→**B**型のそれで，xがbに関して可能解であることを主張するものである．「つもり」がわかれば図5の関数定義区画は見やすいだろう．

functions

optsol (b : B , c : Cost) x : X
post is-optimal (b , c , x) ;

is-optimal : B×Cost×X→**B**
is-optimal (b , c , x) ≙ is-feasible (b , x)
  ∧ ∀y : X • is-feasible (b , y) ⇒ obj (c , x) ≤ obj (c , y) ;

is-feasible : B×X→**B**
is-feasible (b , x) ≙ ∀e : Et • (e) ≥ 0

$\wedge \forall s : St \cdot \sum[Et](\{e \mid e \in E \cdot e.sup=s\}, x) = b(s)$
$\wedge \forall d : Dt \cdot \sum[Et](\{e \mid e \in E \cdot e.dem=d\}, x) = b(d)$;

$obj : Cost \times X \to \mathbf{R}$
$obj(c, x) \triangleq z[Et](E, c, x)$;
$z[@A] : @A\text{-}set \times (@A \xrightarrow{m} \mathbf{R}) \times (@\mathbf{A} \xrightarrow{m} \mathbf{R}) \to \mathbf{R}$
$z(a, f, g) \triangleq$ *if* $a = \{\}$ *then* 0
else let $e \in a$ *in* $f(e) \times g(e) + z(a - \{e\}, f, g)$;
$\sum[@A] : @A\text{-}set \times (@A \xrightarrow{m} \mathbf{R}) \to \mathbf{R}$
$\sum(a, f) \triangleq$ *if* $a = \{\}$ *then* 0
else let $e \in a$ *in* $f(e) + \sum(a - \{e\}, f)$

図 5

以上で，とりあえず問題定義は完成した．さて，つぎはこの仕様の実現を目指して詳細化を図る．関数 optsol を洗練するわけであるが，MODI 法をすでに知っており[10]，それによる実現を考えたもので，状態を定義し，VDM のいう演算を一部直接的に定義することにしたい．

とりあえず，関数 optsol を演算として書いてみよう．状態をまず定義する．

state Hp0 *of*
x : X
end

演算 OPTSOL はそこで，

OPTSOL(b : B, c : Cost)
ext wr x : X
post is-optimal(b, c, x)

と書ける．*ext wr* x : X は状態の x : X を参照・更新することを意味する．参照するだけなら *rd* と書く．*ext* は externals, *wr* は read and WRite, *rd* は ReaD のつもりである．前件 *pre* …は *pre* true なら省略してもよい(b に対する条件は型 B の定義に含めた)．

演算 OPTSOL はつぎのように分解できる．

(IBFS(b) ;
OPT(b. c))

これは基底可能解を求める IBFS(b : B) と最適解を求める OPT(b : B, c : Cost) を参照する演算呼出し文の連接文である．OPT をさらに分解するために，状態を再定義する．そのためには型も新しく定義しなければならない．基底可能解を構成する輸送路の集合を参照するために型 Btree を定義する．基底木である．

Btree = Et-*set*
inv T ≜ is-basistree(T)

Btree の値は，Et の値の集合，つまり E の部分集合 T で，is-basistree(T) を満たすものである．is-basistree(T) は T がグラフ (E, N) で全域木を構成することを意味する．言い換えれば，T は連結で閉路がなく，T は N のすべての節 (頂点) を含むこ

と，すなわち，

$$\text{is-conn}(T) \wedge \text{is-acyclic}(T) \wedge \text{is-span}(T)$$

である．

基底可能解が最適解であることを判定するために輸送路に付随する相対費用が必要であるから，型 Rcost を定義する．

$$\text{Rcost} = \text{Et} \overset{m}{\to} \mathbf{R}$$

この計算のために，節の上で定義される単体係数(ポテンシャル)も必要である．

$$\text{Pot} = \text{Nt} \overset{m}{\to} \mathbf{R}$$

基底木に新しく入れる輸送路が基底木とともにつくる閉路を表す型 Cycle を用意する．

$$\text{Cycle} = \text{Et}^{+}$$

inv l≙ is-mincyclic(*hd* l, l)

型 Cycle の値は Et の値がつくる有限列で，*inv* …を満たすものである．*hd* は有限列型に許す演算で，列の先頭要素をとる．is-mincyclic は，*hd* l から始まる Et の値の列が一番短い閉路をつくることを主張するものである．

型が用意できたから状態を再定義しよう．

state Hpl *of*
x : X
T : Btree
v : Pot
r : Rcost
l : Cycle
ee, le : Et
end

図 6

ee，le : Et は基底に入れる路を出る路である．

演算 OPTSOL を二つの演算に分解したが，それをもう一度書いてみよう．

IBFS(b : B)
ext *wr* x : X
wr T : Btree
post is-bf(b, x, T)；

OPT(b : B, c : Cost)
ext *wr* x : X
wr T : Btree
pre is-bf(b, x, T)
post is-optimal(b, c, x)

ここで，is-bf(b, x, T) は b に関して x が可能解かつ，x, T が基底解となっているという主張である．

演算 OPT をさらに分解する．図 7 のように分解できる．ここでは状態の参照が直接目に見えないが，意味の判読はやさしいだろう．is-opt は相対費用最小の値が非負なら，その解が最適であることを示す関数である．

```
OPT : B×Cost ~→ ()
——pre is-bf (b,x,T)
OPT (b,c) ≜
while true do
  ( POTENTIAL (b,c) ;
    REDUCEDCOST (b,c) ;
    ENTERINGEDGE (b,c) ;
    if is-opt (r,ee) then exit;
    LEAVINGEDGE (b,c)
    UPDATE (b,c)
  )
post is-optimal (b,c,x)
```

図 7

図 7 に現われる演算それぞれを間接的に定義したものが図 8，図 9，図 10，図 11，図 12 である．

```
POTENTIAL (b : B , c : Cost)
ext rd  x : X
    rd  T : Btree
    wr  v : Pot
pre   is-bf (b,x,T)
post  is-pot (c,T,v)
    ∧ is-bf (b,x,T)
```

図 8

```
REDUCEDCOST (b : B,c : Cost)
ext rd  x : X
    rd  T : Btree
    rd  v : Pot
    wr  r : Rcost
pre   is-pot (c,T,v) ∧ is-bf (b,x,T)
post  is-rcost (c,T,v,r)
    ∧ is-pot (c,T,v) ∧ is-bf (b,x,T)
```

図 9

```
ENTERINGEDGE (b : B , c : Cost)
ext rd  x : X
    rd  T : Btree
    rd  v : Pot
    rd  r : Rcost
    wr ee : Et
pre   is-rcost (c,T,v,r) ∧ is-pot (c,T,v)
    ∧ is-bf (b,x,T)
post  is-enter (T,r,ee)
    ∧ is-rcost (c,T,v,r) ∧is-pot (c,T,v)
    ∧ is-bf (b,x,T)
```

図 10

```
LEAVINGEDGE (b : B , c : Cost)
ext rd  x : X
    rd  T : Btree
    rd  v : Pot
    rd  r : Rcost
    rd ee : Et
    wr  l : Cycle
    wr le : Et
pre  ¬ is-opt (r,ee) ∧ is-enter (T,r,ee)
     ∧ is-rcost (c,T,v,r) ∧ is-pot (c,T,v)
     ∧ is-bf (b,x,T)
post   is-leave (x,T,ee,l,le)
 ∧ ¬ is-opt (r,ee) ∧ is-enter (T,r,ee)
     ∧ is-rcost (c,T,v,r) ∧ is-pot (c,T,v)
     ∧ is-bf (b,x,T)
```

図 11

UPDATE (b : B , c : Cost)
ext *wr* x : X
wr T : Btree
rd v : Pot
rd r : Rcost
rd ee.le : E
rd l : Cycle
pre is-leave (x, T, ee, l, le) ∧ ¬ is-opt (r, ee)
∧ is-enter (T, r, ee) ∧ is-rcost (c, T, v, r)
∧ is-pot (c, T, v) ∧ is-bf (b, x, T)
post is-update (ee, le, l, $\overrightarrow{T}$, $\overleftarrow{x}$, T, x)
∧ is-bf (b, x, T)

図 12

間接的に定義した前件と後件のうちグラフに関係するものは，このグラフが図2のような特別の形(完全2部グラフ)であることを利用している．図13がそれらである．

図14がMODI法に関係する関数定義である．ほとんど自明と思えるが，二三説明を補足する．

```
is-basistree : Et-set → B
is-basistree (T) ≜ is-conn (T) ∧ is-acyclic (T) ∧ is-span (T) ;

is-conn : Et-set → B
is-conn (T) ≜ ∀m , n ∈ nset (T) · ∃ p ∈ sts (T) ·
      is-path (p) ∧ m ∈ nseq (p) ∧ n ∈ nseq (p);

is-acyclic : Et-set → B
is-acyclic (T) ≜ ∀ p ∈ sts (T) · is-path (p) ⇒ ¬ is-cyclic (hd p , p);

is-span : Et-set → B
is-span (T) ≜ N = nset (T);

nset : Et-set → Nt-set
nset (T) ≜ {e·sup | e ∈ T} ∪ {e·dem | e ∈ T};

nseq : Et+ → Nt-set
nseq (l) ≜ {l (i).sup | i ∈ inds l} ∪ {l (i).dem | i ∈ inds l};

sts : Et-set → Et*-set
sts (T) ≜ {l | l : Et* · elems l ⊆ T};

is-cyclic : Et × Et+ → B
is-cyclic (e , l) ≜ is-path (l) ∧ l (len l). sup = e.sup ;

is-path : Et+ → B
is-path (p) ≜
  [∀i ∈ inds p-{len p}·
      (odd (i) ⇒ p (i). sup = p (i+1). sup)∧ (even (i) ⇒ p (i). dem = p (i+1). dem)]
∨[∀i ∈ inds p-{len p}·
      (odd (i) ⇒ p (i). dem = p (i+1). dem)∧ (even (i) ⇒ p (i). sup = p (i+1). sup)];
```

```
odd : N → B
odd (x) ≜ x mod2 = 1 ;

even : N → B
even (x) ≜ x mod2 = 0 ;

is-mincyclic : Et × Et+ → B
is-mincyclic (e , l) ≜ is-cyclic (e , l)
      ∧ ∀ ll : Et+ · len ll < len l ⇒ ¬ is-cyclic (e , ll)
```

図 13

```
is-bf : B × X × Et-set → B
is-bf (b , x , T) ≜ is-feasible (b , x) ∧ is-basistree (T) ∧ ∀e ∈ E−T · x (e) = 0 ;

is-pot : Cost × Btree × Pot → B
is-pot (c , T , v) ≜ ∀e ∈ T · v (e . dem) − v (e . sup) = c (e) ;

is-rcost : Cost × Btree × Pot × Rcost → B
is-rcost (c , T , v , r) ≜ ∀e ∈ E − T · r (e) = c (e) + v (e . sup) − v (e . dem) ;

is-enter : Btree × Rcost × Et → B
is-enter (T , r , ee) ≜ ∀e ∈ E − T · r (ee) ≤ r (e) ;

is-opt : Rcost × Et → B
is-opt (r , e) ≜ r (e) ≥ 0 ;

is-leave : X × Btree × Et × Cycle × Et → B
is-leave (x , T , ee , l , le) ≜
    let l ∈ sts (T ∪ {ee}) be st is-mincyclic (ee , l) in
    let ll = [l (i) | i ∈ inds l · even (i)] in
    le ∈ elems ll ∧ ∀ i ∈ inds ll · x (le) ≤ x (ll (i)) ;

is-update : Et × Et × Cycle × Btree × X × Btree × X → B
is-update (ee , le , l , oT , ox , T , x) ≜
    T = oT ∪ {ee} − {le}
  ∧ let δ = ox (le) in
    x = ox † {l (i) ↦ if odd (i) then ox (l (i)) + δ else ox (l (i)) − δ | i ∈ inds l}
```

図 14

1) 有限列演算

hd：T+→T：有限列の先頭要素

len：T*→**N**：有限列の長さ

inds：T*→**N**-set：有限列を自然数区間から T 型要素への写像とみなした時の定義域 [1…*len* l] (l が空列なら *inds* l={})，この *inds* の要素によって列要素が参照できる (l(5) は列 l の先頭から 5 番目の要素を指す.

elems：T*→T-set：有限列要素の集合

2) 演算の後件の記述中に出現する名前の上の←は，演算実行前の値を参照する．(例 is-update(ee, le, l, $\overleftarrow{T}$, $\overleftarrow{x}$, l, x) の中の $\overleftarrow{T}$, $\overleftarrow{x}$ は UPDATE の実行前の T, x の値を表示する．)

仕様の実現へ向かう段階で VDM-SL の文を使うが，その一例として演算 IBFS を図 15 に示す．この手続きは極めて単純だから，文の説明がなくても読めるだろう．

```
IBFS : B ⟕ ()
IBFS (b) ≜
( dcl P : St-Set := S ;
  dcl Q : Dt-Set := D ;
  dcl bb : B := b ;
  dcl TT : Et-Set := {} ;
  dcl xx : X := {e↦0 | e : Et} ;
  ( while p ≠ {} ∨Q ≠ {} do
  ( let p ∈ P in
    let q ∈ Q in
    def e = Et (p , q) in
    ( TT := TT ∪ {e} ;
      ( bb (p) ≤ bb (q) → (xx := xx† {e↦bb (p)} ;
                           bb (q) := bb (q) − bb (p) ;
                           P := P − {p}),
        bb (p) > bb (q) → (xx := xx† {e↦bb (q)} ;
                           bb (p) := bb (p) − bb (q) ;
                           Q := Q − {q}))))) ;
x := xx ;
T := TT)
−−is-bf (b , x , T)
```

図 15

4. おわりに

VDM-SL を管見してヒッチコック問題の仕様を書いてみた．VDM-SL の味見が主目的であって，ヒッチコック問題のコード作成を必ずしも考えたわけではなかったが，このような記述がコード作成のためにも出発点になるだろう．VDM-SL は表現力豊かな書きやすい言語である．それは型構成演算が適切で，さらに式を構成する演算も豊富であることに由来するが，モデル指向という直接に状態を構成完義できることも大きい．細かく見ればパターン照合がいたるところで使用できることも書きやすさの一因である．段階的詳細化のためには関数や演算に対して前件と後件の組による間接的定義や文による詳細化機構も仕様作成や設計記述を大いに楽にしている．形式的記述は，課題や算法を理解しているつもりでも，その理解を徹底的に験めされるものである．チーム作業を行う場合，要員の技術水準を一定にする努力が重要であるが，形式的記述によって試めすことも一法となる．形式的に書いた関数や演算はまた，簡単に関数型や論理型のプログラム言語，たとえば，ML や Miranda や Prolog によって

直ちに書き直すことができる．小さな規模の具体的な問題をそこで解くことができ，チーム作業にとってよい実験ができるようになることも形式的記述の利点である．

VDM-SL による記述は，それを作業チームが共有できれば極めて有効である．というのもそれぞれのチームが特有の記法を用いていることの多い現状であれば，記法にまで言及しないで済む効率の大きさは特筆すべきものといえる．

VDM とそのほかの形式的開発方法，たとえば，Z や RAISE との差についても述べるべきであるが，同じ発想による記述では大きな差を感じない．形式的開発方法とともに開発されたツールについても報告しなければならないが，未使用で今後に報告の機会を得たいと考えている．

VDM-SL の規格案もなお検討中であり，変更の可能性もある．しかし，ほんの僅かな記述実験によっても VDM の十分な成熟度を実感した．大方の興味と実用を期待したい．

参考文献

[1] D. Andrews, Report on the Standardization of VDM-SL 04/08/92 Ref. N-242.
[2] D. Bjørner and C. B. Jones(eds.), The Vienna Development Method. LNCS 61, Springer-Verlag, 1978.
[3] D. Bjørner and C. B. Jones, Formal Specification and Software Development. Prentice-Hall, 1982.
[4] D. Bjørner and O. N. Oest, Towards a Formal Description of Ada. LNCS 98, Springer-Verlag, 1980.
[5] D. Bjørner et al.(eds.), VDM'87 VDM-A Formal Method at Work. LNCS 252, Springer-Verlag, 1987.
[6] D. Bjørner et al.(eds.), VDM'90 VDM and Z-Formal Methods in Software Development. LNCS 428, Springer-Verlag, 1990.
[7] R. Bloomfield et al.(eds.), VDM'88 VDM-The Way Ahead. LNCS 328, Springer-Verlag, 1988.
[8] J. Dawes, The VDM-SL Reference Guide. Pitman, 1991.
[9] S. Hekmatpour and D. Ince, Software Prototyping, Formal Methods and VDM. Addison-Wesley, 1988.
[10] 伊理正夫，古林隆，ネットワーク理論，日科技連，1976.
[11] C. B. Jones, Software Development A Rigorous Approach. Prentice-Hall, 1980.
[12] C. B. Jones, Systematic Software Development using VDM. Prentice-Hall, 1990 (2nd ed.).
[13] C. B. Jones and R. C. Shaw, Case Studies in Systematic Software Development. Prentice-Hall, 1990.
[14] J. T. Latham, V. J. Bush and I. D. Cottam, The Programming Process An Introduction using VDM and Pascal. Addison-Wesley, 1990.
[15] S. Prehn and W. J. Toetenel(eds.), VDM'91 Formal Software Development Methods. 2 vols. LNCS 551, 552, Springer-Verlag, 1991.

執筆者紹介 山 﨑 利 治 (Toshiharu Yamasaki)

1957 年名古屋大学理学部数学科卒業．同年吉沢会計機(株)入社．翌年日本レミントン・ユニバック(株)へ移籍．著書に，「プログラム言語」(昭晃堂，1989 年)，「プログラムの設計」(共立出版，1990 年)，共訳書に，M. T.ベルティーニ，Y.タリノー共著「構造的コボル教則本」(TBS 出版会，1977 年)，M.ジャクソン著「システム開発」(共立出版，1990 年) などがある．現在日本ユニシス(株)システム技術本部 研究開発部 主席研究員．

JSD仕様のRAISE記述への変換

The Translation of JSD Specifications into RAISE Language

峰尾欽二

要　約　産業界における形式的ソフトウェア開発法の普及を促進するための一つの試みを報告する．形式的方法の利点は一部の研究者の間では認められてはいるものの，実務に応用されるには至っていない．数多くの障害がその普及を阻んでいる．一方，産業界で比較的取り組みやすい優れたソフトウェア開発法にJSD（Jackson System Development）がある．しかし，JSDは形式的な仕様記述言語を持たないので，半形式的な方法に位置づけることができる．JSDに，より厳密なスタイルと支援系を与えるために，適切な仕様記述のための形式的な枠組みを調べた．その結果，JSDの仕様を形式的仕様記述言語で記述するにはRAISE（Rigorous Approach to Industrial Software Engineering）が適切であることが解った．モジュール機構や並行処理機能等を備えているからである．ここでは，小さな課題を例として，JSDの仕様をRAISE仕様記述言語で記述し直す．ここで提案する形式指向JSDが十分実用的であることを示す．

Abstract　This is intended to report one of the experimental trials for encouraging a wider acceptance of formal methods for software development efforts. Although some advantages in formal methods have been recognized by some theorists, they still stay a long way from practical use because of a number of existing blocking hurdles. In the meantime, JSD (Jackson system development) has been available in the industry as the method comparatively easy to use. JSD, however, may be positioned as one of semi-formal methods because it has no formal specification description language. Then, with a view to giving JSD a more rigorous description style and more efficient support tools, studies have been made of formal frameworks for its appropriate description of specifications, resulting in the discovery by the author's team of the fact that RAISE (Rigorous Approach to Industrial Software Engineering) is best suited for the description of JSD specifications in a formal specification description language because of its features including modularization mechanism and concurrent operation for systems model building.

Taking an example from a small task, this paper illustrates how JSD specifications can be described in RAISE language to prove that formality-oriented JSD as proposed here serves for productive applications.

1. はじめに

産業界においてもソフトウェア開発に形式的方法を使用する関心は高まりつつある．形式的方法によれば，ソフトウェアの品質も開発効率も顕著に改善することができる．ヨーロッパ産業界の経験がこのことを立証している．仕様記述言語の一つであるLOTOSが国際規格に選定されて以来，形式的方法を標準化しようとする動きが見られる．VDM（Vienna Development Method）を英国の標準規格にしようとする動

本稿は次のPROCEEDINGSからリライトしたものである．
K. Mineo, S. Munakata, T. Yamasaki, "An Experiment on Specification Description in RAISE Method", CONFERENCE PROCEEDINGS JCSE '92, pp. 175～181, March 25-27, 1992, Olympic Plaza Youth Hostel, Seoul, Korea.

きは，この一例である．

形式的方法は，形式的構文と形式的意味を持った，仕様記述のための形式言語を備えている．この言語を通して，数学的厳密さを持ちながらソフトウェア開発のための整合性，完全性，正当性に関して仕様自身を議論することができる．さらにこの形式の利点を活かして，ソフトウェア開発に役立つ支援系，具体的には構造エディタ，型チェッカ，証明系，アニメータ，変換系等，を用いることができる．

JSD は極めて実践的な方法であり，安定した保守しやすいシステムをつくることができる．JSD は二つの原則でこのことを保証する．第一は，課題環境をモデル化することによる．モデルは機能より安定しているからである．第二は，現実世界の重要な実体に対応したプロセスをつくることによる．このことから，自然で簡素なモジュール化が可能になる．JSD のモデルは，非同期通信による協調プロセスのネットワークになる．ネットワーク上の個々のプロセスは，現実世界の実体かシステムに要求される機能に対応する．実体プロセスは，その実体が実行したり，影響を受けたりする行動を発生順序列として構造化したものであり，そのため現実世界と正確に対応付けられる．

こうした JSD の優れた性質とは別に，プロセス間通信に扱いにくさがある．プロセス間は無限長バッファを媒介して通信するため，ネットワークの公平性やデッドロックは扱いにくい．この問題は JSD に同期通信機能を備えた適切な形式言語を与えることで解決できる．さらにその言語に支援系をつくることで一層容易になる．

数多くの仕様を形式言語で書いてみることで，このアイデアを検証してみたい．幸い，この目的に適う言語はいくつも存在する．LOTOS，VDM，Z，RAISE 等はこの候補である．ここではまず RAISE を採り上げる．その第一の理由は，その名前に由来する．RAISE は Rigorous Approach to Industrial Software Engineering の略であり，VDM を産業界向けに発展させたものである．第二の理由は，RAISE がモジュール機構や並行処理機能を備えていて，いまの課題に適切であるからである．

本稿ではRSL（RAISE Specification Language）で仕様を書く．RSLのモジュール機構と並行処理機能は，JSD仕様のプロセスモデルとプロセス間通信をより厳密なものにするだろう．次章以降では，よく知られているシンプル銀行（SIMPLE BANK）[8]を採り上げ，いくつかのスタイルでRSL仕様を書く．つぎに，J.ジャクソンによる JSD 仕様を引用した後，同じ仕様を JSD 風の RSL 仕様に変換する．

2. RSL の仕様

シンプル銀行の口述風な仕様は，以下のとおりである．

「この銀行には多勢の顧客がいる．顧客は頭金を添えて自分の口座を開設できるし，解約もできる．顧客はいつでも自分の口座に預金したり，引き出したりできる（超過引出しは認められる）．しかし，超過引出しにも預金残高にも利息はつかない．この銀行システムでは超過引出しには報告書が要求されている．つまり，顧客が口座残高の値を負にする引出しを行った場合には，その都度報告を必要とする．銀行の支店長は，どの顧客でもその残高を調べることができる．つまり，顧客名をシステムに入力すれば，その顧客の最新の残高が出力される．」

RSL は広範囲な記述力を備えた言語である．抽象型と具体型，関数型と操作型，逐次型と並行型，それぞれの仕様記述を許す．ここでは三つの記述スタイルを紹介する．つまり，抽象的関数型，具体的関数型，具体的操作型である．RSL では，課題，開発段階に応じて適切なスタイルを選択しなさい，という精神である．

2.1 抽象的関数型仕様

```
scheme BANK =
  class
    type
      Cust,
      Amount, Balance,
      Cstate,
      Opt_report == empty | report
    value
      new : Cstate,

      invest : Cust × Amount × Cstate → Cstate,

      payin : Cust × Amount × Cstate ~→ Cstate,

      withdraw : Cust × Amount × Cstate ~→
                          Cstate × Opt_report,

      terminate : Cust × Cstate → Cstate,

      enquiry : Cust × Cstate ~→ Balance
    axiom
      forall c1, c2 : Cust, a1, a2 : Amount, cs : Cstate •
      [payin_invest]
        payin(c1, a1, invest(c2, a2, cs))
          ≡ if c1 = c2 then invest(c1, a1 + a2, cs)
                else invest(c2, a2, payin(c1, a1, cs)) end,
      [withdraw_invest]
        withdraw(c1, a1, invest(c2, a2, cs))
          ≡ if c1 = c2 then (invest(c1, a2 − a1, cs), if a2 − a1 < 0 then report
                                                      else empty end)
            else invest(c2, a2, withdraw(c1, a1, cs)) end,
      [terminate_new]
        terminate(c1, new) ≡ new,
      [terminate_invest]
        terminate(c1, invest(c2, a2, cs))
          ≡ if c1 = c2 then terminate(c1, cs)
                else invest(c2, a2, terminate(c1, cs)) end,
      [enquiry_invest]
        enquiry(c1, invest(c2, a2, cs))
          ≡ if c1 = c2 then a2
            else invest(c2, a2, enquiry(c1, cs)) end
  end
```

class…end は，クラス式といい，仕様が持つ値とそれらの性質を定義する．つまり，BANK という名前を付けた抽象データ型の台（値集合 carrier）は，型 Cstate の値であり，台に作用する演算群 new, invest, payin 等を定義する．

type 以下では，この仕様に必要なデータ型の名前を定義している．この段階では，Cust, Amount, Cstate 等名前だけであり，詳細な型定義は行っていない．その意味で，この仕様は抽象的である．Opt_report は可変型と呼ばれ，empty か report から成り立つことを示す．

value 以下では，この抽象データ型に必要な演算を定義する．六つの演算のうち，

new は型 Cstate の値であることを示しているだけで，いまの抽象レベルでは，空の値を持つことまで示す必要はない，と考えている．その他の演算は，定義域の値から値域の値への関数である．×は直積を，⇀は部分関数を示す符号である．

演算 invest は，顧客の型 Cust と数値を示す型 Amount で顧客たちの状態を示す Cstate を更新するものである．この段階ではまだ，invest が口座を開設する演算であるとは理解しがたいが，名称から類推してほしい，という精神でいる．預金する演算 payin も同様である．引出しの演算 withdraw では，出力側は Cstate と Opt-report の二つの型が許される．口座を解約する演算 terminate は，顧客 Cust で顧客たちの状態 Cstate を更新する．演算 enquiry では，顧客と顧客たちの状態から数値(Balance)を出力する．

axiom 以下では，値の持つ性質を公理として代数的に記述する．

ここでは，公理の考え方を示そう．この抽象データ型は，Cstate 型の状態（データベースと考えてもよい）を持ち，各々の演算は，その状態を更新したり，参照したりするものである．

状態は，

$$\text{invest}\,(c_n, a_n, \text{invest}\,(c_{n-1}, a_{n-1}, \cdots(c_1, a_1, \text{new})\cdots))$$

によってつくられる，と考えることができる．terminate は，それ以前の invest を除去したものと考えられる．payin や withdraw は，一部の値を更新するものであり，enquiry は状態を変えない．Cstate の状態に対して，各々の演算がどう作用するかを考え，そこで成立する関係を等式として表現する．

公理名［payin-invest］が示す公理は以下を示す．状態 invest (c2, a2, cs) に作用する演算 payin (c1, a1, ⋯) は，c1 と c2 の値が等しければ，その状態を変更 (a1＋a2) し，そうでなければ，その前の状態 cs に作用する．再帰的に定義されているから，c1 と同じ値を見い出せないことがあるだろう．その時は，payin (c1, a1, new) に到達するはずである．この仕様では，この値は定義していない．payin の宣言で⇀を付して部分関数として定義しているだけである．この意味でこの仕様は，過少仕様といえる．この抽象レベルでそこまで定義する必要はなかろう，という精神である．他方，演算 terminate では，公理［terminate-new］を追加して，解約口座が存在しない時の演算の振舞いを定義している．

2.2 具体的関数型仕様

```
scheme BANK =
  class
    type
      Cust,
      Amount = Int,
      Balance = Amount,
      Cstate = Cust ~m~> Balance,
      Opt_report == empty | report(Cust, Text)
    value
      new : Cstate = [],

      invest : Cust × Amount × Cstate → Cstate
      invest(c, a, cs) ≡ cs † [c ↦ a],
```

```
    payin : Cust × Amount × Cstate ⇏ Cstate
    payin(c, a, cs) ≡ cs † [c ↦ cs(c) + a]
                      pre c ∈ dom cs,

    withdraw : Cust × Amount × Cstate ⇏
                                Cstate × Opt_report
    withdraw(c, a, cs) ≡
                ( cs † [c ↦ cs(c) − a],
                    if cs(c) − a < 0 then
                      report(c, "overdrawn")
                    else
                      empty
                    end )
                  pre c ∈ dom cs,

    terminate : Cust × Cstate ⇏ Cstate
    terminate(c, cs) ≡ cs \ {c}
                       pre c ∈ dom cs,

    enquiry : Cust × Cstate ⇏ Balance
    enquiry(c, cs) ≡ cs(c)
                     pre c ∈ dom cs
end
```

データ型は前節の例に比べてより具体的になっている．Cstate＝Cust$\xrightarrow[m]{}$Balance は，定義域 Cust から値域 Balance への写像を示す型である．

new は，空の銀行をつくる演算である．演算は，value 以下で宣言の行と公理の行の組で定義するが，このとき，全称記号（forall）を付けず，簡略に表現する．

新しく口座を開設する演算 invest の公理行，invest (c, a, cs)≡…は，cs に値 c から a への写像 [c↦ a] を重ね合わせることを示す．† は，重ね合わせ（override）の演算記号である．

payin は，特定の顧客 c の残高 cs(c) に整数値 a を加えて，cs を更新（重ね合わせ）する演算である．⇏により部分関数として宣言された payin は，pre 行で前件，つまり指定された顧客は銀行に口座を開設していなければならない，を持つ (dom は定義域を示す符号)．この意味でも先の例題より詳細になっている．

withdraw も同様に考えればよい．ただし，口座の残高が負になった場合は，例外レポート report を出力しなければならない．そうでない場合は何もしない．

terminate は，特定の顧客の口座を解約する演算を意味する．cs\{c} は，cs の定義域からその要素 c を除去することを示す．

enquiry は，特定の顧客の残高を出力する演算で，その顧客は口座を開設していなければならない，ことは自明だろう．

2.3 具体的操作型仕様

```
scheme BANK =
  class
    type
      Cust,
      Amount = Int,
      Balance = Amount,
      Cstate = Cust →m Balance,
      Opt_report == empty | report(Cust, Text)
    variable
```

```
  cs : Cstate
value
  new : Unit → write cs Unit
  new() ≡ cs := [],

  invest : Cust × Amount → write cs Unit
  invest(c, a) ≡ cs := cs † [c ↦ a],

  payin : Cust × Amount ~→ write cs Unit
  payin(c, a) ≡ cs := cs † [c ↦ cs(c) + a]
                pre c ∈ dom cs,

  withdraw : Cust × Amount ~→
                          write cs Opt_report
  withdraw(c, a) ≡
          cs := cs † [c ↦ cs(c) − a] ;
          if cs(c) < 0 then
            report(c, "overdrawn")
          else
            empty
          end
          pre c ∈ dom cs,

  terminate : Cust ~→ write cs Unit
  terminate(c) ≡ cs := cs \ {c}
                  pre c ∈ dom cs,

  enquiry : Cust ~→ read cs Balance
  enquiry(c) ≡ cs(c)
                pre c ∈ dom cs
end
```

RSL では操作型の仕様記述を許す．この型の仕様では変数，この例では cs，を持ち，この変数に値を代入することができる．関数型以外に操作型仕様記述を準備した理由は，いくつかあるようである．ほとんどの仕様は操作的プログラム言語で開発されるから，段階的開発のある局面で操作型で仕様を記述することは便利であろう．操作型仕様は関数の引数の数を減らす利点もある．課題によっては，始めから操作型で記述した方が適切である場合もあるだろう．以上が操作型仕様が準備された理由と考えられる．

ここでは先の関数型仕様を操作型で書き直してみた．仕様 BANK は，型 Cstate の変数 cs を持つ．演算が変数の値を変更する場合は，演算の型の宣言行で変数名の前に write を書く．変数の値を参照するだけの場合には，read を書く．演算は関数であるからその入出力に対しては，その型を宣言の行に書く．入出力を必要としない場合には，特殊な型 Unit を書く．仕様例は，前節の仕様を参考にすれば，解読は容易だろう．

3. JSD の仕様

JSD は，産業界のソフトウェア技術者にも比較的よく知られている方法である．課題の仕様を文献[8]から引用する（図 1，図 2，図 3）．

図 1 で CUSTOMER-0 は実世界のプロセスであり，CUSTOMER-1 はシステム内のモデルプロセスである．両者はデータ列 C を通して結合している．CUSTOMER-0 と CUSTOMER-1 の結合は，非同期である．つまり，円の中の C は無限バッファを意味する．同様に ENQUIRY-FN プロセスは，CUSTOMER-1 の現在の状態を非同期

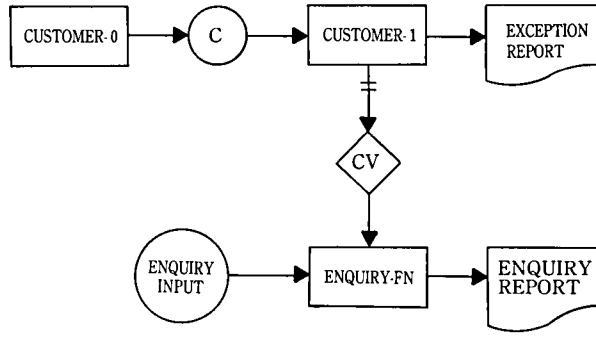


図 1　システム仕様図
Fig. 1　System specification diagram

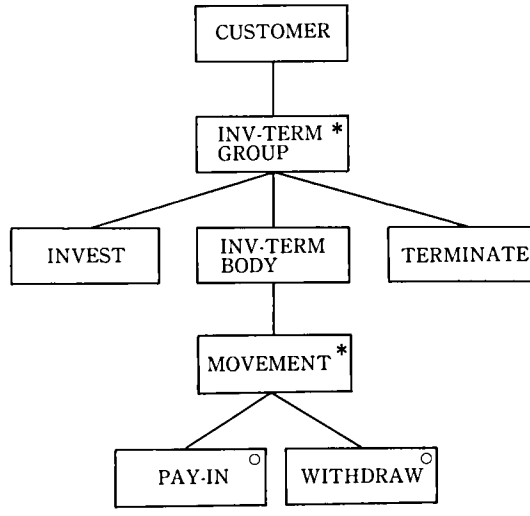


図 2　実体構造
Fig. 2　Entity structure

に調べる．

CUSTOMER-1 は実世界の顧客たちを模擬したものであるから，プロセスの数は実世界の顧客たちの数と同じである．

この仕様は，顧客の初期モデルにシステムが要求する機能を追加して拡張したものであるから，この仕様記述は，理解しやすく，保守しやすいものでもある．この仕様が実際の計算機の上で動くならば，仕様記述の正しさを確証できる．計算機上で JSD の仕様を動かすことは，ソフトウェアのプロトタイプに他ならず，ソフトウェア開発には極めて有効である．ここでは JSD の仕様を RSL の仕様記述に変換する．つくられる RSL 仕様は，厳密なものであり，適切なツールで支援することが可能になる．

4. JSD の仕様から RSL の仕様へ

本章では RSL のモジュールと並行処理を扱う．モジュールには，スキームとオブジェクトの 2 種類がある．スキームは，宣言の集まりからなるクラス式 class…end に名

```
CUSTOMER-1 seq
  balance := 0;
  read C;
  INVEST seq
    balance := amount;
  INVEST end
  read C;
  CUSTOMER-BODY itr while (PAY-IN or WITHDRAW)
    MOVEMENT sel (PAY-IN)
      PAY-IN seq
        balance := balance + amount;
      PAY-IN end
      read C;
    MOVEMENT alt (WITHDRAW)
      WITHDRAW seq
        balance := balance – amount;
        P-EXC-REPORT sel (balance < 0)
          write 'overdrawn';
        P-EXC-REPORT end
      WITHDRAW end
      read C;
    MOVEMENT end
  CUSTOMER-BODY end
  TERMINATE;
CUSTOMER-1 end


ENQUIRY-FN seq
  read ENQUIRY-INPUT;
  ENQUIRY-FN-BODY itr
    ENQUIRY seq
      get state-vector of specified CUSTOMER-1;
      write 'balance is', balance;
    ENQUIRY end
    read ENQUIRY-INPUT;
  ENQUIRY-FN-BODY end
ENQUIRY-FN end
```

図 3　構造文

Fig. 3　Structure text

前を付けたものである．先の RSL の仕様例では，BANK がそれである．オブジェクトは，クラス式をいわば具体例化したもので，その一つ一つを識別子で区別する．馴染みの用語でいえば，スキームはクラス式の「型」であり，オブジェクトはクラス式の「インスタンス」である．以下の例では，B1 と B2 はスキーム BANK のオブジェクトである．

object
  B1 : BANK
  B2 : BANK

オブジェクト内の宣言部分（type, value 等が示す）は，他のオブジェクトから参照されたり，参照したりすることができる．他のオブジェクト内を参照する時は，接頭辞としてオブジェクト名を付ける．いまの例では，B1. new と B2. new は，おのおのオブジェクト B1 と B2 内の演算 new への参照である．演算 B1. new と B2. new は同じではない．

オブジェクト配列を使って，同じ種類のオブジェクトを一時に多数定義できる．たとえば，

```
type
  Bank_No={|n : Int・1≤n∧n≤3|}
object
  B[i : Bank_No] : BANK
```

では，スキーム BANK の三つのオブジェクト B[1], B[2], B[3] を定義している．

RSL では，スキームはオブジェクトを引数として，パラメータ化できる．その用法を二つのプロセス間の簡単な通信例で考えよう．

```
scheme CHANNEL =
  class
    channel
      source, increment : Int
  end

scheme SENDER(H : CHANNEL) =
  class
    value
      i : Int = 0,
      Sender : Unit → in H.increment
                      out H.source Unit
      Sender() ≡
        H.source!i; let j = H.increment?
                    in i := j end; Sender()
  end

scheme ADDER(H : CHANNEL) =
  class
    value
      Adder : Unit → in H.source
                     out H.increment Unit
      Adder() ≡
        let i = H.source? in i := i + 1;
            H.increment!i end; Adder()
  end
```

二つのスキーム，SENDER と ADDER は，スキーム CHANNEL のオブジェクト H でパラメータ化されているため，スキームの中の二つのプロセス，Sender と Adder は，H が持つ共通のチャネル，source と increment で通信できる．プロセス宣言の中で in の後に入力チャネル名を，out の後に出力チャネル名を書いて，プロセスが通信するチャネルを定義する．

RSL の並行処理の機構と表記は，基本的に CSP から引き継いだものである．互いに通信する並行プロセスは，チャネルを介して，同期的に通信する．プロセス Sender がチャネル H. source を介して値 i を出力しようとしているとき，つまり H. source ! i が実行可能になっているとき，もう一つのプロセス Adder が同じチャネルを介して入力の準備ができていれば，つまり，H. source ?が受信準備できているとき，両者は通信可能となる．プロセスは本質的にはチャネルを備えた関数である．この例では二つのプロセスとも入出力を持たず(そのことを Unit が示している)，チャネルを介して，無限に通信し合うことになる．実際に二つのプロセスが相互に通信し合うシステムとして完成させるのには，プロセス間の並行記述等が必要であるが，これは後の例で示す．

通信環境によっては，同時に異種通信を行いたいことがある．外部選択枝□によって異種通信の選択を表す．expr 1□expr 2 と書くと，expr 1 と expr 2 はおのおの他のプロセスと通信するが，いずれか一方が通信し同時に両者が通信することはない．いずれが評価されるかはその時の外部環境に依存する．

```
scheme TYPE =
  class
    type
      Database = Int-set,
      Cust = {|e : Database • p(e)|}
    value
      p : Database → Bool
  end


scheme CHANNEL =
  class
    channel
      enquiry : Unit,
      enq_res : Int
  end

scheme MANAGER(T : TYPE, H[i : T.Cust] : CHANNEL) =
  class
    channel
      externalin : T.Cust,
      enqreport : T.Cust × Int
    value
      Manager : Unit →
          in externalin,
             {H[i].enq_res | i : T.Cust}
          out enqreport,
              {H[i].enquiry | i : T.Cust}
          Unit
      Manager() ≡
          let i = externalin? in
            H[i].enquiry!() ;
            let j = H[i].enq_res? in
              enqreport!(i, j)
            end
          end
  end

scheme CUSTOMER(H : CHANNEL) =
  class
    type
      Balance = Int
    channel
      invest, payin, withdraw : Int,
      terminate : Unit,
      overdraft : Text
    value
      Customer : Unit ⇒ in invest, payin, withdraw, terminate, H.enquiry
                          out overdraft, H.enq_res
                          Unit,
      Customer1 : Balance ⇒
          in payin, withdraw, terminate, H.enquiry
          out overdraft, H.enq_res
          Unit
    axiom
      Customer() ≡
          let b = invest? in Customer1(b) end,
```

```
        ∀ b : Balance • Customer1(b) ≡
            let x = payin? in Customer1(b + x) end
            ‖
            let x = withdraw? in
                if b < x then
                  overdraft!"overdrawn"
                end ;
                Customer1(b − x)
            end
            ‖
            H.enquiry? ; H.enq_res!b ; Customer1(b)
            ‖
            terminate? ; skip
    end

object BANK :
    class
      object
        T : TYPE,
        M : MANAGER(T, H),
        H[i : T.Cust] : CHANNEL,
        C[i : T.Cust] : CUSTOMER(H[i])
      value
        Bank : Unit →
              in {C[i].any|i : T.Cust}, M.externalin, {H[i].enq_res|i : T.Cust}
              out M.enqreport, {H[i].enquiry|i : T.Cust}, {C[i].any|i : T.Cust}
              Unit
        Bank() ≡
            M.Manager()
              ‖
            ‖ {C[i].Customer() | i : T.Cust}
    end
```

シンプル銀行の並行プロセス版RSLの仕様を理解する準備が整った．顧客を示すデータ型は，スキーム TYPE の中で具体型 Cust で表現されている．MANAGER と CUSTOMER 間をつなぐチャネルは，スキーム CHANNEL のオブジェクト配列で表現される．スキーム CUSTOMER は仮引数として，H : CHANNEL を持ち，オブジェクト宣言，C[i : T. Cust] : CUSTOMER(H[i]) によって，顧客オブジェクト C[i] とチャネルオブジェクト H[i] が対応する．顧客たちはオブジェクト配列 C[i : T. Cust] で表され，その配列の大きさは実際の顧客数と同じである．

プロセス Manager は，ここでは未定義の外部プロセス（もしくは入出力機構）とチャネル externalin と enqreport を介して通信している．同様に一つのプロセス Customer も未定義の外部プロセスとチャネル invest, payin, withdraw, terminate, overdraft を介して通信している．

Manager は，多数のチャネル {H[i].enq_res | i : T. Cust} と {H[i].enquiry | i : T. Cust} を持ち，おのおの対応するプロセス C[i].Customer と接続する．

オブジェクト BANK は，ひとつのオブジェクトプロセス M. Manager() と多数のオブジェクトプロセス {C[i].Customer()|i : T. Cust} の並行プロセスとして実現される．記号 ‖ は，その左式と右辺が並行プロセスであることを示す．

この仕様は，先に見た機能中心の抽象データ型による仕様記述よりも複雑に見える．通信プロセスに馴染みのない実践家にとっては，とくにそうであろう．しかしこの仕様は，実世界のモデルに基づいているために，JSD のモデルと同様に安定している．

JSD 仕様と比較しても，この仕様は，理解しずらいかもしれない．しかしこの仕様

は，形式に基づいている，つまり厳密であり，数学的意味を持ち，ツール上で実行可能である．

さらにJSDとの比較でいえば，JSD仕様の通信は，非同期であるが，RSLでは同期的である．通信プロセスの仕様でデッドロックや飢餓状態を回避するには，後者の方が有利であろう．

5. おわりに

JSDの小さな仕様をRSLの記述に変換してみた．このことの意味を以下に記す．

今まで議論してきたように，JSDによれば，機能中心の方法によるよりも，安定した，理解容易で保守しやすい仕様を得ることができる．ここではJSDのシステム仕様図と構造文をRSL記述に変換することで，半形式的なJSDの仕様に形式を与えた．得られた仕様は，形式指向のJSDといえる．

正しい仕様の作成に貢献するツールを作ることもできる．構文チェッカや型チェッカは，ヨーロッパの経験によれば，それだけでも実務家の仕様作成に相当貢献することが知られている．RSLの記述からAdaやC^{++}への変換系は，JSDの実現段階を改善するだろう．仕様の詳細化の正しさを確認できる証明系もすでに存在する．RAISEを初めとする形式的方法を実務の場に応用する技術的環境は整っている，といえる．

日本の現状では，形式的方法の実践普及には障害が多い．良い仕様例を沢山形式言語で書き直してみることは，形式的方法の普及を促進する一つの方法である．

今後の課題としては，もう少し大きな問題の仕様を書いてみること，そしていろいろなタイプで書き比べてみてそれらを評価すること，さらにできれば証明系を用いて正しさを確認しながら実行可能な言語まで実現すること，がある．

なお，稿中のRSLの記述に関して，デンマーク，CRI A/SのBent Dandanell氏に誤りの指摘と貴重な助言を頂いたので，ここに感謝する．

参考文献 [1] D. Bjørner et al.: Formal, Model-oriented Software Development Methods, InfoJapan'90 Proceedings, 1990, IPSJ, pp. 33～58.
[2] S. Brøck et al.: RAISE Method Manual, RAISE/CRI/DOC/3/V1, CRI A/S, Denmark, 1990.
[3] P. M. Bruun et al.: RAISE Tools User Guide Release 1.2.1, LACOS/CRI/DOC/4/V6.0, CRI et al., 1992.
[4] J. R. Cameron: JSP & JSD: The Jackson Approach to Software Development, Second Edition, IEEE Computer Society Press.
[5] The RAISE Language Group: The RAISE Specification Language, Prentice Hall, 1992.
[6] C. George: The RAISE Specification Language: A Tutorial, VDM'91 Symposium Proceedings, LNCS 552, Springer-Verlag, 1991, pp. 141～237.
[7] C. A. R. Hoare: Communicating Sequential Processes, Prentice-Hall International, 1985. (吉田訳，ホーアCSPモデルの理論，丸善，1992)
[8] M. A. Jackson: System Development, Prentice-Hall International, 1983. (山崎，大野監訳，システム開発 JSD法，共立出版，1989)
[9] C. B. Jones: Systematic Software Development using VDM, Second Edition, Prentice-Hall International, 1990.

[10] S. Prehn, W. J. Toetenel (Eds.): Formal Software Development Methods, VDM'91 Symposium Proceedings, LNCS 551, Springer-Verlag, 1991.
[11] A. Sutcliffe: Jackson System Development, Prentice Hall, 1988.
[12] 山崎利治，ソフトウェア開発の形式的方法，技報28号，日本ユニシス，1991.

執筆者紹介 峰 尾 欽 二 (Kinji Mineo)
昭和38年東京工業大学経営工学科卒業，41年日本ユニシス(株)入社，教育部のプログラミング教育担当を経て，現在，研究開発部所属．

宣言型通信サービス仕様記述言語からプロセス仕様を生成する手法

A Method of Generating Process Specifications from a Declarative Communications Service Specification Language

河田慶三, 田倉 昭, 太田 理

要 約 われわれは，通信システム設計者でなくても，通信サービスを容易に記述できる形式的通信サービス仕様記述言語STRを提案してきた．STRでは通信サービスを広域状態遷移を表現するものと定義し，広域状態遷移をプロダクション規則の集合で記述する．STRのプロダクション規則はグラフの書き換え規則とみなすことができる．STR記述を実現する通信ソフトウェアは，通信することによってグラフの同形判定を行う．従来，通信ソフトウェアはオートマトンとしてモデル化されてきたが，オートマトンではグラフの同形判定を行えない．

本稿ではグラフの同形判定を行えるような通信ソフトウェアのモデルを提案し，STRからそのモデルに従った通信ソフトウェアを自動生成する手法について論じる．

Abstract The authors have been proposing STR, a declarative communications service specification language, which allows even those who are not communications systems designers to give an easy description of communications services. STR, where communications services are defined as representing global state transitions, is designed to describe such transitions in sets of production rules. STR's production rules can be regarded as the rules for rewriting graphs. The communications software compatible with STR descriptions judges the isomorphism of graphs designated in the STR language by means of communications. The modeling of communications software has so far been done with the help of the automaton theory, by which it is impossible to recognize isomorphic graphs.

This paper discusses a communications software model which is capable of identifying isomorphic graphs and how to let STR generate communications software conforming to a designated model.

1. はじめに

近年，情報化社会の進展にともない，通信サービス利用者のニーズは多様化し，これまでには考えられなかった新しい通信サービスを提供する必要が生じつつある．また，通信サービスを実現するソフトウェアの開発の効率化も重要な課題になっている．

従来，通信サービス仕様は自然言語による記述，通信ソフトウェアの動作概要を示す状態遷移図やメッセージ・シーケンス・チャートを参考資料として添付した形式で記述されてきた[1]．こういった通信サービス仕様の記述法は，サービスの概要を把握するには有益であるが，曖昧さを排除することがむずかしく，通信サービス仕様記述から通信ソフトウェアを自動的に生成することは不可能であった．

われわれは，こういった問題を克服するために，曖昧なく通信サービス仕様を記述できる形式的通信サービス仕様記述言語STRを提案してきた[2][3]．STRでは，通信ネ

本稿は，電子情報通信学会，信学技報 AI 92-6 より学会の許可を得て転載したものである．

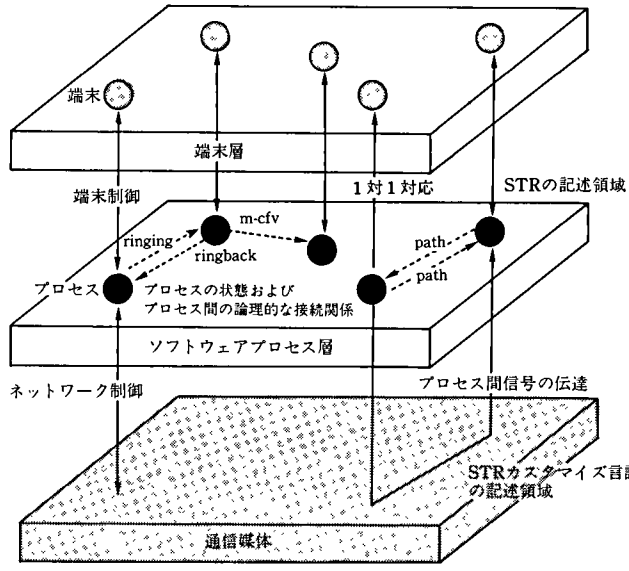


図 1　通信システムのモデル

Fig. 1　System model

ットワークをブラックボックスとみなし，その内部機構については記述しない．そのため通信システム設計者でなくても，容易に通信サービスを記述できる．

通信システムは図1に示すように，端末層，ソフトウェア・プロセス層，通信媒体の3層構造をしていると仮定する．ソフトウェア・プロセス層は，分散された均質なプロセスから構成されている．通信サービスとは，端末入力によって引き起こされる複数プロセスに関する広域的な状態遷移を規定するものととらえる．STR は広域状態遷移をプロダクション規則の集合で記述する．広域状態として複数のプロセスが複雑にからみあった状況を表現できるので，非常に多くの通信サービスが広域状態遷移によって記述できる．STR を用いることによって，今後必要となるであろう新しい通信サービスを容易に記述することが可能になる．

通信ソフトウェアは，ソフトウェア・プロセス上に実装される．プロセス自体は局所的な情報しか保持していないが，周辺プロセスと通信することによって，広域状態を把握し広域状態遷移を実現する．従来，通信ソフトウェアのモデルとして，オートマトンが用いられてきたが，単純なオートマトン・モデルでは，広域状態遷移を実現できない．そのため，通信マシンのモデルとして，オートマトンを一般化した新しいモデルを提案する．

本稿では，STR から通信ソフトウェアを自動生成する手法について論じる．第2節で，通信システム，通信サービスおよび STR を定義する．また，STR とプロダクション・システムとの関係を論じる．STR のプロダクション規則は，グラフの書き換え規則とみなすことができる．第3節で，STR とグラフ文法の関係について説明する．第4節で，通信ソフトウェアのモデルについて論じる．第5節で，STR から通信ソフトウェアの仕様を SDL 形式で生成する手法について説明する．

2. 通信サービス仕様

2.1 通信システム

通信システムは，端末層，プロセス層，および通信媒体から構成される．本稿では，プロセス層についてのみ論じる．

プロセスは分散配置されており，各プロセスは識別子(PID)を持つ．プロセスは各時点において，特定の状態を持つ．状態中には，接続している周辺プロセスのIDが含まれる．プロセスは状態に応じて端末を制御する．システム中のすべてのプロセス上には，同一のソフトウェアが実装される．そのため，特別な機能を持ったプロセスは存在せず，プロセスはすべて均質である．

プロセスの集合PDに対して，PD中のプロセスの状態およびそれらの間の関係をPDの広域状態と呼ぶ．また，システムを構成するすべてのプロセスに関する広域状態を，システムの状態と呼ぶ．

プロセス集合PDの広域状態遷移とは，PD中の特定のプロセスが対応する端末から特定の入力を受けることによって，PDの広域状態(始広域状態)が新しい広域状態(次広域状態)に遷移することである．

以上の仮定のもとで，通信サービスを次のように定義する．

(通信サービス) 通信サービスとは，広域状態遷移を与えるものである．

この定義は非常に一般的であり，多くの新しいサービスをこの定義の枠組内で表現することが可能になる．

2.2 プロダクション・システム

通信サービスとは，広域状態遷移であると定義したが，AIの分野において，広域状態を論理式で表現し広域状態の遷移を扱う問題が，プロダクション・システムとして数多く研究されてきた．ここでは，STRIPS[4]というロボットの問題解決システムを例として，プロダクション・システムについて説明する．

STRIPSでは状態と目標を論理式で記述し，状態の遷移をプロダクション規則で与えている．状態を表現する論理式は，リテラルの連言（論理積）で記述される．

リテラルは，次のように定義される．すなわち，リテラルは素式あるいは素式の否定であり，素式は複数の項を引数として持つ述語である．項とは，定数または変数である．たとえば，on (C, A) は定数C，Aを引数として持つ述語であるので，素式であってリテラルでもある．

図2に，状態と目標の記述例を示す．図の論理式中の項，A，B，Cは積木を表す定数である．

STRIPSのプロダクション規則は，前提条件式，消去リスト，追加式の3種類の要素で構成される．プロダクション規則の前提条件式が成立しているとき，その規則を適用することが可能であり，規則を適用した場合，現状態から消去リストで指定された状況が消去され，追加式で指定された状況が付加される．プロダクション規則中では，積木を表現するのに変数を用いる．図3に，プロダクション規則の例を示す．

STRIPSでは，以下の手続きPRODUCTIONを実行することによって，DATAで表現するシステムの状態が逐次に更新され，最後に目標状態を得る．

手続き　PRODUCTION

1. DATA ←初期のデータベース
2. until DATA が終了条件を満足するまで do
3. begin
4. DATA に適用可能な規則の集合の中からある規則 r を選ぶ.
5. r を DATA に適用し DATA を変更する.
6. end

4.で適用可能な規則 r を選択する際に，規則中の変数と実データとの間のユニフィケーションが実行される.

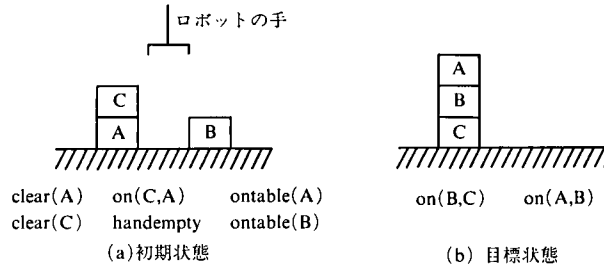


図 2 STRIPS の状態と目標の表現

Fig. 2 Representations of states and goals in STRIPS

```
(1)pickup (x)
前提条件式：ONTABLE(x),HANDEMPTY,CLEAR(x)
消去リスト：ONTABLE(x),HANDEMPTY,CLEAR(x)
追加式：    HOLDING(x)
```

図 3 プロダクション規則例

Fig. 3 Example of production rule

2.3 通信サービス仕様記述言語 STR

STR では，STRIPS と同様に広域状態を論理式で表現し，広域状態遷移を，プロダクション規則の集合で記述する．STR のプロダクション規則(以下 STR 規則と記す)は，始広域状態，イベント(端末入力)，次広域状態の 3 項で表現される．イベントは 1 個または 2 個の引数を持つ述語で表現される．STR 規則において，項としてはプロセスを表す変数だけが許される.

通信サービスの動作を STR 記述で記述したものを図 4 に示す．この規則は，ダイヤルトーン状態にあるプロセス A から，空 (idle) 状態であるプロセス B にダイヤルしたとき，プロセス B が着信転送を設定 (m-cfv(B, C)) しており，その転送先であるプロセス C が空状態の時には，B への呼びは，C に転送され，A は呼返音 (ringback-tone) 受信状態へ遷移し，C は呼出音 (ringing-tone) の鳴っている状態に遷移する

```
dialtone(A),idle(B),m-cfv(B,C),idle(C)    /* 始広域状態* /
dial(A,B) ：                              /* イベント* /
ringback(A,C), ringing(C,A),              /* 次広域状態* /
pingring(B,A),m-cfv(B,C).
```

図 4 通信サービス規則例

Fig. 4 Example of a communication service rule

ことを規定している．

図4の規則で，dialtone，idle等はプロセスの状態を表現する述語であり，dialはイベントを表現する述語である．

3. グラフ文法

プロセスは周辺プロセスと通信することによって広域状態を把握する．そのため，STR規則の始広域状態記述において，イベント生起プロセスから始広域状態記述中のすべてのプロセスへのパスが存在せねばならない．このようにプロセスの連結性を議論するには，論理式表現よりもグラフ表現の方が適している．

STR規則中の始広域状態記述とイベント記述は，あわせて1個のラベル付きの有向グラフとして表現できる．また，次広域状態記述も1個のラベルつき有向グラフとして表現できる．

図4の規則に対応するグラフ表現を図5に示す．図5において，規則中のringback (A, C)という記述は，点Aから点Cへのringbackというラベルを持つ有向辺として表現される．

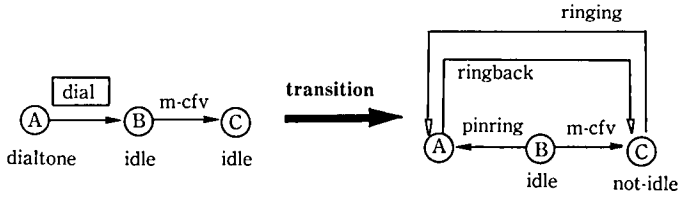


図5　サービス規則に対するグラフ表現

Fig. 5　Graph representation of a service rule

このように，STR規則は，グラフの書き換え規則とみなすことができる．

グラフの書き換えシステムはグラフ文法として定式化され，パターン認識等に応用されてきた[5]．グラフ文法は，グラフの中の点を別のグラフで置き換えるものと，グラフ中の辺を別のグラフに置き換えるものに大別できるが，STRは後者に属する．

通信システム全体の状態も，グラフで表現できる．これをシステム・グラフと呼ぶことにすると，STR規則の意味するところは，次のように定義できる．

(STR規則の意味) STR規則は，システム・グラフ中の部分グラフで規則の始グラフと同形なものを，規則の次グラフと同形なものに置き換えることを指定する．

プロダクション・システムにおける変数のユニフィケーションは，グラフ書き換えシステムにおいては，同形グラフの探索問題に置き換えられる．図6に，STR規則が適用される状況を図示する．

3.1 規則の優先順位

イベントが生起したとき，複数の規則が適用可能になる場合がある．このような規則間の競合を回避するため，規則間に優先順位をつける．ここでは，同一イベントを持つ規則の集合に対して，グラフの包含関係に基づいた次のような半順序を導入する．

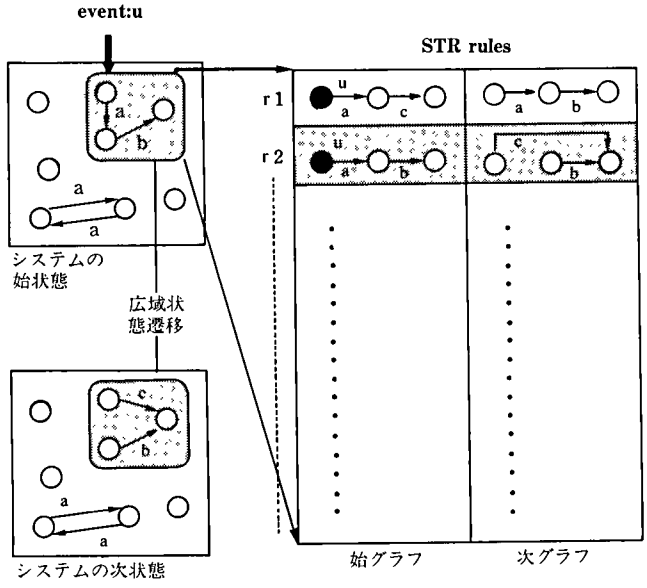


図 6 STR 規則の適用

Fig. 6 Application of STR rules

同一のイベントを持つ二つの規則 r 1, r 2 に対して，r 1 の始グラフが r 2 の始グラフを含む時その時に限り，規則 r 1 は規則 r 2 よりも優先度が高い．この順序は全順序ではないため，規則の競合を完全に回避することはできない．上の半順序を用いても規則の競合を回避できないとき，適用可能な規則のどれか一つが実行されるものとする．

3.2 規則に対する制限

プロセスは通信を行うことにより，部分グラフの同形判定を行う．部分グラフ同形判定問題はグラフの点の個数に関して NP 完全であることが知られており[6]，単純に扱えば処理しきれない．そのため，本稿では扱うグラフに次のような制限を加える．

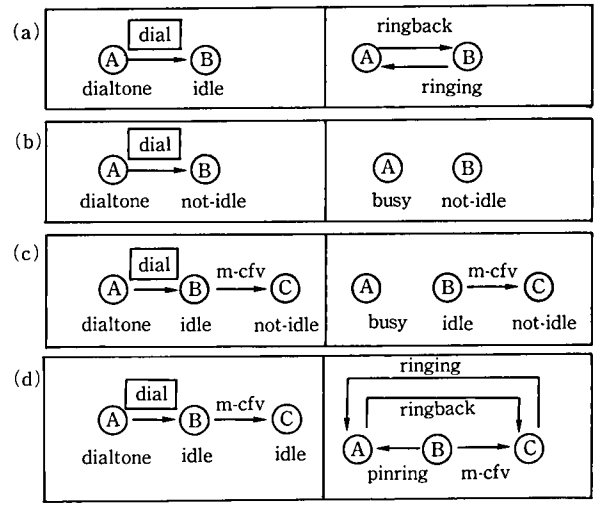


図 7 STR 規則例

Fig. 7 Example of STR rules

・規則の始グラフにおいて，辺またはラベルを持っている点は一列に並ぶ．この点の並びを幹という．

図7にSTR規則例を示す．この例では，イベントとしてdialを持つ規則だけをとりあげてある．

4. 通信ソフトウェア

4.1 プロセス間通信による広域状態遷移の実現

STRIPSにおいて，動作主体はあくまでロボットであり，積木はロボットによって操作されるものにすぎなかった．われわれのシステムにおいて，ロボットは存在せず，動作主体は分散配置されたプロセスである．STRIPSにおいて，ロボットは常にシステム全体の状態を把握することが可能であった．われわれのシステムにおいて，プロセスは局所的な情報しか保持しない．プロセスは，通信することによって周辺プロセスと情報を交換し，広域的な状態を把握して，適用できるSTR規則を決定する．このように，分散配置されたプロセス間で互いに協調的にふるまうことによって，広域状態遷移を実現する．

1) プロセスの状態……各時点におけるシステム中のプロセスの状態は，変数を含まない素式の連言で記述できる．各素式の第1項には，そのプロセス自身の識別子(PID)が入る．たとえば，識別子P1を持つプロセスで，ダイアルトーンが鳴っていて，識別子P2を持つプロセスへ着信転送が設定されている状態は，次式で記述できる．

$$dialtone(\mathrm{P}1),\ m\text{-}cfv(\mathrm{P}1,\mathrm{P}2)$$

このように状態記述中にPIDを含むので，プロセスの状態数はシステム中に存在するプロセスの個数に依存する．しかし，状態記述中のプロセスIDを無視すれば，プロセスの状態はシステムに依存せずにSTR記述だけから，有限個に分類できる．その分類番号を状態IDと呼ぶことにすると，プロセスの状態は，状態IDとPIDの列で表現できる．

2) プロセス間信号……プロセス間で送受信する信号として，状態通知信号とそれに対する応答信号の2種類用意する．

状態通知信号は，自分の状態を隣接プロセスに伝えることによって，隣接プロセスに適用できる規則の決定を依頼するために用いられる信号であり，応答信号は，決定された規則を通知するために用いられる信号である．応答信号の特殊なものとして，拒否信号がある．これは，適用できる規則が存在しないことを意味する信号である．

状態通知信号は，適用できる可能性のある規則の集合を陰に表している．順次，状態通知信号が伝えられていくに従って，適用できる可能性のある規則が絞り込まれ，最後に規則が一意に決定される．

信号はプロセスの状態を伝達する．そのため，信号中には信号IDとは別に信号を発信したプロセスの保持するプロセスのIDが含まれる．

3) プロセス間通信……プロセスP1でイベントが生起した場合，P1の状態だけで適用を決定できるSTR規則が存在するならば，その規則に応じて状態遷移を

行う．そのような規則が存在しないならば，P 1 は隣接プロセスの一つ P 2 へ状態通知信号を送信する．P 2 から応答信号を受信すると，応答信号の指示する規則に従って状態遷移を行う．応答信号として拒否信号を受信した場合は，別の隣接プロセス P 3 へ状態通知信号を送信し同様な処理を続ける．

隣接プロセス P 2 が P 1 から状態通知信号を受信した場合，P 1 からの情報と P 2 の状態で適用を決定できる規則が存在するならば，P 1 にその規則を意味する応答信号を返信し，P 2 はその規則に従って状態遷移を行う．そのような規則が存在しないならば，隣接プロセスの一つ P 4 へ状態通知信号を送信する．この状態通知信号には，P 2 の状態と P 1 の状態を含ませる．P 4 から応答信号を受信すると，応答信号の指示する規則に従って状態遷移を行う．応答信号として拒否信号を受信した場合は，別のプロセス P 5 へ状態通知信号を送信し同様な処理を続ける．図 8 に，プロセス間通信の概略図を示す．

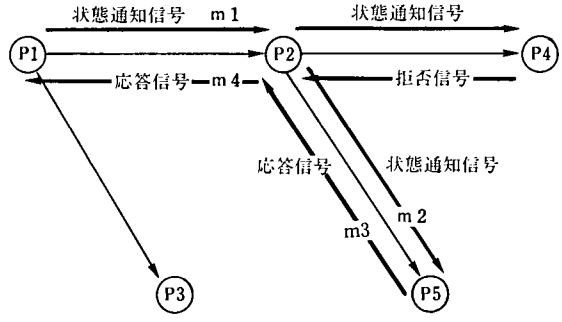


図 8　プロセス間通信例

Fig. 8　Communications among processes

4)　プロセス ID の送信……図 9 は，図 8 中のプロセス P 2 が状態 s 1 で P 5 から信号 m 3 を受信した後，m 3 の指定する規則 r 1 に従って状態 s 2 へ遷移する状況を示している．

m 1 中には P 1 の保持する PID が含まれ，m 3 中には P 5 の保持する PID が含

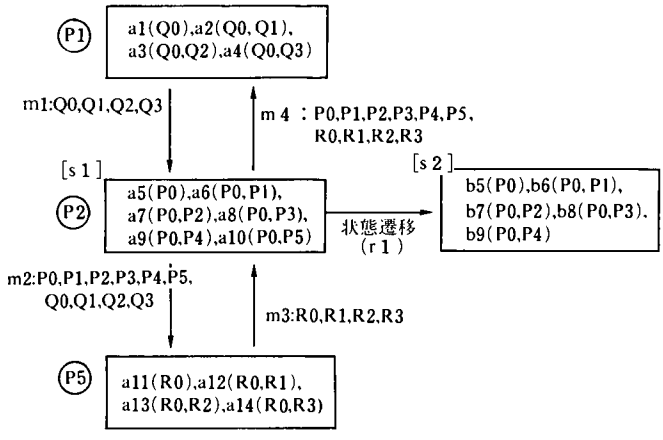


図 9　PID の送信例

Fig. 9　PID transfer

まれる．s1からs2に遷移する際に，保持しているPIDをm1とm3中のPID情報を利用して再設定する．

4.2 通信ソフトウェア

1) グラフの同形判定……図10に示す始グラフを持つ二つの規則r1とr2を考える．ここで，r1とr2は同一のイベントに対する規則であるとする．図中の二つのグラフにおいて，点A，B，Cの並びが幹を構成する．システム中にA，B，Cに対応するプロセスが存在し，それらの識別子がそれぞれPA, PB, PCだとする．

PAでイベントが起きると，PA，PB，PCの順に状態通知信号が伝達され最終的にPCで規則r1とr2の判定を行う．PCはlabel 4で自分とつながっているプ

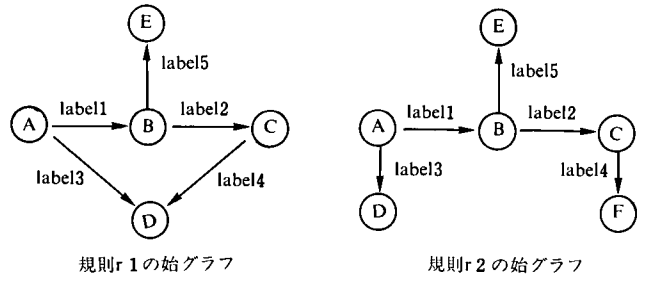


図 10　グラフの同形判定

Fig. 10　Isomorphic graph recognition

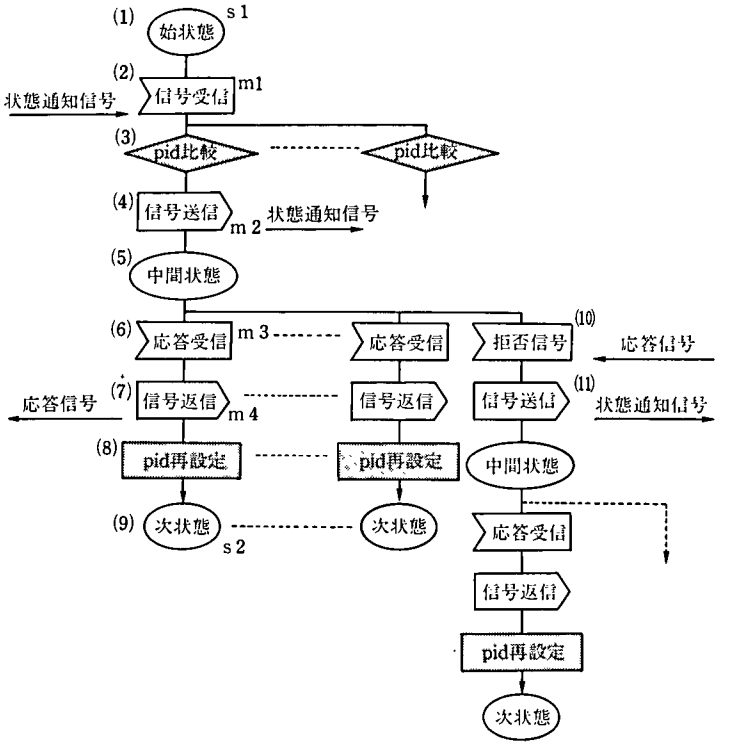


図 11　通信ソフトウェアのモデル

Fig. 11　Communication software model

ロセスが，PA と label 3 でつながっているかどうか判定する．この判定は，PC の保持する PID と状態通知信号中の PID を比較することによって実現できる．

2) 通信ソフトウェアのモデル……図 11 に，われわれの通信ソフトウェアの動作概要を SDL 形式で示す．図 11 において，(3)の PID の比較と(8)の PID の再設定を行っている部分が，従来のオートマトン・モデルと異なる部分である．

5. STR から通信ソフトウェアの生成

プロセス動作仕様の生成過程は，次の三つのステップに分けることができる．

(ステップ 1) 規則の分類

STR 規則を規則中のイベントおよび始グラフ中の点近傍によって分類し，規則の分類木を作成する．

(ステップ 2) プロセス動作パターンの生成

分類木を探索することによって，プロセスの動作パターンを生成する．

(ステップ 3) プロセス動作パターンの合成

プロセス動作パターンを用いて，アイドル状態から到達可能なすべての状態を生成し，各状態におけるプロセスのふるまいを生成する．この時に，STR 記述中の否定論理式を展開する．

5.1 規則グラフ中の点近傍

STR 規則の始グラフにおいて，ラベルを持つ点はイベント生起点を先頭にして一列に並んでいる(幹)．この並びに応じて幹中の点の間に親子関係を設定できる．この親子関係に基づいて，始グラフ中の点の近傍を以下の 3 種類導入する．

1) 単純近傍……v と v から出ている辺およびそれらに付随するラベルで構成される近傍である．単純近傍はプロセスの状態を表現している．

2) 祖先近傍……v の単純近傍の各辺に対して，その辺の終点 w が v の祖先を始点とする辺の終点になっている場合，祖先の種類（親，祖父等）と祖先を始点とする辺に付随しているラベルを単純近傍に付加したもの．w が複数の祖先に対して，それらを始点とする辺の終点となっている場合，もっとも関係の近い祖先から出ている辺を優先し，その情報だけを付加する．

3) 長男近傍……v の祖先近傍中の辺のうち，v の長男に向かう辺に長男への辺であることを示すラベルを付けたもの．

図 12 に，規則グラフとその点近傍の例を示す．この例において，点 1 の祖先近傍は単純近傍と同じである．また，点 3 の長男近傍は祖先近傍と同じである．

5.2 分類木の生成

STR 規則を規則中のイベント，幹中の点の単純近傍，祖先近傍，長男近傍をキーとして分類し，規則の分類木を作成する．分類木中のノードには，対応する STR 規則の集合および分類キーとなったイベントまたは近傍が含まれる．分類木の葉には，分類された結果として 1 個の規則だけが含まれる．分類木は，文字列の集合中から特定の文字列を探索するために用いられるデータ構造 trie [8] を拡張したものになっている．これは，STR 規則の始グラフは幹という特徴を持つため可能になっている．図 13 に分類木の概略図を示す．

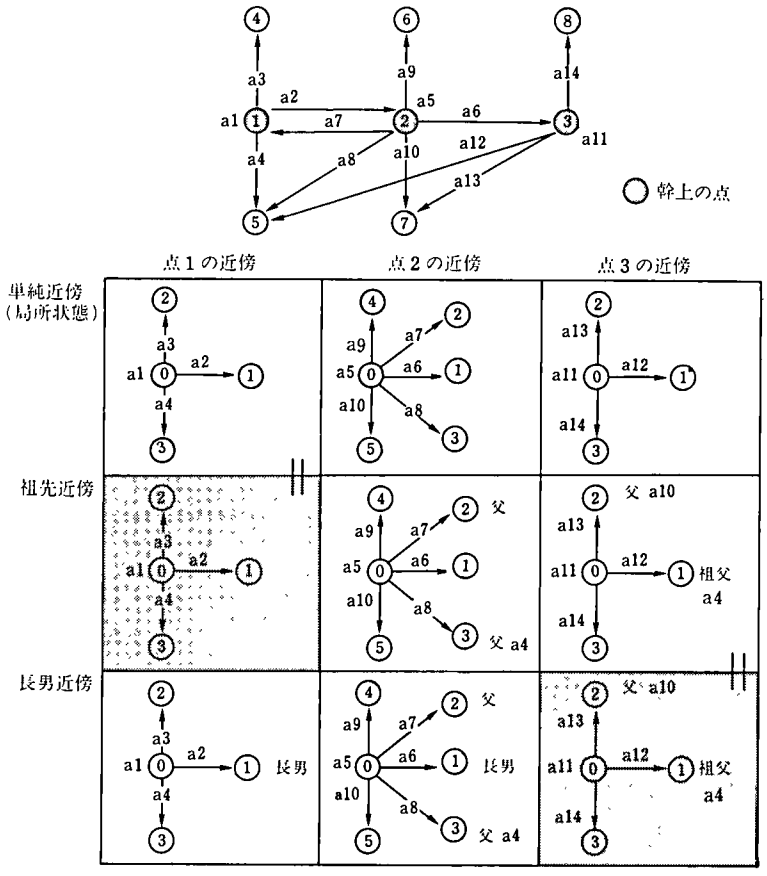


図 12　グラフ中の点近傍

Fig. 12　Vertex neighbourhoods in a rule graph

5.3　プロセス動作パターンの生成

図 13 の STR 規則の分類木から，図 11 に示されているようなプロセス動作パターンを生成する．

分類木において，第 i 番目の点の近傍に関するノードすべてで構成される部分木に対して，1 個プロセス動作を生成する．分類木を縦型に全探索することによって，順次プロセス動作を生成していく．

1)　始状態・次状態の生成……分類木において，単純近傍に対応するノードからプロセスの状態を取り出す．

2)　PID 比較動作の生成……分類木中の祖先近傍に対応しているノードから祖先近傍を取り出す．取り出した祖先近傍中の祖先情報が付随している辺に対して，PID 比較動作を生成する．

たとえば，ラベル a 8 を持つ辺が祖先情報として（父，a 4）を持つ場合（図 12 の点 2 の祖先近傍），a 8 という関係で接続しているプロセスの ID と，受信信号中で親プロセスが a 4 という関係で接続しているプロセスの ID を比較する．

3)　送信信号の生成……分類木中の長男近傍に対応するノードに対して，状態通知

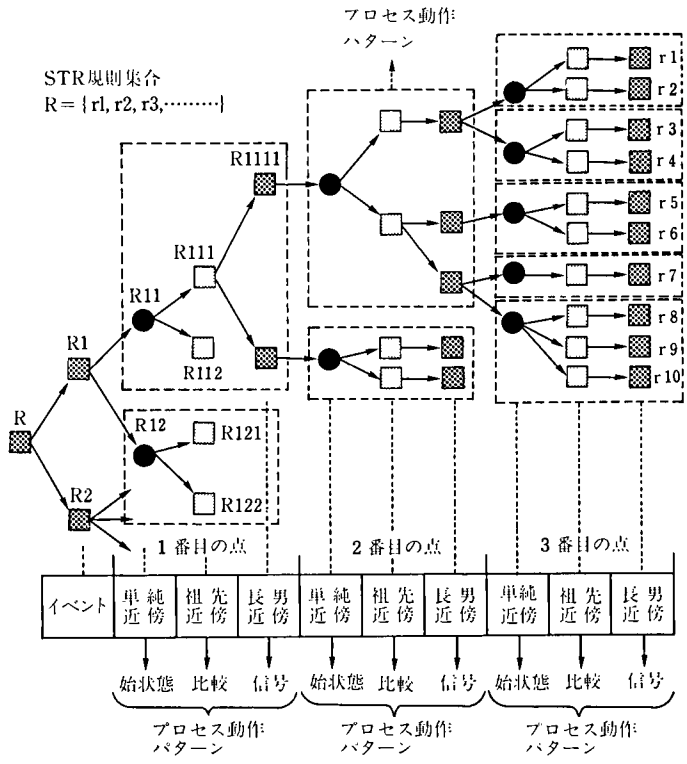


図 13 分 類 木

Fig. 13 Classification tree

信号を1個生成する．信号中には，状態中の PID と受信した状態通知信号中の PID を含める．生成された信号は，このノードが保持している STR 規則集合を暗に意味している．

4) PID 再設定動作の生成……近傍抽出時の情報を利用して，PID の再設定動作を生成する．

図 14 に STR 規則グラフから単純近傍（プロセスの状態）を抽出した例を示す．図のように，近傍抽出時に近傍中の各辺の終点と規則グラフ中の点の対応リストを作成しておく．状態中の PID は近傍中の辺の終点と対応しているので，このリストを用いて，状態中の PID と規則グラフ中の点との対応を確かめることができる．図 15 に図 14 中の規則 r 1 を適用して，図 9 中の P 2 が s 1 から s 2 へ状態遷移する場合の PID 設定例を示す．

図 14 中の対応リストを用いて，状態 s 1，s 2，信号 m 1，m 3 中の PID と規則グラフ中の点の対応を知ることができる．図 15 は，これらのリストから s 2 中の PID を s 1，m 1，m 3 中から探す方法を例示している．

5.4 プロセス動作パターンの合成

生成されたプロセス動作パターンを用いて，アイドル状態から到達可能なすべての状態を生成し，各状態におけるプロセス動作を生成する．

① アイドル状態の展開

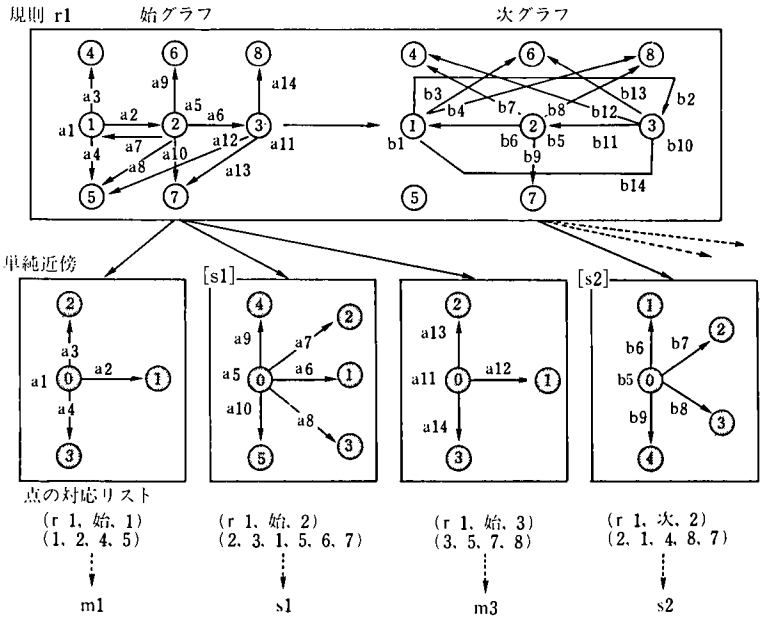


図 14　近傍の抽出

Fig. 14　Extraction of neighbourhoods from rule graphs

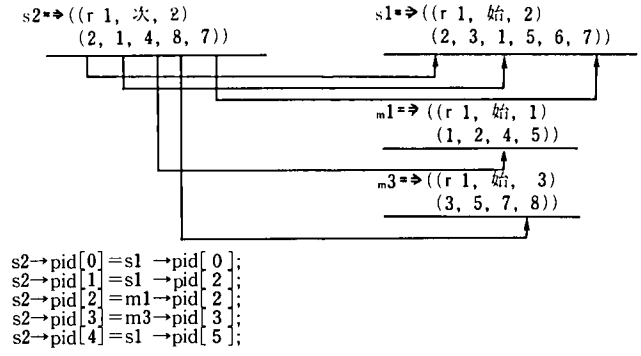


図 15　PID 設定例

Fig. 15　PID update

ステップ 2 で生成されたプロセス動作パターンのうち，idle を始状態として持つパターンを集め，アイドル状態時の動作を合成する．

② 次状態の展開

①で生成されたプロセス動作のすべての次状態 s に対して，ステップ 2 で生成されたプロセス動作パターンのうち，s に含まれる状態を始状態として持つものを集め，そのパターンと①で生成された動作パターンとを合成する．

図 16 に次状態の展開を例示する．図中のパターン 1 は，アイドル状態から展開が進んで，状態 s 1 を始状態とするパターンが張りついているところである．図中のパターン 2 は，始状態 s 3 が s 1 に含まれ，パターン 1 と同じ信号 m 1 を受信した場合のプロセス動作パターンである．パターン 1 中には，拒否信号を

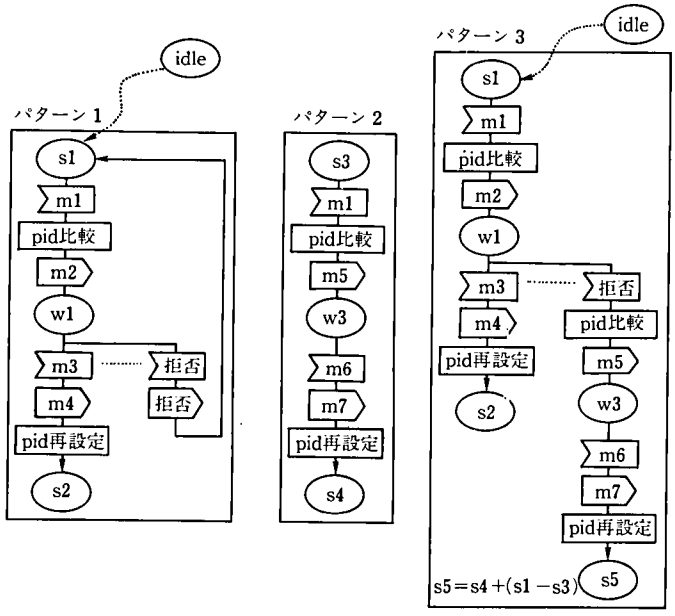


図 16　プロセス動作パターンの合成

Fig. 16　Synthesizing process behaviour patterns

受信して始状態に戻る動作が必ず含まれている．パターン1中の拒否信号受信後の動作を，パターン2中のm1受信後の動作に置き換える．このとき付加されるパターン2の次状態s4をs5に書き換える．s5はs4にs1とs3の差分(s1−s3)を付加した状態である．こうして，新しい状態が生成される．

合成されたパターン3を①に戻して，②の次状態の展開を新しい状態が生成されなくなるまで繰り返す．

5.5　STR中の否定記述の展開

5.5.1　STR中の局所状態記述

STRのプロダクション規則において，同一プロセス変数を第1引数として持つリテラルを集めたものを，局所状態記述と呼ぶ．STR規則中の局所状態記述は，プロセスの状態を表現している．局所状態記述中に否定表現が含まれる場合，1個の状態記述で複数の状態を表現する．一般に状態記述sdは，次の形式で表現できる．

$$sd=((M_0(A_0), M_1(A_0, A_1), \cdots, M_k(A_0, A_k)),$$
$$(N_0(A_0), N_1(A_0, A_1), \cdots, N_k(A_0, A_k)))$$

ここで，M_0，M_1，…，M_kは，それぞれ否定のつかない述語を要素とする集合であり，N_0，N_1，…，N_kは，それぞれ否定された述語を要素とする集合である．

状態記述sdは，次の式(5-1)，(5-2)を満たす状態すべてを表現している．

$$s=(L_0(A_0), L_1(A_0, A_1), \cdots, L_k(A_0, A_k)) \quad (5\text{-}1)$$
$$\forall L_i, L_i \supset M_i, L_i \cap N_i=\phi \quad (5\text{-}2)$$

1個の局所状態記述が表現する状態の数は巨大であるが，そのすべてを生成する必要はない．局所状態記述で表現される状態のうち，アイドル状態から到達可能な状態

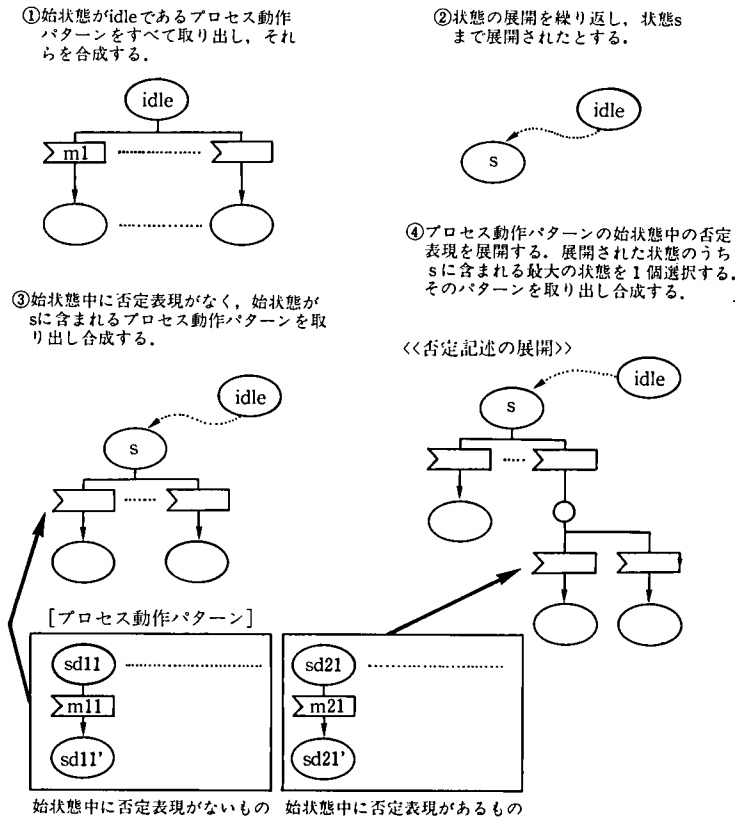


図 17　プロセス動作パターン合成時の否定記述の展開

Fig. 17　Example of negative expressions at process behaviour pattern synthesis step

だけ生成してやればよい．ステップ2で生成されたプロセス動作パターン中では，状態は論理式表現されている．プロセス動作パターン合成時に，否定記述を展開する．図17に否定記述の展開のようすを図示する．

6. おわりに

STR記述から生成された通信ソフトウェアは，プロセス間通信を行うことによって，STR規則グラフの同形判定を行う．規則グラフ中の点近傍によって規則の分類木を作成すると，分類木中のノードと通信ソフトウェアの動作要素がうまく対応する．この関係を利用して，STRから通信ソフトウェアを自動生成する手法を説明した．

本稿では，通信信号中にはプロセスの保持する情報がすべて含まれるものとしたが，通信ソフトウェアは信号中の情報をすべて利用するとは限らない．そのため，信号中に含まれる情報はもっと少なくても良い．こういった最適化をどう実現するかについては，今後の課題である．

本研究を進めるにあたり，熱心な御討論を頂いたATR通信システム研究所諸兄に深謝の意を表する．

参考文献

[1] Bellcore: "LSSGR features common to residence and business customers III", Issue 2, 1989.

[2] Y. Hirakawa, T. Takenaka "Telecommunication service description using state transition rules", Proc. Sixth Int. Work. Software Specification and Design, Como, Italy, Oct. 1991. (to be published).

[3] K. Kawata, Y. Hirakawa "Efficient Process Generation from State Transition Based Telecommunication Service Specifications", Proceedings of 5-th JCCNSS, Jul, 1992, pp. 3～8.

[4] R. E. Fikes, N. J. Nilsson "STRIPS: a new approach to the application of theorem proving to problem solving", Artificial Intelligence, 2(3/4), 1971, pp. 189～208.

[5] H. Ehrig, H. J. Kreowski, G. Rozenberg (EDs.) "Graph Grammars and Their Application to Computer Science", Lecture Notes in Computer Science 532, 1991.

[6] M. R. Garey, D. S. Johnson "Computers and Intractability: A Guide to the Theory of NP-completeness", Freeman, San Francisco, 1979.

[7] M. T. Liu "Protocol Egineering", ADVANCES IN COMPUTERS, Vol. 29, 1989, pp. 79～195.

[8] E. Fredkin "Trie memory", Comm. ACM 3: 9, 1960, pp. 490～499.

執筆者紹介

河　田　慶　三 (Keizo Kawata)

1981年京都大学理学部卒業. 1984年同大学院修士課程修了. 同年日本ユニシス(株)入社. ソフトウェアの開発に従事. 1989年12月ATR通信システム研究所に出向. 通信ソフトウェアの自動生成に関する研究に従事. 情報処理学会会員.

田　倉　　昭 (Akira Takura)

1979年東京工業大学理学部 情報科学科卒業. 1981年同大学院修士課程修了. 同年日本電信電話公社（現日本電信電話株式会社）入社. プロトコル設計支援およびその適用技術に関する研究に従事. 1992年2月ATR通信システム研究所に出向. 電子情報通信学会, 情報処理学会, IEEE会員.

太　田　　理 (Tadashi Ohta)

ATR通信システム研究所通信ソフトウェア研究室長. 1968年九州大学工学部 電子工学科卒業. 1970年同大学大学院修士課程修了. 同年日本電信電話公社（現日本電信電話株式会社）入社, 電子交換機のソフトウェア研究開発に従事. 1987年からNTTにおける情報・通信機器向けの共通OSの開発に従事. この間トロン協会CTRON専門委員会の幹事を歴任. 1992年2月にATR通信システム研究所に出向, 現在に在る. 工学博士. 電子情報通信学会, 情報処理学会, IEEE会員.

ユーザインタフェース生成系「御結び」について

OMS/B: A User Interface Generation System

溝 上 昌 宏

要 約 近年の計算機に関する環境の飛躍的な発展によって，プログラミング言語システムにおけるユーザインタフェースの系統的な記述方法論の確立が重要な研究課題となっている．「御結び」(OMS/B: Object Manipulating System/Basic) は，直接操作型のインタフェースを構築するためのシステムであり，Common Lisp のプログラミングにおけるオブジェクトブラウザとして有効である．また，その基本概念は他のプログラミング言語システムにおける視覚的ユーザインタフェース生成系を実現するための基盤となる技術である．

Abstract The recent remarkable development in the environment of computations has made it an important subject of studies to establish a systematic methodology by which to describe user interfaces in programming language. The OMS/B (Object Manipulating System/Basic) has proved effective for the creation of a direct manipulation interface and as an object browser in the sphere of Common Lisp programming. In addition, its underlying concepts are thought of as providing a basic technique which helps implement a generation system for graphical user interfaces by using other programming language systems as well.

1. はじめに

プログラミング言語でプログラミングを行うために必要なソフトウェア環境として，ユーザインタフェースを系統的に生成するシステムの研究が行われている．一般に，プログラミング環境におけるユーザインタフェースにはキーボードからの入力を中心とするコマンド型インタフェースとポインティングデバイスからの入力を中心とする直接操作型インタフェースがある．

「御結び」(OMS/B：Object Manipulating System/Basic) [4][5][6]は，マウスからの入力による直接操作型のインタフェースを構築するためのシステムであり，対象となるプログラミング言語として Common Lisp[1]を想定している．「御結び」によって構築されたユーザインタフェースでは，Common Lisp のオブジェクトを表示型と呼ばれる表示の仕方に従ってディスプレイ上に表示し，マウスで指示しながらオブジェクトを直接操作することができる．本稿では，「御結び」の概要を説明し，実装における問題点の一つである表示の更新について述べる．

2. 「御結び」の概要

2.1 基本概念

「御結び」は Common Lisp のオブジェクトをディスプレイ上に表示し，マウスで指示しながら直接操作を行うことのできるユーザインタフェースを生成するシステムである．ディスプレイ上のオブジェクトを表現している部分を表示 (view) と呼ぶ．同じオブジェクトでもその表現方法はいろいろ考えられる．「御結び」では，オブジェ

クトの表示方法と操作方法を表示型（viewtype）として定義することによって，ユーザインタフェースを記述する．オブジェクトは表示型を指定することによって表示され，一つのオブジェクトに対して同時に複数の表示を対応させることができる．表示型はオブジェクトのクラスに対して定義され，任意のクラスはそのスーパークラスの表示型を継承する．同名の表示型を異なるクラスに対して定義することも可能で，オブジェクトをその表示型で表示する時はそのスーパークラスのうちで最も特定的なクラスに対する定義が用いられる．

「御結び」では，表示に関するすべての操作をマウスによって行うことを前提としている．マウスの移動やボタンのアップダウン等の動作をイベントと呼び，それに応じて実行される処理をアクションと呼ぶ．標準で備えられているアクションは次の通りである．

- 表示スロットの値を変更する．
- 表示の内容を更新する．
- 表示を複写する．
- 表示型を変更する．
- 表示を消去する．
- 表示の場所を移動する．

このように，「御結び」はオブジェクト・インスペクタとしての機能を持っている．

2.2 表示型の定義

表示型を定義するにはマクロ defview を用いる．defview の構文と意味は次の通りである．

defview *view-type* (*name type*) &body *body*

型 *type* のオブジェクトに対して表示型 *view-type* を定義する．表示型はクラスまたはインスタンスに対して定義できる．*body* は表示型定義の本体であり，make-view および make-slot-view を用いて表示の仕方を定義する．*name* は *body* の中でオブジェクトを参照する変数として使用できる．

2.3 表　　示

一般に，Lisp のオブジェクトには，値を保持できる場所を持つオブジェクトとそうでないオブジェクトがある．たとえば，コンスの car および cdr は値を保持できる場所であり，その場所の値を変更することができる．つまり，オブジェクト自身を変更することはできないが，それに含まれる下位のオブジェクトを変更することはできる．これに対し，数値やキャラクタは下位のオブジェクトを持たないオブジェクトである．

実際にオブジェクトを表示するには，マクロ make-view または make-slot-view を用いる．これらはマウス操作による値の変更を許すかどうかによって使い分けられる．make-view によって表示されたオブジェクトは変更できないが，make-slot-view によって表示されたオブジェクトは変更できる．したがって，トップレベルのオブジェクトの表示には make-view を用い，オブジェクトのスロット値の表示には make-slot-view を用いる．それぞれの構文と意味は次の通りである．

make-view *object* &optional *view-type* &rest *options* &key :view

make-view は *object* を表示型 *view-type* で表示する．表示されたオブジェクトを

変更することはできない.

make-slot-view *object slot-form* &optional *view-type* &rest *options* &key : view

make-slot-view は *object* の *slot-form* で表されたスロットの内容を表示型 *view-type* で表示する. 表示されたスロットの値は変更することができる. *slot-form* がオブジェクトのスロットを表す形式でない時は, make-view で表示した結果と同じになる.

2.4 表示の書式制御

マクロ with-aligned は, defview の *body* 中に記述して表示の書式制御を行う. with-aligned を入れ子にして用いることによって表形式で表示することができる. with-aligned の構文と意味は次の通りである.

with-aligned (*direction-and-border*
&optional *alignment tabular*
&body *body*

with-aligned は *body* の中で返された表示または枠を *direction-and-border*, *alignment*, *tabular* に従って位置を揃えて並べた枠を作り, それを値として返す. *direction-and-border* は, :vertical, :vertical-simple-border, :vertical-border, : horizontal, :horizontal-simple-border, :horizontal-borderのいずれかで, 表示または枠を並べる方向と, それらの周りに表示される罫線の有無を指示する. それぞれの意味は表 1 の通りである.

表 1 キーワード direction-and-border
Table 1 direction-and-border keywords

キーワード	方向	罫　　線
:vertical	垂直	表示されない.
:vertical-simple-border	垂直	一つ下のレベルの表示または枠の間に表示される.
:vertical-border	垂直	一つ下のレベルの表示または枠の間と, with-aligned が値として返す枠の周りに表示される.
:horizontal	水平	表示されない.
:horizontal-simple-bordar	水平	一つ下のレベルの表示または枠の間に表示される.
:horizontal-border	水平	一つ下のレベルの表示または枠の間と, with-aligned が値として返す枠の周りに表示される.

alignment は一つ下のレベルの表示または枠の位置の揃え方を指示し, *tabular* は二つ下のレベルの表示または枠の位置の揃え方を指示する. *alignment* は *direction-and-border* が : vertical-…の時は :right, :center, :left のいずれかで, :horizontal-…の時は :top, :center, :bottom のいずれかになる. 省略時はいずれも :center になる. *tabular* は *direction-and-border* が :vertical-…の時は :right, :center, :left を要素とするリスト, :horizontal-…の時は :top, :center, :bottom を要素とするリストになる.

2.5 表示のマウス動作の追加/削除

表示のマウス動作を追加するには関数 add-mouse-action を, 削除するには関数 delete-mouse-action を用いる. それぞれの構文と意味は次の通りである.

add-mouse-action *view key string function*

表示 *view* の上で動作 *key* が行われた時に関数 *function* が実行されるように登録する.

delete-mouse-action *view key*

表示 *view* の上での動作 *key* の登録を削除する.

2.6 表示型の定義例および表示例

次の例は, クラス sequence のオブジェクトに対して表示型 sequence-view を定義した例である.

```
defview sequence-view (seq sequence)
  (with-aligned (: vertical-border : center)
    (dotimes (i (length seq))
      (make-slot- view seq (elt seq i)))))
```

文字列"abc"を定義した sequence-view で表示すると図のようになる.

#\a
#\b
#\c

次の例は, with-aligned を入れ子にして表形式の表示型を定義した例である. cons-view はクラス cons のオブジェクトに適用される.

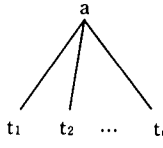
```
(defview cons-view (cons cons)
  (with-aligned (: vertical-border : left' (: left 1) left))
    (with-aligned (: horizontal-simple-border)
      (make-view"CAR : "sheet-view)
      (make-slot-view cons (car cons)
                  (if (consp (car cons) 'cons-view'default-view))
      (with-aligned (: horizontal-simple-border)
        (make-view"CDR : "sheet-view)
        make-slot-view cons (cdr cons)
                  (if (consp (cdr cons) 'cons-view'default-view)))))
```

ここで, sheet-view は何もアクションが登録されていない表示型である. リスト(12) を定義した cons-view で表示すると図のようになる.

CAR:	1	
CDR:	CAR:	2
	CDR:	NIL

次の例は, クラス cons のオブジェクトを木表現で表示する表示型 tree-view である.

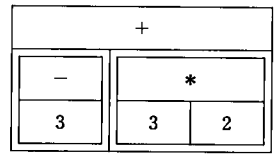
a を親, $t_1, t_2, \cdots t_n$ を a の子とする木



を，(a t_1 t_2 … t_n) で表すものとすると，tree-view を次のように定義できる．

```
(defview tree-view (form t)
    (make-view form 'default-view))
(defview tree-view (form cons)
  (with-aligned (: vertical-dorder : center)
    (cond ((= (length form) 1)
      (make-slot-view form (first form) 'tree-view))
      (t
      (make-slot-view form (first form) 'tree-view)
      (with-aligned (: horizontal-simple-border)
        (dotimes (i (1- (lenght form)))
        (make-slot-view form (elt form (1+i) 'tree-view)))))))
```

リスト (+ (− 3) (* 3 2) を tree-view で表示すると図のようになる．



3. 表示の更新

3.1 一貫性の保持

「御結び」では，表示はオブジェクトへのポインタを持っているが，オブジェクトから表示へのポインタは存在しない．これは，ある表示からそれに対するオブジェクトを参照することはできても，逆にオブジェクトからそれを表示している表示をすべて参照するということはできないことを意味する．したがって，ある表示が対応するオブジェクトの最新の状態を常に表示していることを保証することは一般に困難である．「御結び」において，表示とオブジェクトが一致しているかどうかが問題になるのは，マウス操作によって表示スロットの値を変更した場合と，使用者が定義した関数の副作用等によってオブジェクトの構造が変化した場合である．

前にも述べたように，マクロ make-slot-view は第一引数として，slot-form で表現されるスロットを持つオブジェクト *object* を必要とする．これによって，make-slot-view による表示は，表示されているオブジェクト(スロット値)へのポインタ以外に，その親オブジェクトへのポインタも持つことができる．このことを利用すれば，マウス操作によってオブジェクトのスロット値を変更した時には，そのオブジェクトを表示する他の表示も同時に更新することができる．マウスでスロット値の変更が指示されると，標準のアクションでは，次のような手続きで表示が更新される．

① 変更されるスロット値（オブジェクト）の一つ上のレベルのオブジェクトを認識する．

② 実際にスロットの値を変更する．

③ 存在するすべての表示に対して，それぞれに対応するオブジェクトが①で認識したオブジェクトを含んでいるなら，表示を更新する．

表示はウィンドウシステムにおけるウィンドウとして実現されている．③では，古

いウィンドウを消滅させて最新の内容を表示するウィンドウを新たに生成することによって表示の更新を行う．

こうして，マウス操作によるスロット値の変更では，オブジェクトと表示は一致する．しかし，これだけでオブジェクトと表示の一貫性を完全に保証することはできないので，明示的に画面上のすべての表示を更新する再表示機能を提供している．

3.2 更新時間の削減

前節で表示スロットの値の変更とそれに伴う表示内容の更新の方法について述べた．前節で示した手続きでは，表示を最新にする時はウィンドウ階層におけるトップレベルの表示（ウィンドウ）から下のすべてのウィンドウを新たに生成することになるので，多くのウィンドウから実現されている表示の更新や画面全体の再表示には時間がかかる．そこで，次の二つの手法によって表示の更新にかかる時間の削減を試みた．

・遅延更新

・表示の再利用

ここでは，これらの手法について述べる．

3.2.1 遅 延 更 新

「御結び」では，すべての表示が画面に現れているわけではなく，画面に現れない表示が存在する．たとえば，ある表示に対して表示型の変更を指示すると，変更前の表示は一時的に見えなくなる．変更後の表示に対して，表示型を元に戻すように指示すれば，変更前の表示が再び画面に現れる．

前節の表示スロットの変更のアクションでは，オブジェクトに変更が生じると，それを表す表示は実際に画面上に現れているか否かに関わらずすべて更新していた．しかし，オブジェクトの構造に変化が生じて，表示と一致しなくなっても，それが実際に画面に現れていない表示ならば，すぐに更新する必要はなく，それが次に画面に現れた時に一致していれば十分である．そこで，更新が必要な表示のうち，実際に画面に現れている表示については更新を行うが，画面に現れていない表示については更新予定のフラグをたてておく．そして，画面に現れていなかった表示が再び画面に現れた時に，更新予定フラグを調べてフラグがたっていれば表示内容を更新してから表示する．すぐに更新する必要のない表示は，必要になるまで更新を遅らせるわけである．

3.2.2 表示の再利用

make-view あるいは make-slot-view によってオブジェクトを表示する際には，ウィンドウに関するメモリ等のリソースが新しく確保される．ウィンドウの生成にかかる時間を省略するため，表示の更新ではできる限り新しい表示を作らず，それまで表示されていた表示を再利用して表示文字列だけを変更する．

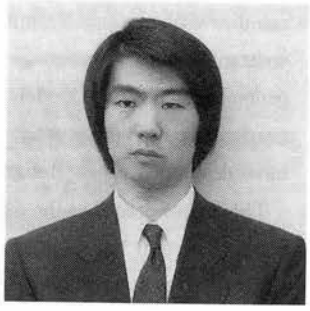
make-slot-view のキーワード引数 :view に既存の表示を指定すると，可能ならばそれを再利用する．ただし，表示すべきオブジェクトの構造に適合しなければ再利用できないので，この場合は :view は無視される．なお :view は表示の更新において使用される内部的な引数であり，使用者が直接指定することは想定していない．

4. おわりに

ユーザインタフェース生成系「御結び」について述べた．「御結び」は Common Lisp を拡張した形で実現されており，Common Lisp のプログラミングにおけるオブジェクト・ブラウザとして有効であるばかりでなく，他のプログラミング言語システムにおける視覚的ユーザインタフェース生成系実現の基盤となる技術である．なお本稿は，通商産業省工業技術院 電子技術総合研究所 言語システム研究室における技術指導のもとに行った研究成果に基づいている．御指導を頂いた言語システム研究室の二木厚吉室長ならびに戸村哲技官に感謝の意を表する．

参考文献
[1] G. Steele, "Common Lisp the Language", Digital Press, second edition, 1990.
[2] K. Kimbrough, L. Oren, "Common Lisp User Interface Environment", Texas Instruments Inc., July 1990.
[3] R. Scheifler, et al.,"CLX Common Lisp X interface Programmer's Reference."
[4] 川辺治之，戸村哲，二木厚吉，"Common Lisp Object System に基づく Object Manipulating System—基本システムの概要—"，日本ソフトウェア科学会第5回大会論文集，1988.
[5] 川辺治之，"OMS/B—直接操作型インタフェース記述システム"，ユニシス技報，第28号，1991，pp. 114～121.
[6] 戸村哲，大蒔和仁，二木厚吉，"言語システムのためのユーザインタフェース生成システム",電子計算機相互運用データベースシステム研究開発成果発表会講演予稿集，1991.
[7] 戸村哲，"プログラミング言語システム「つくばね」における言語と処理系構成法の研究"，電子技術総合研究所研究報告，第915号，1990.
[8] 井田昌之，元吉文男，大久保清貴，"bit 別冊 Common Lisp オブジェクトシステム—CLOS とその周辺—"，共立出版，1989.

執筆者紹介 溝上昌宏 (Masahiro Mizogami)
1965年生．1989年名古屋大学工学部 情報工学科卒業．同年日本ユニシス(株)入社．知識システム部所属，KEE および KES II の提供・保守に従事．現在，企画課所属，ES デモツールの開発に従事．



要求仕様理解モデルのプログラム設計への適用と評価

The Application of a Specification-comprehensive Model to Program Designing and Its Evaluation

熊 本 厚 志

要 約 要求仕様書を理解し，プログラムとして実現すべき機能を明確にする作業は，重要な設計作業の一つである．しかし，自然語で記述された要求仕様書に含まれる意味内容を整理し，実現すべき機能を明確にすることは，設計作業に不慣れな者にとっては容易な作業ではない．そこで，われわれは，要求仕様に含まれている意味内容を段階的に整理していくためのモデル（自然語理解モデル）を提案し，モデルに基づく設計作業支援システムの開発を行ってきた．

本稿では，設計作業支援システムを大学の学生実験に適用することにより，モデルと支援システムの有効性を実験的に評価した結果について述べる．具体的には，設計対象を階層的なデータフロー図で表現する作業において，支援システムを利用する場合と利用しない場合とで，得られる階層的データフロー図の構造的な差を比較する．比較の結果，支援システムを利用すると，設計対象において実現すべき機能の詳細化や分割の面では個人差が小さくなることがわかった．

Abstract In systems designing, it is important to understand requirement specifications and to clarify the functions to be made available in a program product. It is no easy task, however, for those who are not familiar with systems designing to put together the meanings and contents in a requirement specification written in natural language for clarification of functions to be included in a program. Then, after proposing a certain model (a model capable of understanding natural language) which serves to gradually put in order what are included in a requirement specification, the author and his colleagues have developed a new design support system based on this model.

This paper describes the experimentally evaluated efficiency of the model as well as of the new design support system based on sample systems developed by students. Specifically, compared are structual differences in hierarchical data flow diagrams resulting from the use and non-use of the support system to represent targated designings in hierarchical data flow diagrams. The comparison has revealed that the use of the support system helped variations of individual students drop off in terms of the break-down and partition of the functions to be implemented in object designs.

1. はじめに

近年，ソフトウェア品質と生産性向上の立場から，ソフトウェア開発過程の上流工程である要求分析と設計の自動化に対する関心が高まってきている．これまでにも，具体的なソフトウェア設計法の提案が盛んに行われている．

このような設計法の一つに複合設計法がある[1]．複合設計法は適用範囲が広いために開発現場への普及度も高く，支援システムの開発，製品化が進んでいる．

複合設計法では，段階的な機能分割を繰り返すことによって設計を詳細化していく．

その初期ステップの作業がデータフロー図による機能抽出である．データフロー図それ自体は記法が明確なため書きやすい．しかし，機能を導き出す過程についてのガイドラインが不十分なため，満足すべきデータフロー図が作成できるかどうかは設計者の経験に依存する．このことが複合設計法の修得を困難にしている．

一方，われわれが提案する自然語仕様の理解モデルでは，要求仕様の意味内容を段階的に整理する手順を示している[2][4]．とくに，設計対象となるシステムが小規模で，システム全体が1枚のデータフロー図で表現できる場合には，要求仕様からデータフロー図を作成する過程のガイドラインとしてモデルが有効であることは，適用実験によってすでに確認されている[4]．しかし，システムの規模が大きくなり，データフロー図を階層的に構成する必要のある場合については評価されていない．

筆者は，このモデルに基づいた階層的なデータフロー図の作成作業を支援するシステムの試作を行ってきている[5]．本稿では，この支援システムを大学の学生実験に適用することにより，モデルと支援システムの有効性を実験的に評価する．とくに，設計対象を階層的なデータフロー図で表現する作業に注目し分析を行う．

以下2章では自然語仕様の理解モデルの概要を紹介する．3章ではモデルに基づく設計作業支援システムの概要を説明する．4章では支援システムの適用実験の概要を述べる．5章では適用実験によって得られたデータの分析を行う．6章では今後の課題について簡単にまとめる．

2. 自然語仕様の理解モデル

2.1 モデルの概要

自然語仕様の理解モデルでは，要求仕様からデータフロー図を次に示す四つの過程を経て作成する(図1)[6]．

過程1：自然語解析

要求仕様中の曖昧な表現，冗長な表現等を整理し，定型化された単文に変換する．

過程2：P，Kグラフへの変換

定型化された単文を元に要求仕様に現れる要素間の構成関係（「AはBとCから構成される」等）をPグラフと呼ぶグラフで表現する．また，要素の分類関係((AはA′とA″2種類に分類される」等）をKグラフと呼ぶグラフで表現する．両グラフはノードに要素名を持つ木構造のグラフである．

過程3：Fグラフへの変換

PグラフとKグラフを元にして要素間に導入される機能を表現するFグラフ(要素Xが要素Yに変換される過程を表す)を作成する．Fグラフは階層的に構成される．

過程4：データフロー図への変換

階層的に構成されたFグラフからデータフロー図への変換を行う．

このようにモデルでは要求仕様をデータフロー図に変換する過程を大きく四つに分け，各過程での作業内容を明確にしている．したがって，設計作業に不慣れな者でもデータフロー図を効率良く作成することができる．また，比較的単純な構造を持つ三つのグラフを用いるため，最終的に得られるデータフロー図の構造のばらつきは小さ

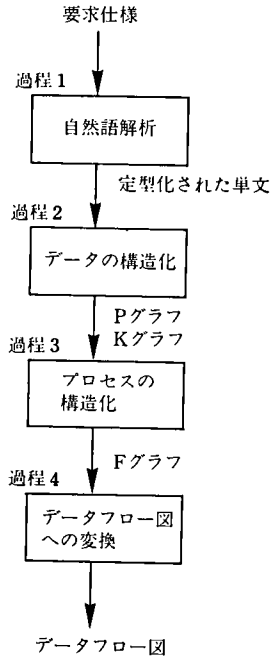


図 1　モデルによる設計手順

Fig. 1　Process of design using a model

くなると期待される.

2.2　P グラフ

P グラフは,要素間の構成関係を表現するための木構造をしたグラフである(図 2). ある親要素が同一の子要素の繰り返しで構成されている場合, 両者の間を I (is-Iter-of)のラベルの枝で結び, 親要素を I 集合体と呼ぶ. 一方, 複数種類の子要素から構成されている場合, P (are-Parts-of) のラベルの枝で結び, 親要素を P 集合体と呼ぶ. また, 子要素を持たない要素を個体と呼ぶ. 以下では, I 集合体, P 集合体, 個体のことを要素の構成型と呼ぶ.

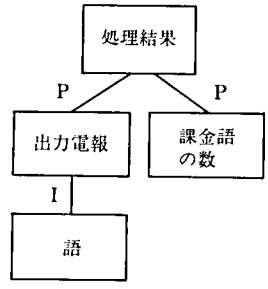


図 2　P グラフの例

Fig. 2　Example of P graph

2.3　K グラフ

K グラフは要素の分類関係を表現するための木構造をしたグラフである(図 3). 親要素に接続されている子要素は,その親要素の分類後の状態(分類結果)を表している.

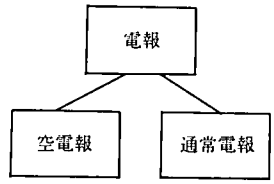


図 3　K グラフの例
Fig. 3　Example of K graph

2.4　F グラフ

F グラフは要素間の機能（操作）関係を表現したグラフであり（図 4），初期要素（F グラフへの入力にあたる要素）が，中間要素への変換を経て，目標要素（F グラフから出力される要素）へ変換される過程を表している．この例では，初期要素“電報”が目標要素“処理結果”へ変換される過程を表している．

F グラフ上では要素を□，操作を○でそれぞれ表す．分類欄には分類操作を，変換欄には変換操作をそれぞれ配置する．状態 1 欄には変換操作（および，分類操作）の入力となる要素を，状態 2 欄には変換操作の出力となる要素を，分類結果欄には分類操作の結果となる要素をそれぞれ配置する．

状態 1 欄に配置された要素と状態 2 欄の要素との間の結び方には次の 2 通りがある．①変換操作を経由して結ぶ場合と，②状態 1 の要素に対して分類操作を行い，その分類結果である要素と状態 2 の要素を変換操作を経由して結ぶ場合，である．

また，F グラフ上の変換操作の内容をより詳細に表現する必要がある場合には下位（子）の F グラフを作成する．

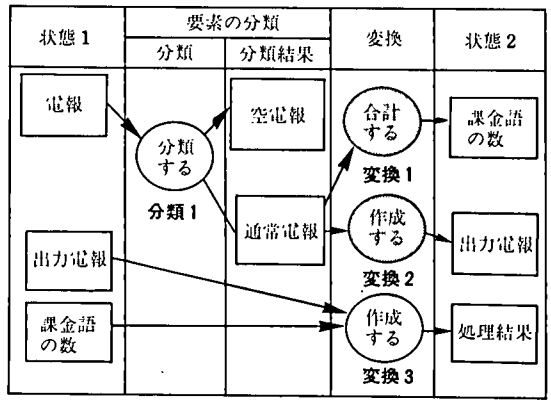


図 4　F グラフの例
Fig. 4　Example of F graph

2.5　P，K グラフから F グラフへの変換

F グラフの多くの部分は P，K グラフから得られる情報から作成が可能であり，足りない部分を設計者が補うだけで完成できる．図 4 の例では斜線の変換操作（変換 1，変換 2）が設計者の補った部分で，その他の操作，要素は P，K グラフから導出されたものである．P，K グラフから F グラフへの変換手順を図 4 の F グラフの作成過程を例に説明する．

2.5.1 Pグラフからの操作の導出

設計者は必要に応じて，Pグラフ上の要素間の構成関係を表1に従ってFグラフ上の変換操作に置き換える．図4で，目標要素“処理結果”はPグラフ（図2）ではP集合体であるので，表1の目標要素のP集合体の欄を参照し，変換操作“出力電報(子要素）と課金語の数（子要素）から処理結果（親要素）を作成する”に置き換えFグラフに配置する（図4の変換3）．この状態1欄へ配置された中間要素“出力電報”，“課金語の数”は状態2欄へも配置する．

表1 Pグラフの構成関係から導出される変換操作
Table 1 Change operation from P graph

	構成型	導出する変換操作
初期要素	I集合体	（親要素）から（子要素）を取り出す
	P集合体	（親要素）から（子要素）を取り出す．
目標要素	I集合体	（子要素）を（親要素）へ出力する．
	P集合体	（子要素）から（親要素）を作成する．

2.5.2 Kグラフからの操作の導出

設計者は必要に応じて，初期要素のKグラフ上の分類関係をFグラフ上の分類操作に一意に置き換える．図4では初期要素“電報”のKグラフ(図3)の分類関係“電報には空電報と通常電報の2種類がある”はFグラフ上では分類操作“電報を空電報，通常電報に分類する”に置き換えられる（図4の分類1）．

2.5.3 変換操作の追加

P，Kグラフから操作を導出した後，Fグラフを完成させるために必要な変換操作を設計者が追加する．図4では変換1，および変換2を追加して，Fグラフを完成している．

2.6 Fグラフの階層化

Fグラフの階層化は基本的にPグラフの構成関係に基づいて行う．たとえば，図5の最上位階層の変換1の下位のFグラフ（第2層）では入力要素“電報の流れ”の子要素“電報”に関する変換操作，分類操作を決定することにより，変換1の内容を詳細化する．また，変換1.1，変換1.2の内容を詳細化する第3層の各Fグラフでは“電報”の子要素“語”に関する変換操作，分類操作の決定を行う，といった手順で階層化を行う．

2.7 データフロー図

データフロー図とはデータの流れに着目して，設計するシステムを図式化したものである[7]．図6にデータフロー図の例を示す．矢印がデータフロー（データの流れ）を，1本線がデータストア（データの一時記憶領域，ファイル）を，バブル（○）がプロセス（機能，データに付して行われる処理）を表している．また，＊は境界点と呼ばれ，データフロー図で表されたシステムへのデータの出入口を表す．

設計対象のシステムの規模が大きい場合には，階層的に構成された複数のデータフロー図によってシステム全体を表現することが必要となる．すなわち，システム全体

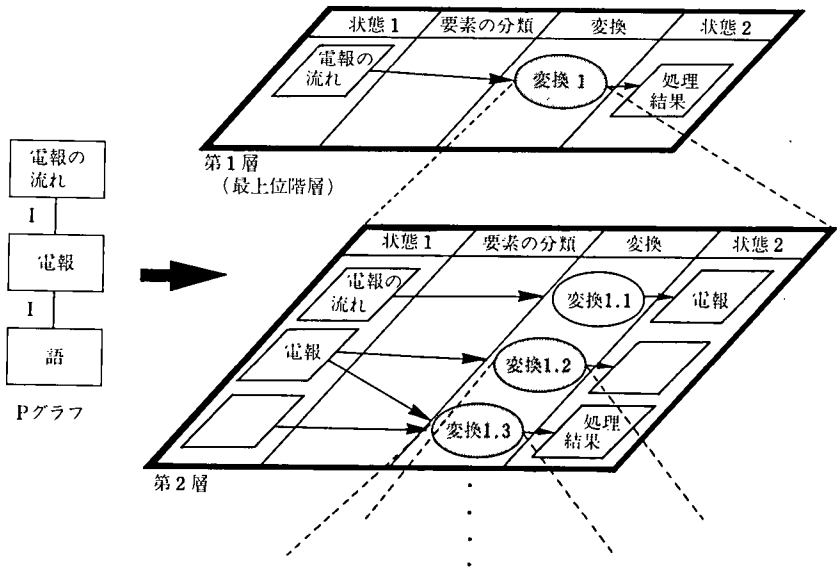


図 5　F グラフの階層化

Fig. 5　Example of F graph

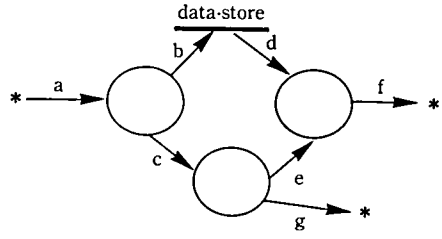


図 6　データフロー図の例

Fig. 6　Example of data flow diagram

をいくつかのサブシステム（プロセス）に分けてデータフロー図を作成する．次に，各プロセスの内容を詳細化した子データフロー図を作成する．以下，必要があれば，さらに子データフロー図を作成し，プロセスを詳細化していく．

以下の章では，階層的に構成された複数のデータフロー図群のことを「階層的 DFD」と呼び，その構成要素である 1 枚のデータフロー図のことを「DFD」と呼ぶ．

3. 支援システム

3.1 システムの概要

図 7 にデータフロー図作成支援システムの構成を示す．支援システムは，自然語理解モデルで定める過程 3 を支援する F グラフ変換モジュールと，過程 4 を支援する階層的 DFD 変換モジュールの二つのモジュールから構成されている．システムへの入力はデータベース化された P，K グラフであり，出力は階層的 DFD である．階層的 DFD は，既存のデータフロー図エディタで使用可能なようにフォーマット変換して出力される．

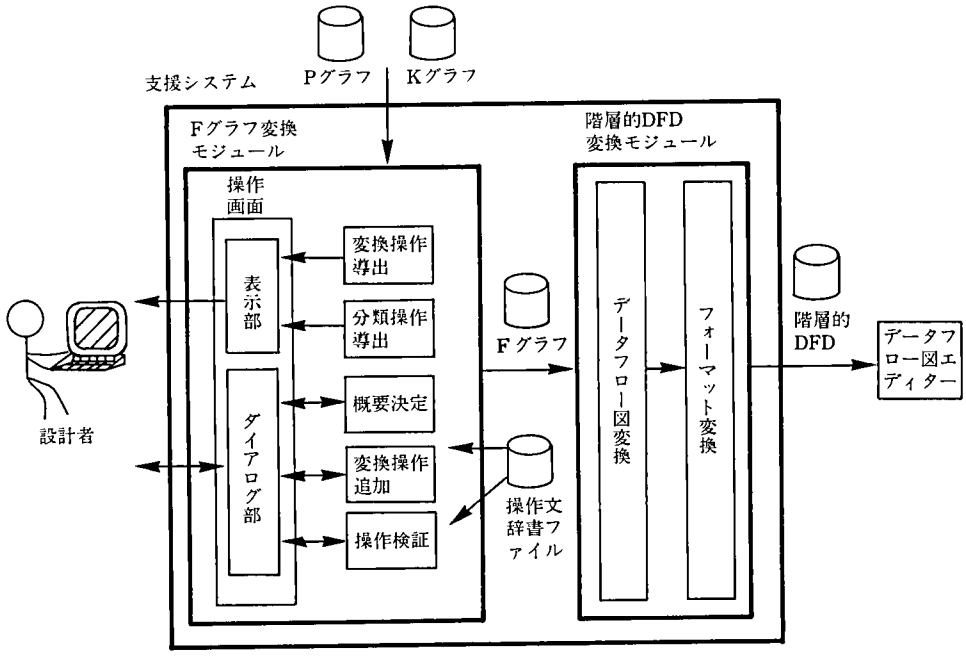


図 7　支援システムの概要

Fig. 7　The outline of support system

3.2　F グラフ変換モジュール

3.2.1　F グラフ変換モジュールの概要

F グラフ変換モジュールは 2.5 節で述べた P, K グラフから F グラフへの変換を機械的に支援するもので，概要決定，変換操作導出，分類操作導出，変換操作追加，操作検証の五つのサブモジュールと操作画面から構成されている．また，操作画面は表示部とダイアログ部の二つに分かれており，表示部には作成中の F グラフの内容が表示され，ダイアログ部においてシステムと設計者が対話を行う．F グラフへの変換は基本的にシステム主導で行われ，設計者がシステムからの質問に答える形で行われる．

システムによる F グラフ変換の手順を図 8 に示す．最初に最上位階層の F グラフの作成を行ったあと，下位階層の F グラフの作成を必要なだけ繰り返し行う．図 9 (a) に示す P, K グラフが入力として与えられた場合を例に，F グラフ変換の手順を説明する．

3.2.2　最上位階層の作成

ここでは，概要決定モジュールにより最上位階層の F グラフを作成する．最上位階層の F グラフは設計対象システムの処理概要を表すものである．システムは，まず P グラフの各木の根にあたる要素を，順次ダイアログ部に表示し，それがシステムの入力にあたるものか出力にあたるものか（または両方か）を設計者に決定させる．図 9 (b) では，電報の流れを入力に，処理結果を出力に決定している．次に，以下の 1) から 3) の手順により変換操作を決定する．

1)　出力要素候補の中から追加する変換操作の出力となる要素を選択する．

2)　その要素を出力するために必要な入力要素を入力要素候補の中から選択する．

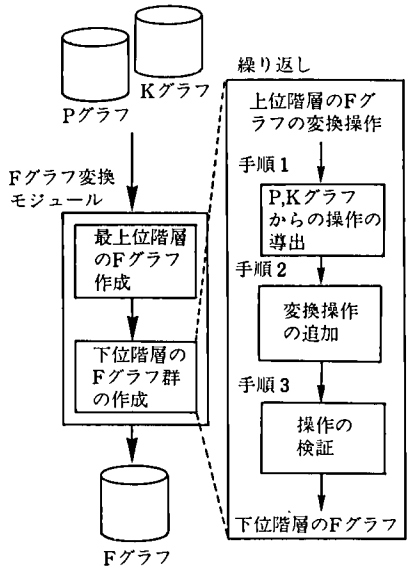


図 8　F グラフ変換手順

Fig. 8　Change process of F graph

3)　変換操作に名前を付ける．

図 9 (c)の例では，“電報の流れ”を入力として“処理結果”を出力する変換操作“電報解析処理”を決定している．

3.2.3　下位階層の作成

下位階層の作成は図 8 の手順 1 から手順 3 により行う．手順 1 はシステムが自動的に行い，手順 2，手順 3 はシステムと設計者の対話により行う．前節で決定した変換操作“電報解析処理”の下位の F グラフを作成する場合を例に各手順を説明する．

1)　手順 1：P，K グラフからの操作の導出 (自動処理)……ここでは，モデルで定められている規則に従って (2.5.1 項，2.5.2 項参照) 変換操作導出モジュールが P グラフ上の関係から変換操作を，分類操作導出モジュールが K グラフ上の関係から分類操作を自動的に導出し表示部に表示する．

ただし，システムでは以下の①から③の条件に従って，導出を行っている．

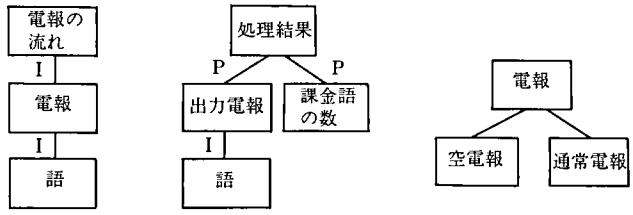
①　元の変換操作の入力要素のいずれかが K グラフ上に現われている場合，その分類関係から分類操作を導出する．

②　①の条件に該当しなかった場合で，元の変換操作の入出力要素のいずれかが I 集合体である場合，その構成関係から変換操作を導出する．

③　①，②の条件にともに該当しなかった場合で，元の変換操作の入出力要素のいずれかが P 集合体である場合，その構成関係から変換操作を導出する．

“電報解析”の例は条件 2 に該当するので変換操作“電報の流れから電報を取り出す”を導出し，表示している (図 9 (d))．

2)　手順 2：変換操作の追加……ここでは，変換操作追加モジュールが要素や変換操作の候補をダイアログ部に表示し，設計者がその中から適切なものを選択することにより F グラフに変換操作を追加する．



P グラフ　　K グラフ

(a) 支援システムへの入力

```
（システムの入出力を決定してください）
要素名＝電報の流れ
（この要素は？ 1＝入力 2＝出力 3＝入出力）
1
要素名＝処理結果
（この要素は？ 1＝入力 2＝出力 3＝入出力）
2
```

(b) 入出力決定のダイアログ画面

```
（入力要素候補）　　（出力要素候補）
1 電報の流れ　　1 処理結果
（出力要素を選択してください）
1
（入力要素を選択してください）
1
（変換操作名を入力してください）
電報解析
```

(c) 変換操作決定のダイアログ画面

```
（状態1　　分類　　分類結果　　変換操作　状態2）
電報の流れ ーーーーーーーーー取り出すーー電報

電報　　処理結果
```

(d) 変換操作導出後の表示画面

```
（入力要素候補）　　（出力要素候補）
1 電報 of 電報の流れ　　1 処理結果

（出力要素を選択してください）
1
（入力要素を選択してください）
1
（変換操作を選択してください）
1 電報から処理結果を作成する
2 電報を処理結果へ複写する
1
```

(e) 変換操作追加のダイアログ画面

```
通常電報から処理結果を作成する
（この操作の出力要素＝処理結果の構成要素である
課金語の数が，どの入力要素からも得られませんで
した．
課金語の数をこの操作の入力に追加します。よろし
いですか？ y or n ）
y
```

(f) 変換操作検証のダイアログ画面（1）

```
（課金語の数を出力する操作を決定します）
（入力要素を選択してください）
1 電報
2 課金語の数
1
（変換操作を選択してください）
1 電報を課金語の数へ複写する
2 電報により課金語の数を整形する
3 電報の課金語の数を合計する
4 電報により課金語の数を初期化する
3
```

(g) 変換操作検証のダイアログ画面（2）

＊斜体字は設計者の入力を表す

図 9　F グラフ変換モジュールの操作例

Fig. 9　Example of operations

変換操作の追加は以下の①～③の手順により行う．

① 出力要素候補の中から追加する変換操作の出力となる要素を選択する．

② その要素を出力するために必要な入力要素を入力要素候補の中から選択する．

③ 変換操作の候補の中から最も適切な変換操作を選択する．

変換操作の候補はシステムが操作文辞書ファイルの中から選び出す．操作文辞書ファイル（表2）は，入力要素の構成型と出力要素の構成型の組み合わせとその間に導入しうる変換操作を登録したファイルであり，事務処理システムを設計するのに必要十分と思われる変換操作が登録されている．また，このファイルで検証'要'になっている変換操作は，手順3の変換操作の検証の対象となる．

図9(e)の例では変換操作“電報から処理結果を作成する”を決定している．

表2 操作文辞書ファイル（一部分のみ）
Table 2 Operation dictionary file

検索キー		変換操作	検証
入力	出力		
I	P	(入力)から(出力)を取り出す	要
I	P	(入力)から(出力)を作成する	要
I	個	(入力)の(出力)を合計する	—
P	個	(入力)の(出力)を合計する	—
I	個	(入力)の(出力)を計算する	—

I=I集合体　P=P集合体　個=個体

3) 手順3：操作の検証……ここでは操作検証モジュールが，設計者の決定した変換操作のうち，検証の必要なものの検証を行う．その結果，不適当な部分があれば，それを設計者に知らせ，設計者はそれに応じて適切な修正を加える．手順2で決定された変換操作“電報から処理結果を作成する”の検証を行う場合を例に説明する．

システムは“処理結果”のPグラフと“電報”のPグラフを比較し，“処理結果”を作成するためには“課金語の数”が入力として不足していることを推測し，それを設計者に知らせる．ここでは設計者が入力として“課金語の数”を追加して変換操作の修正を行っている（図9(f)）．

また，この場合の“課金語の数”のように新たに状態1欄に追加された中間要素については状態2欄にも配置し，変換操作追加モジュールにより，その要素を出力する変換操作を追加する．図9(g)の例では課金語の数を出力する変換操作として“電報の課金語の数を合計する”を追加している．

3.3 階層的DFD変換モジュール

階層的DFD変換モジュールはデータフロー図変換モジュールとフォーマット変換モジュールの二つのサブモジュールから構成される．

データフロー図変換モジュールでは，階層的に構成された各Fグラフを階層的DFDの各DFDに自動変換する．FグラフからDFDへの変換は以下の方法で行う．

Fグラフの各操作をDFDのプロセスに置き換える．同時に入力要素をプロセスの入力データフローに，出力要素を出力データフローに，それぞれ置き換える．ただし，

システム全体の入出力(すなわち，各Pグラフの木の根)に当たる要素はファイル(データストア)に置き換える．

このとき，あるプロセスの出力データフローと別のプロセスの入力データフローが同じ名前を持っていれば，これを両プロセスの間を接続するデータフローに変換する．

フォーマット変換モジュールでは，データフロー図を既存のデータフロー図エディタで使用可能なフォーマットへ変換し，出力する．

4. 適 用 実 験

モデルおよび支援システムの有効性を確認するために二つの実験を行った．いずれの実験においても課題として与える要求仕様は誤りがなく，階層的DFDを作成するのに十分な規模である．また，被験者は大阪大学基礎工学部情報工学科2年生である．

実験1では，酒問屋の在庫管理問題の要求仕様を被験者である学生に与える[8]．各学生はそれを元にモデルに基づいてP，Kグラフを作成し，支援システムを利用してFグラフ，階層的DFDを作成する．被験者に対しては複合設計法，および，自然語仕様の理解モデルについて事前に説明を行っている．また，支援システムの使用方法の簡単な練習も行っている．ただし，それらを用いて実際に設計作業を行うのは全員初めてである．

実験2では，スーパーマーケットのレジを出入りする現金を管理するシステムの要求仕様を被験者である学生に与える．学生はモデル，支援システムを利用せずに要求仕様から階層的DFDを作成する．この実験の被験者は実験1の被験者とは異なる．被験者に対しては複合設計法についてのみ事前に説明を行っている．ソフトウェアの設計作業を行うのは全員初めてである．

5. 分　　析

実験1，および実験2で作成された階層的DFDに対して三つの観点から比較分析を行った．ここでは三つの評価尺度を簡単に説明した後，各々の分析結果について述べる．

5.1 評 価 尺 度

5.1.1 データフロー図の量的な均一性

モデルおよび支援システムを利用することにより個人差の少ないデータフロー図が得られることが期待できる．これを測るために，データフロー図の構成要素であるデータフロー，およびプロセスの総数を各学生ごとにカウントし比較を行った．

5.1.2 機能の段階的な詳細化

階層的DFDを構成する目的の一つは，設計対象システムで実現すべき機能を段階的に詳細化することにある．本稿では機能の詳細化はできるだけ段階的に行われるべきであると仮定し，これを測るために機能詳細化評価尺度を用いた[9][10]．機能詳細化評価尺度は階層的DFDをDFDパスツリーに置き換えて評価を行う[11]．

図10にDFDパスツリーの例を示す．図10(a)が元となる階層的DFDであり，図10(b)がそれに対するDFDパスツリーである．DFDパスツリーの各頂点は階層的DFDの各DFDに対応する．また，DFD-Aのプロセスdのように子DFDを持たない

プロセスは，そのプロセスとそこに出入りするデータフローのみからなるDFDがあると仮定して，DFDパスツリーに頂点として追加する．

また，DFDパスツリーの各頂点の数字はDFDパス(データがDFDに入ってから出ていくまでにたどる経路)の数である．たとえば，図10(a)のDFD-CのDFDパスは，d1→d3→d6, d1→d4→d5, d2→d5であり，その数は3である．このDFDパスの数が対応するDFDにおける処理（機能）の複雑さを表すものと仮定する．

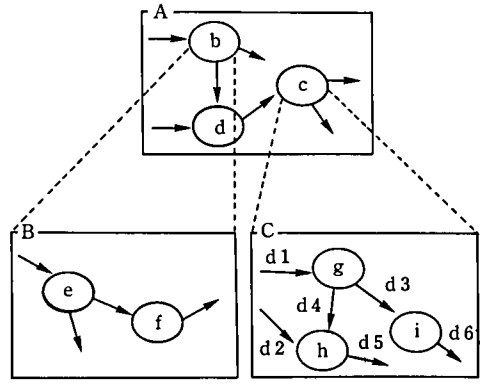
DFDパスツリー T に対する機能詳細化評価尺度 $R(T)$ は次式で表される．

$$R(T)=\left(\sum_{i=1}^{m} r_i\right)/m$$

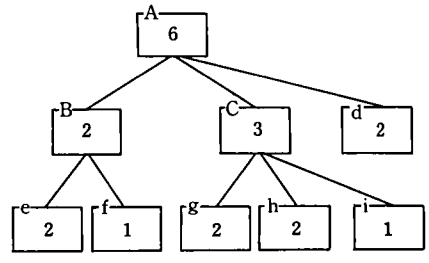
m：DFDパスツリー T の葉の数

r_i：DFDパスツリー T の根から葉 v_i までの経路上の頂点のDFDパス数の分散

ある階層のDFDにおいて機能の詳細化が急激に行われている場合には，$R(T)$ の値は大きくなる．したがって，$R(T)$ の値が小さいほど機能の詳細化が段階的に行われていると考えられる．



(a) 階層的DFD



(b) DFDパスツリー

図 10　階層的 DFD と DFD パスツリー

Fig. 10　The DFD path tree and HDFD

5.1.3　機能の均等な分割

データフロー図を階層的に構成するもう一つの目的は，設計対象システムで実現すべき機能をその抽象レベルから詳細レベルにわたって各レベルごとに分割して設計し，表現することにある．本稿では機能の分割はできるだけ均等に，すなわち，ある

DFDの子DFDの間で機能の大きさが大きく異なることのないように行うべきであると仮定し，これを測るために機能分割評価尺度を用いた[9][10]．この尺度も前述したDFDパスツリーに対して評価を行う．DFDパスツリー T に対応する階層的DFDにおける機能分割の評価尺度は次式で表される．

$$D(T)=\left(\sum_{j=1}^{n} d_j\right)/n$$

n：DFDパスツリー T の葉以外の頂点（内部頂点）の数

d_j：DFDパスツリー T の頂点 v_j の子の頂点のDFDパス数の分散

あるDFDの子DFDのうち，一つだけに機能が集中している（パス数が多い）場合には $D(T)$ の値は大きくなる．したがって，$D(T)$ の値は小さいほど機能分割が均等に行われていると考えられる．

5.2 分析結果

以上の評価尺度により実験1，実験2の結果の分析を行った．分析対象者数は実験1が55名，実験2が54名である．

5.2.1 データフロー図の量的な均一性

実験1，および実験2で作成された階層的DFDのプロセス数の度数分布を図11に，データフロー数のそれを図12に示す．横軸がプロセス数（データフロー数），縦軸が人数を表している．プロセス数の平均値は実験1が39.84，実験2が18.78であり，分散値は実験1が337.88で，実験2が31.28である．データフロー数の平均値は実験1が107.02，実験2が57.13であり，分散値は実験1が2055.62で，実験2が322.56である．

いずれの場合も実験1の分散値は実験2より大きくなっており，個人差が大きくなってしまっていることがわかる．

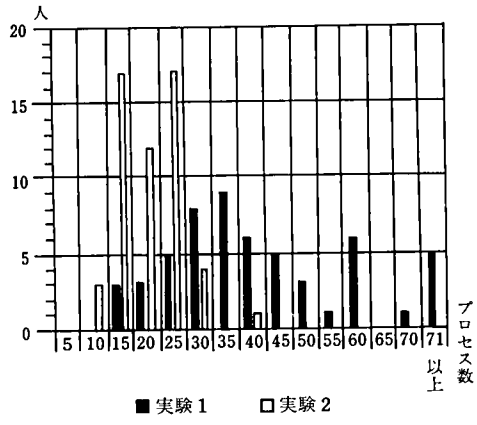


図 11 プロセス数の度数分布

Fig. 11 Frequency distribution of process

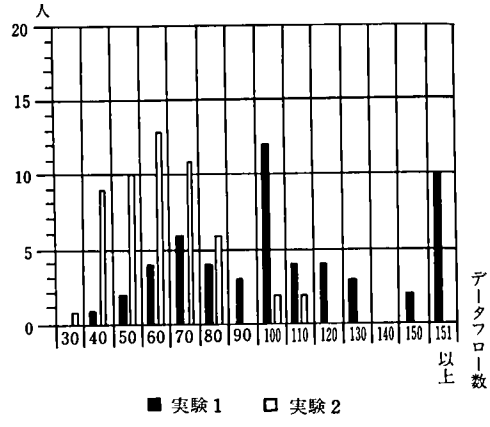


図 12 データフロー数の度数分布

Fig. 12 Frequency distribution of data flow

5.2.2 機能の段階的な詳細化

実験1，および，実験2で作成された階層的DFDに対する機能詳細化評価値の度数分布を図13に示す．横軸が機能詳細化評価値，縦軸が人数を表している．平均値は実験1が33.10，実験2が91.90であり，分散値は実験1が1890.44で，実験2が

15641.80 である．

実験 1 と実験 2 では，機能詳細化評価値の平均に約 3 倍の差があることがわかる．二つの実験の間で機能詳細化評価値に大きな差のあることは，図 13 において，機能詳細化評価値の最頻値（モード）が実験 1 では 10 であるのに対して，実験 2 では 100 超となっていることからもわかる．また，実験 1 の最頻値の度数は 24 で全体の 4 割以上である．

5.2.3 機能の均等な分割

実験 1，および実験 2 で作成された階層的 DFD に対する機能分割評価値の度数分布を図 14 に示す．横軸が機能詳細化評価値，縦軸が人数を表している．平均値は実験 1 が 3.03，実験 2 が 8.49 であり，分散値は実験 1 が 7.14 で，実験 2 が 131.49 である．

実験 1 と実験 2 では機能分割評価値の平均に約 3 倍の差があることがわかる．最大値も実験 1 では 15.40 であるのに対して，実験 2 では 62.88 である．また，図 14 より，機能分割評価値の最頻値（モード）は共に 5 であることがわかる．ただし，実験 1 における最頻値の度数は 48 で，データ全体の約 9 割を占めている．

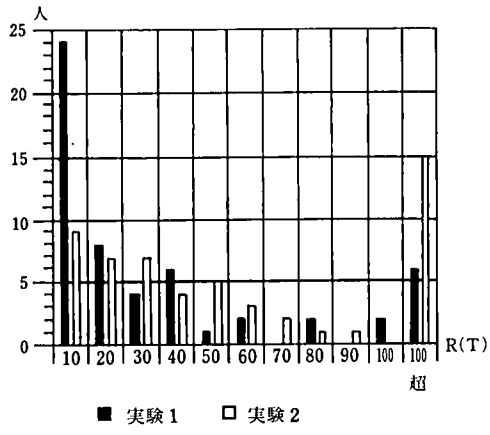


図 13　機能詳細化評価値の度数分布

Fig. 13　Frequency distribution of function detailization

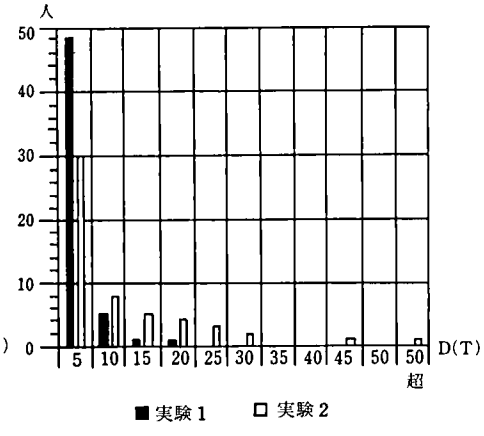


図 14　機能分割評価値の度数分布

Fig. 14　Frequency distribution of function division

5.2.4 分析結果に対する考察

今回の分析では，量的な均一性では支援システム（およびモデル）を利用した場合の方が利用しない場合よりも個人差が大きくなっている．これはモデルの過程 1，過程 2 を支援するシステムがないため，学生の作成した P，K グラフに個人差が出たことが原因と考えられる．システムでは P グラフに基づいて F グラフ (DFD) の階層化を行っているので，P グラフが異なると，階層的 DFD を構成する DFD の数が異なってくる．そのため今回のように単純にプロセス，データフローの総数を比較しただけでは個人差が大きくなってしまうと考えられる．

機能詳細化評価値，機能分割評価値の分析結果は実験 1（支援システムおよびモデルを利用した場合）の方が評価値が小さく，個人差も小さい（分散値が小さい）．このことから，モデルおよびシステムを利用することによって，設計の初心者でも機能を段階的に詳細化し，かつ機能を均等に分割することが可能になることがわかった．

この二つの評価値が良い（小さい）ということは，言い換えると，見やすい（理解しやすい)階層的DFDが得られているということである．設計対象システムで実現すべき機能を単に書き出すだけであれば，1枚のDFDにすべての機能（プロセス）を書けば事が足りるが，それでは設計対象システムが大きくなればなるほど，プロセス間のつながりが複雑になり，でき上がったDFDは理解しがたいものになってしまう．DFDを階層的に構成しなければならない最大の理由はここにある．しかし，DFDを階層的に構成する作業には明確なガイドラインがなく，設計者の経験に大きく依存しているため，設計作業に不慣れな者が理解しやすい階層的DFDを作成することは困難である．今回の分析結果から，モデルおよび支援システムがこの作業のガイドラインとして有効であることが確認できた．

6. おわりに

自然語仕様の理解モデルとそれに基づく支援システムを大学の学生実験に適用することにより，その有効性を実験的に確認した．今回の実験からモデルおよび支援システムが階層的DFDの機能の段階的詳細化と均等な分割に関して有効であることが確認できた．また，P，Kグラフの個人差が最終的な出力である階層的DFDの均一性に影響することがわかった．

今後は，階層的DFDの作成に要する工数や，階層的DFDからプログラムコードへの変換の容易さの点からも評価を行うことを検討している．また，理解モデルの過程1，および過程2（Pグラフおよび Kグラフの作成）の支援機能を支援システムに追加し，理解モデル全体を支援するシステムとすることも検討している．

本研究に関し御指導，御協力を頂いた奈良先端科学技術大学院大学情報科学研究科図書館長 鳥居宏次教授，ならびに大阪大学基礎工学部情報工学科 菊野亨教授，松本健一助手，楠本真二助手，同学科鳥居研究室の皆様に深謝の意を表する．

参考文献 [1] E. Yourdon and L. L. Constantine: "Structured Design", Prentice-Hall (1979).
[2] 大野浩史，菊野亨，鳥居宏次，"自然言語によるプログラム仕様に対する理解モデルの提案"，情報処理学会知識工学と人工知能研究会資料，AI-60，1988.
[3] 大野浩史，菊野亨，鳥居宏次，"抽象化に基づく文章理解モデルのソフトウェア設計法への適用"，情報処理学会ソフトウェア工学研究会資料，SE-62，1988.
[4] 大野浩史，菊野亨，鳥居宏次，"プログラム設計のための構造表現モデルの提案と複合設計への適用"，情報処理学会ソフトウェア工学研究会資料，SE-65，1989.
[5] 熊本厚志，大野浩史，菊野亨，鳥居宏次，"階層的データフロー図作成支援システムの試作"，ソフトウェア・シンポジウム論文集'91，1991.
[6] 大野浩史，"自然語仕様からのデータフロー図構成法"，ユニシス技報，第28号，1991.
[7] T. DeMarco, "Structured Analysis and System Specification", Prentice-Hall, 1978.
[8] 山崎，久保，鈴木，大野，"共通問題によるプログラム設計技法解説"，情報処理，25，9，pp. 934～945，1984.
[9] 西山，"データフロー図の階層構造から見た機能分割の適切さの評価"，大阪大学基礎工学部情報工学科特別研究報告，1992.
[10] 熊本厚志，松本健一，鳥居宏次，"階層的データフロー図作成支援システムの評価"，ソフトウェア・シンポジウム論文集'92，1992.
[11] 片岡欣夫，松本健一，熊本厚志，鳥居宏次，"構造化分析法における階層的データフローダイアグラムの不偏性評価尺度の提案"，電子情報通信学会知能ソフトウェア工学

研究会資料，KBSE 92-18，1992.

執筆者紹介　熊　本　厚　志（Atsushi Kumamoto）

1961年生．1985年岡山大学経済学部経済学科卒業．同年日本ユニシス(株)入社．金融機関向けシステムの開発，保守を担当．1990年大阪大学基礎工学部情報工学科受託研究員．現在，関西支社システム技術一部分散システム課所属．情報処理学会会員．

授業の開発
——教授目標の抽出と構造化

The Development of Instructions
——The Extraction and Structuring of Instruction Objectives

内田修市

要　約　教育内容を充実するためには，受講者の要求にそった教育の目標を掲げ，その目標にそった授業の計画（準備）・実施・評価を繰り返し行い，教育内容・指導方法を改善し，またそれらの教材等も精錬化させていく必要がある．

これら教育カリキュラムを作成・評価する上での基礎となる情報を提供するものの一つが教授目標および目標間の関連（教授目標ネットワーク）である．経験の豊富な教師は，精錬化された教授目標ネットワークを数多く脳裏に保持しており，それらを受講者の目的・状況別に切り替えながら，相対的に効果の高い授業を実践していることが想定できる．

この経験豊富な教師の脳裏に保持された教授目標ネットワークを，誰もが理解できる形式で表現できれば，多くの教師・生徒に多大の利益をもたらすことができる．

しかし教授目標ネットワークを構築・改善する方法は，未だ確立されていないのが現状であり，複数の教材を階層化する手法として，ISM (Interpretive Structural Modeling)法[1][2]が存在しているという状況である．

本稿では，教授目標ネットワークをもとに教育カリキュラムの作成から評価までの活動を行うべきという主張のもと，まず教授目標ネットワークとはどのようなものかを紹介し，教育カリキュラム全般にわたって教授目標ネットワークを活用する上での利点を述べる．

次に，誰もが教授目標ネットワークを構築・評価できるように，教授目標を記述する上での考え方を述べ，その考え方にそった教授目標の抽出方法・抽出された目標の構造化方法・評価の方法を紹介する．

Abstract　Besides determining its objectives which meet student requirements, the betterment of education requires the repeated planning (preparations), practice and evaluation of the objective-conscious instructions followed by improvement in what and how to teach as well as the refinement of teaching materials. It is instruction objectives and relationships between the objectives (instruction objective networks) that serve as basic information on which the preparation and evaluation of educational curriculums are based. Well-experienced teachers, with a refined abundance of instruction objective neworks in their minds, are assumed to be giving relatively more effective lessons, shifting them to satisfy students' own objectives and class situations. The description, understandable to everyone, of the instruction objective networks held by those who are rich in teaching experience could bring a great deal of benefits to both teachers and students.

In the meanwhile, no methods are available at present of building and bettering such networks but the interpretive structural modeling (ISM)[1][2] which helps construct teaching materials in hierarchical order. In favor of the claim that all activities ranging from the preparation of educational curriculums through their appraisal be based on instruction objective networks, this paper discusses what such

networks actually are and several advantages in the use of them in the making of curriculums. In order for anyone to create and evaluate teaching objective networks, the next mention is about some attention to be paid in describing objectives, how to extract appropriate ones, and how to structure extracted objectives and ways of evaluating them.

1. はじめに

教育内容を充実するためには，受講者の要求にそった教育の目標を掲げ，その目標にそった授業の計画（準備）・実施・評価を繰り返し行い，教育内容・指導方法を改善し，また使用する教材等も精錬化させていく必要がある[3]~[5]．

しかし，これら教育目標・内容の決定・改善は，教師の経験のみをもとに行われており，その妥当性に関する議論はあまり行われていない．

このため，教育内容の設定が曖昧であったり，妥当性が個人の尺度でなされたり，評価・改善活動が不明確なステップで実施されることになる．

教授目標ネットワークを明確にし，それをもとに教材を開発して，授業の計画・実施・評価を行えば，上記の多くの問題を解決する糸口をつかむことができる．

本稿では，教授目標ネットワークとはどのようなものかを紹介し，教育カリキュラム全般にわたって教授目標ネットワークを活用する上での利点を述べる．

次に，誰もが教授目標ネットワークを構築・評価できるように，教授目標を記述する上での考え方を述べ，その考え方にそった教授目標の抽出方法・抽出された目標の構造化方法・評価の方法を紹介する．

2. 教授目標ネットワークの概要

教授目標ネットワークとは，「何を（実施・理解）するには，何を（実施・理解）しなければならないか」を目標・前提の立場から表現したものである．一つの目標は，教授の対象となる内容領域と生徒に求める行動目標によって記述される．通常は，教科の章・単元ごとに設定する．

図1に算数の約分を実施するための教授目標ネットワークの例を示す．個々のノード内に記述されている文を「教授目標」と呼ぶ．個々の教材（テキスト・演習問題・OHP・等）は，この教授目標に対応づけて開発する．

階層構造の左右の広がりは教授対象の広がりを示し，上下は，包含関係と下位の能力が上位の能力の前提であることを表している．たとえば，「分数の既約分数を算出できる」ためには，「分数の既約分数とは何かを理解している」，「分子・分母の最大公約数を算出できる」，「分子・分母を最大公約数で除算できる」のすべての能力を組み合わせた能力が要求されることを意味している．

3. 教授目標ネットワークの有効性

教授目標ネットワークを作成することによる教師側から見た有効性を，以下に列挙する．

1)　教育内容全体・部分の関連を誰もが容易に把握できる．
（生徒・教師自身の理解促進）

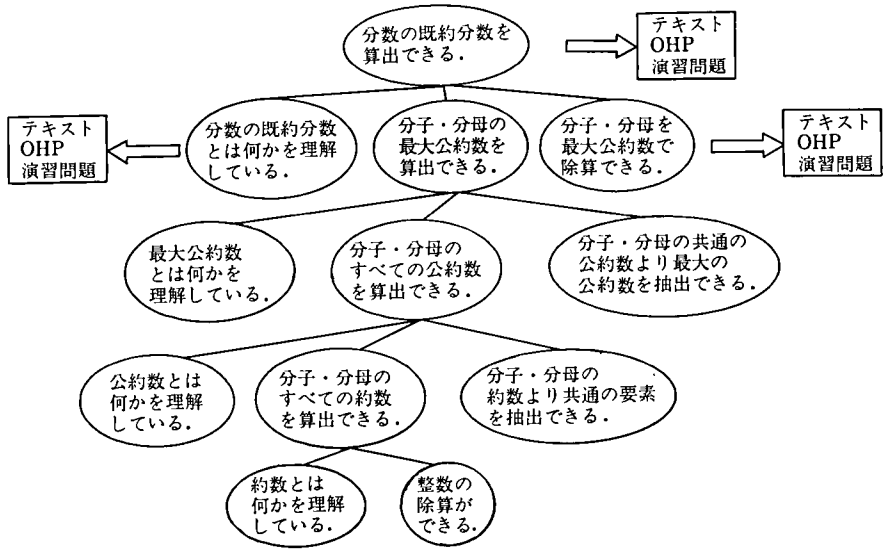


図 1　教授目標ネットワークの例

Fig. 1　Instruction target network example

2)　生徒に期待する必要な前提知識が明確になる.

3)　形成的評価*を利用し，履修課程の改善・改良を行うことができる.

4)　議論の際の共通の場を提供でき，その意志疎通が円滑になるので，共同して教授内容・履修課程の改善に当たれる.

5)　どの範囲をどのような順序で教授するかの手がかりを提供できる.

このように教授目標ネットワークは，授業開発全般にわたっての基盤となるものである.

4.　教授目標の考え方

本章では，図1に示した楕円の内側の記述（教授目標）の考え方とその表記方法について述べる．教授目標は，教授後の生徒の能力状態・到達レベルを測定可能かつ評価可能な形式で記述する.

教授目標の記述は，以下のように内容領域とその行動目標を含んだ形式で記述されるべきであると考える.

教授目標＝内容領域＋行動目標

教授目標は上記構成のもと，その記述内容から表1のように分類される.

4.1　教授目標の形式1の考え方

教授目標の形式1の内容領域部分には，学習の対象・分野・領域・範囲に関しての記述を行う．そのとき，総称名・名詞だけでなくそれらに付随する属性名を用いて領域を特定する必要がある.

たとえば，「分数」だけでなく「分数の既約分数」と記述する.

---

*　形成的評価(Formative Evaluation，形を造る)：学習指導の各段階において生徒の学習状態を把握し，生徒の到達済みの教授目標群と未到達の教授目標群を明らかにすることで，教師の指導・生徒の学習改善を測るための評価．生徒のランクづけのための評価ではない.

表1　教授目標記述
Table 1　Description of instruction target

	内容領域	行動目標
形式1	内容領域［を］	操作動詞［(が) できる］
形式2	内容領域［を］	理解している 知っている
形式3	内容領域［を］ ＋操作動詞［することに関して］ 内容領域［を］ ＋操作動詞[する(方法｜仕組み｜etc…)を]	理解している 知っている

操作動詞部分には，前に定義した内容領域に関して，どのレベルまでの行動の変化を望むかを記述する．その時，何らかの測定可能な操作動詞を用いて表現する．

たとえば，「分数の既約分数［を］＋算出できる」と記述する．なお，操作動詞の例は以下のとおりである．

比較できる	列挙できる	構築できる
識別できる	記述できる	構成できる
選択できる	描写できる	設計できる
区別できる	解釈できる	
区分できる	説明できる	質問できる
分類できる	定義できる	批判できる
分離できる	要約できる	指摘できる
配列できる	応用できる	指示できる
保存できる		決定できる
収集できる	計画できる	調整できる
合成できる	予測できる	発見できる
結合できる	測定できる	推論できる
計算できる	分析できる	認識できる
チェックできる	評価できる	関係づけ（れる）できる
確認できる	検証できる	対応づけ（れる）できる
同定できる		
示（せる）	一般化できる	
提示できる	公式化できる	
	正当化できる	

4.2　教授目標の形式2の考え方

内容領域の考え方は形式1と同様であるが，ここでは形式2での「知っている」と「理解している」の違いと，形式1・2の捉え方の相違に関して解説する．

まず，「知っている」と「理解している」の違いであるが，「知っている」は，内容領域の「意味」を理解せずに暗記している状態で，世に実現するものを連想記憶している場合を示す．たとえば，「1+2」の答は「3」であると他人から答のみを聞いて暗記しており，「2+1」の結果を尋ねると「解らない」場合である．一方，「理解している」は，内容領域の「意味」を理解している状態（内容領域の状態が認識可能で，頭の中である一定の操作を行える状態になっている）で，「1+2」の結果も，「2+1」の結果も，同様に「3」であることが解る状態を示す．

つぎに形式1と形式2の捉え方の相違に関して解説する．

形式1は，生徒が外界からの情報に反応し，他人が認識できる形で行動をとれるレベルまでを求め，形式2は外界からの情報に反応し，認知できる状態（行動は伴わない）までを生徒に求める．形式1･2とも同様の内容領域であれば，形式1の方が上位の教授目標となる．

なお，形式3は内容領域に操作動詞を含む場合で，考え方は形式2と同様である．

5. 教授目標ネットワーク作成手順

この章では，教授目標ネットワークを作成するための手順に関して概要を述べ，さらにその詳細を解説する．

5.1 教授目標ットワーク作成手順の概要

次の図2のStep 1～Step 7を満足解が得られるまで，繰り返し実施する．

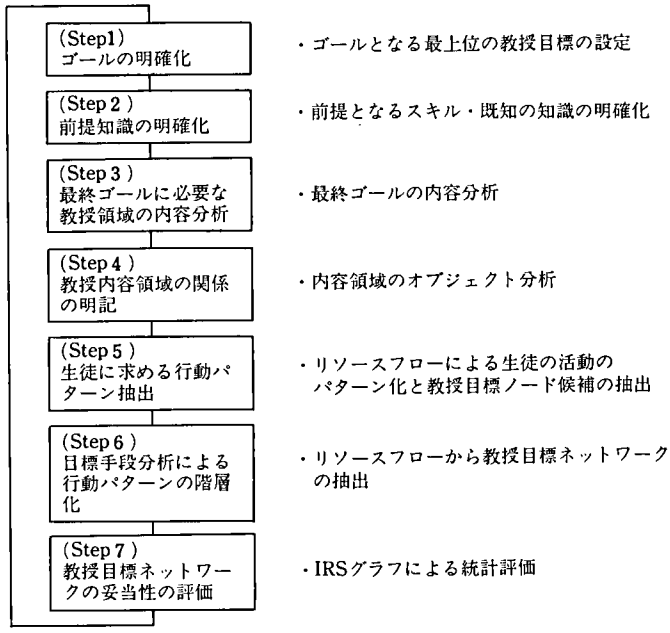


図 2 教授目標ネットワーク作成手順概要

Fig. 2 The outline of make an instruction target network

5.2 教授目標ネットワーク作成手順の詳細

5.2.1 ゴールの明確化（Step 1）

教授プロセスによって，最終的に達成されなければならない最上位の教授目標（ゴール）を明らかにする．これは，図1の「分数の既約分数を算出できる」に該当するもので，必要ならば複数選出してもかまわない．ただし，下位の領域を共有する場合のみである．構造が独立する場合は，教授の単元を分割する．

5.2.2 前提知識の明確化（Step 2）

教授に当たって，既知とみなす内容領域およびその行為･行動目標を明らかにする．

学習対象者の経験，職歴，既学習対象等を考慮して，平均的学習者のスキルを推定し，それを教授目標で表現する．ここで定義した教授目標は，教授の対象とはしない．

図3に図1の教授目標ネットワークを作成した際の前提知識の例を示す．

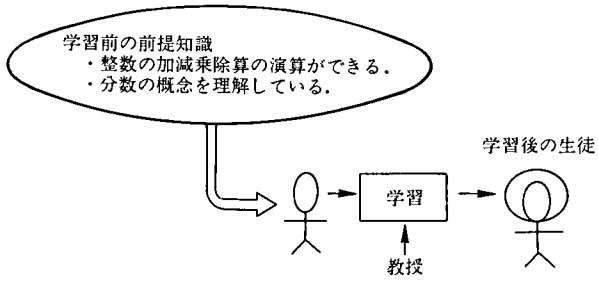


図 3　前提知識の例

Fig. 3　Example of premise knowledge

5.2.3　最終ゴールに必要な教授領域の内容分析（Step 3）

ここでは最終ゴールを理解するために必要な教授領域の内容を分析する．教授目標ネットワークの上位に行くほど，抽象度の高い用語で教授目標の内容領域が記述される可能性があるので，読者ごとに解釈が異ならないよう，ゴールに設定した目標記述の意味を明確に定義する．Step 2 で定義した前提知識を持つ学習者が理解できるレベルになるまで，用語の意味定義を繰り返し実施する．またこの Step 3 は，ゴールを達成するのに，どの解法を教授するかを決定することでもある．図4にその例を示す．

ゴールに設定した教授目標記述
　　分数の**既約分数**を算出できる．

用語の意味の繰り返し定義
　既約分数
　　分数の分子・分母を**最大公約数**で割って，簡単な分数にしたもの
　最大公約数
　　公約数の最大値
　公約数
　　いくつかの整数が共通に保持している**約数**
　約数
　　ある整数を割り切れることのできる整数
　分数
　　2整数の割り算の商
　　（整数・小数は分数の特殊形と見る）

図 4　用語の意味の繰り返し定義

Fig. 4　Repetition definition for meaning of key word

5.2.4　教授内容領域の関係の明記（Step 4）

このステップでは，オブジェクトモデル[6][7]を用いて教師がいかに教材を理解しているか，または生徒に教材をいかに教授するかを確定・明記する．さらに内容領域をオブジェクトモデル表現することで，ゴールを達成させるための既知・未知の学習領域が明確になる．さらに既知の領域との関連から，今回学習する未知の領域の位置づけを明らかにすることができる．

Step 3 で選出した用語をその意味を解釈しながら，表2の記法を用いて表現し直す．その際，オブジェクトモデルの整合性・妥当性を維持するために既知の用語を加

表 2　オブジェクトモデル記法

Table 2　Object model notation

表記法	意　味
子供 ― 扶養する ― 父親	1：1の関係　関係が理解しずらい場合は関係名を線上に補足
バイク ― 2 ― タイヤ	1：2の関係
子供 ― おもちゃ	1：多（0以上）の関係
車 ◇― ボディー / エンジン	構成関係（全体：部分）
生物 ― 動物 / 植物	分類関係（一般化：特殊化，抽象化：具体化）

味する必要がある場合は，その要素も加える．また加味した既知の用語の意味を定義する必要があれば，Step 3 に戻り設定しておく．

図 5 に，Step 3 より作成したオブジェクトモデルの例を示す．なお，図 5 の破線は既知の領域であることを示している．

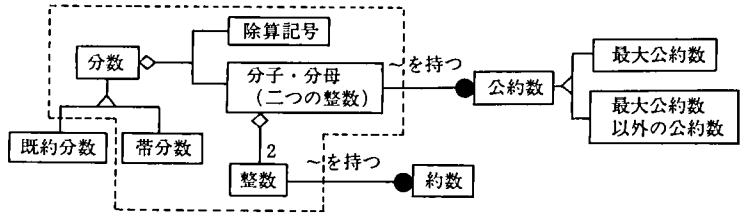


図 5　オブジェクトモデルの例

Fig. 5　Example of object model

5.2.5　生徒に求める行動パターン抽出（Step 5）

生徒にどのような行動過程を経て，ゴールとなる内容領域を生成させるかを表現するため，ここでは，リソースフロー図を用いて表現する．ここでのリソースフロー図は，資源を変換させるプロセス（資源変換行動・行為の総称名），資源を運ぶリソースフローの 2 要素を用いて記述する．

プロセス　：生徒の活動のプロセス（資源変換行動・行為の総称名）を，楕円で囲み表現する．

リソースフロー：プロセスを経て生産される資源およびプロセス実施に必要な資源はリソースフローの矢印で表し，矢印上には，資源の名前や総称名を記述する．

まずStep 3で設定したゴールの教授目標記述をリソースフロー図で表現する．次に，ゴールのリソースフロー図を順次詳細化していく．詳細化の観点としては，ゴールとなる内容領域の出力フローから順に前方向へとたどり必要なプロセスを決定し，入力フローを確定する．この作業をゴールでの入力リソースが出現するまで繰り返す．なおリソースフロー名の候補は，オブジェクトモデル内の用語より抽出することができる．

ここで抽出されたプロセスは，次ステップで検討する教授目標ノードの候補となる．図6にリソースフロー図の例を示す．

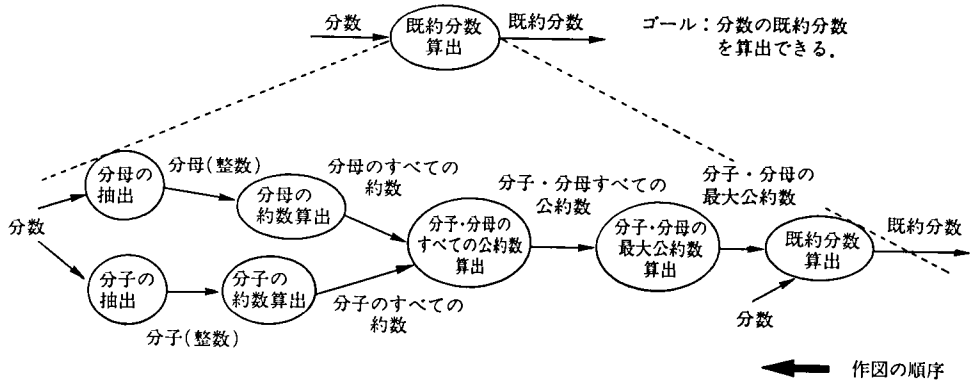


図 6　リソースフロー図の例

Fig. 6　Example of resource flow

5.2.6　目標手段分析による行動パターンの階層化（Step 6）

このステップでは，前ステップで抽出されたリソースフロー図から教授目標ネットワークへの変換を行う．

リソースフロー図で抽出されたプロセスの後方から前方向が，教授目標ネットワークのトップから下位方向に相当する．したがって，このステップではリソースフロー図を参照し，教授目標ネットワークをトップから下位方向へと構築していく．

まず，リソースフロー図の最終プロセスを教授目標ネットワークのトップノード(図7の1）にする．上位目標を達成させるために必要な下位目標をリソースフローの後方に記述されたプロセスより順次抽出（図7の2）する．これをリソースフローのプロセスがなくなるまで，繰り返し下位方向へと実施する．その際，以下の作業を平行して実施する．

1)　プロセス表現から教授目標表現への変換

2)　下位の教授目標の補足（図7の3，4)

（上位の目標を前提能力を利用して達成する必要がある場合，その補足をする）

図8は，上記の工程を繰り返して作成した教授目標ネットワークである．破線の四角で囲まれた部分は，下位の教授目標が同一なので，上位教授目標の内容領域をマージさせ，構造を単純化させる．

5.2.7　教授目標ネットワークの妥当性の評価（Step 7）

Step 6で作成した教授目標ネットワークの妥当性を評価するに当たって，以下の二

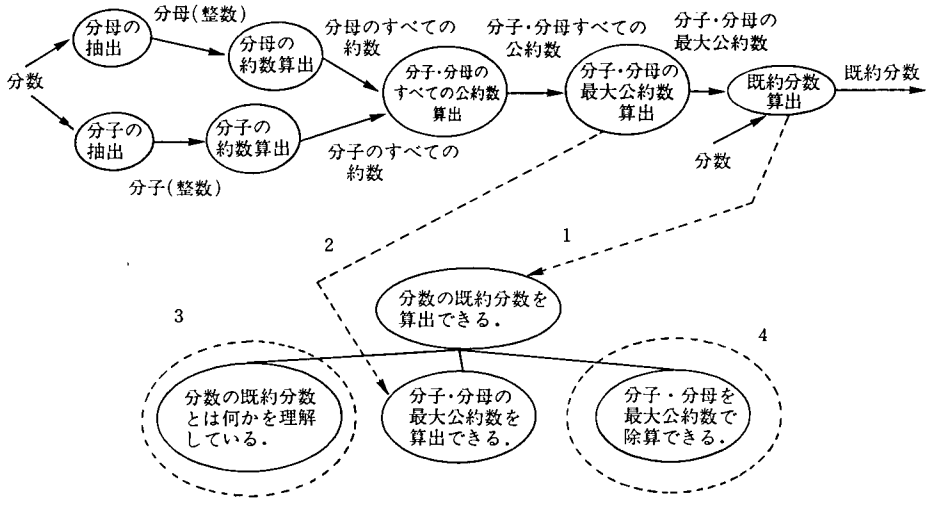


図 7　教授目標ネットワークへの変換例

Fig. 7　Example of change for instruction target network

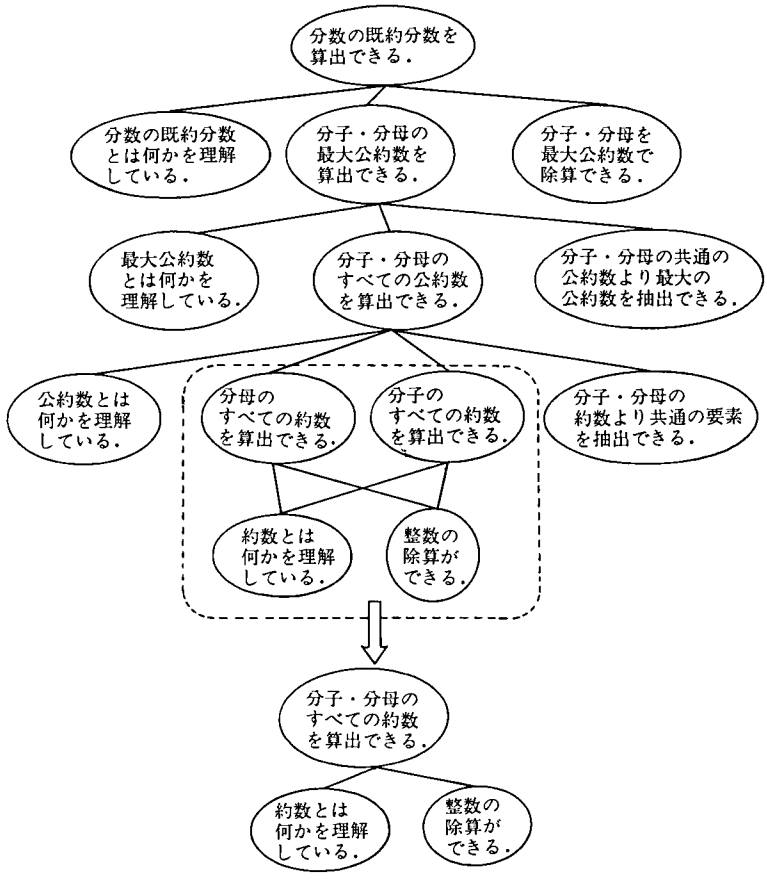


図 8　改訂後の教授目標ネットワーク

Fig. 8　Instruction target network of after revised

つの手法が利用できる．

1)　IRS (Item Relation Structure) グラフによる評価 [8]

2)　S-P 表 (Student-Problem) による評価 [9]

本稿では，IRS グラフを利用した評価の方法に関して述べる．S-P 表に関しては，参考文献を参照願いたい．

IRS グラフとは，学習者のテスト結果の部分的・断片的な関係を整理し，全教材間の関連構造を導き出したものを指す．

今回は，この IRS グラフの考え方を流用し，個々の教授目標に付随したテスト・アンケート等の計測結果をもとに教授目標ネットワークを評価する．

まず，教授目標の部分的な関係の妥当性を確認する方法を述べ，次に全教授目標間の関係を評価する方法に関して紹介する．

上位教授目標を j 下位教授目標を i とすると，その妥当性は以下のように評価できる．

$$\text{項目関連係数 } R_{ij}=1-\frac{\text{下位教授目標 } i \text{ が理解できずかつ 上位教授目標 } j \text{ が理解できている 生徒の率}}{\text{下位教授目標 } i \text{ が理解できない 生徒の率} \times \text{上位教授目標 } j \text{ が理解できた 生徒の率}}$$

この演算の結果，項目関連係数 R_{ij} が“1”のとき，教授目標 j が i より上位であることが妥当であると見なし，項目関連係数 R_{ij} が≦0 のとき，教授目標 j が i より上位であることは妥当でないと見なす．図 9 に教授目標間の上下関係が妥当なケースと妥当でないケースの例を示す．図 9 内で用いている“0”と“1”は，以下のように解釈する．個々の教授目標に付随したテストやアンケートを実施し，その結果，理解したと見なせる場合“1”と表し，理解できていないと見なせる場合“0”と表している．

妥当なケース

生徒	目標 i	目標 j
1	1	1
2	1	1
3	1	0
4	0	0
5	0	0

$R_{ij}=1-\frac{0}{0.4\times0.4}$

$=1$

妥当でないケース

生徒	目標 i	目標 j
1	1	0
2	1	0
3	1	0
4	1	0
5	0	1

$R_{ij}=1-\frac{0.2}{0.2\times0.2}$

$=1-5$

$=-4$

図 9　項目関連係数

Fig. 9　The item relation coefficient

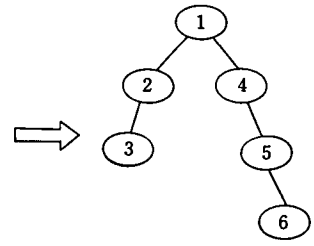
次に，全教授目標間の関係を導き出す方法に関して紹介する．まず，上記で示した個々の教授目標間の評価結果を図 10(a)にまとめる．

図 10(a)を上位教授目標 1 の列から縦方向に見ていくと，すべての下位教授目標に“1”が立っている．これは，教授目標 1 がすべての教授目標の上位（すなわち最上位の教授目標である）にあることが妥当であることを示している．

上位教授目標 j

下位教授目標 i	1	2	3	4	5	6
1	−	−.33	−.12	−.60	−.20	−.20
2	1.00	−	−.75	.0	−.50	−.50
3	1.00	1.00	−	−.20	−.80	−.80
4	1.00	0.	−.50	−	−.80	−.80
5	1.00	0.	−.40	1.00	−	.0
6	1.00	−.40	−.40	1.00	1.00	−

(a)



(b)

図 10　項目関連係数表と評価後の教授目標構造

Fig. 10　Item relation coefficient and instruction target network of after estimated

次に，上位教授目標 2 の列から縦方向に見ていくと，教授目標 3 の行と交わる部分のみが“1”になっている．これは，教授目標 3 は教授目標 2 より下位にあるべきであることを示している．

以上のようなアプローチで図 10(a)を分析すると，図 10(b)の教授目標ネットワークを作成することができる．これにより，教師は図 10(b)で作成された教授目標ネットワークと Step 5 で作成した教授目標ネットワークを比較し，妥当な教授目標ネットワークへ近づけていくことができる．

6. おわりに

教授目標の抽出方法・構造化方法・評価の方法に関して述べてきた．

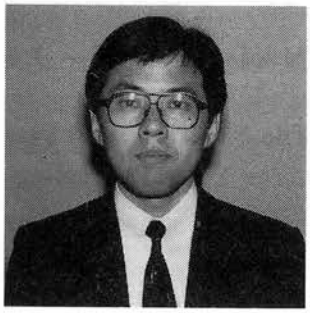
本手順により作成された教材を利用した，ICAI システムが現在開発され，評価・改良中である．この ICAI システムは，個々の生徒の理解度に応じて教授目標ネットワークを能動的に移動し，生徒の能力に見合った適切な教材を生徒に示しながら，生徒を誘導教育するものである．

今後の課題としては，本手順の評価・改良を行い計算機上に教材開発のための支援環境を構築することである．さらに，ICAI システムとも連動させ，誰でもが容易に教材開発できる環境を構築していきたい．

最後に本稿を作成する機会を与えて戴いた関西支社　教育部の丸山部長，ICAI システムの開発を推進して戴いた中新部長，本稿作成に際し有益な助言を戴いた伊東部長，橘田課長はじめ教育部内インストラクタ各位に感謝の意を表したい．

参考文献
[1] 赤堀，編著，“授業の開発” みずうみ書房，1987.
[2] 佐藤隆博，“授業設計と評価のデータ処理技法” 明治図書，1980.
[3] B. S. ブルーム，他，(梶田，他訳)，“教育評価ハンドブック”，第一法規，1973.
[4] 西之園春夫，“授業の過程”，教育学大全集 30，第一法規，1981.
[5] 橋本重治，“新・教育評価法総説”，上巻 金子書房，1976.
[6] J. ランボー，他，(羽生田，監訳)，“オブジェクト指向方法論”，トッパン，1992.
[7] P. Coad, E. Yourdon, “Object-Oriented Analysis” Press/Pretice-Hall, 1990.
[8] 竹谷誠，“IRS テスト構造グラフの構成法と活用法”，日本教育工学雑誌，1980.
[9] 佐藤隆博，“S-P 表の作成と解釈” 明治図書，1975.

執筆者紹介　内　田　修　市 (Shuichi Uchida)
1957年生．1980年九州産業大学経営学部 産業経営学科卒業．同年日本ユニシス(株)入社．1992年関西支社教育部，UNIX関連の客先教育を担当．情報処理学会会員．



NAP（音声データ処理）の機能と特徴

The Functions and Features of the Network Application Platform (NAP) for Voice Processing

三 輪 次 郎

要 約 NAP (Network Application Platform) は，デジタル交換機と音声データベースをAシリーズ・コンピュータに統合し，COBOL等の高級言語や4GLに電話機やファクシミリとの通信機能を提供するプラットフォームである．NAPによって音声やファクシミリイメージをAシリーズ・コンピュータで直接入出力することが可能となった．

DBMSやDCMSがそれぞれデータベースやデータ通信の領域で発揮してきた効果と同等のものを，VNMS (Voice Network Management System) が音声と音声ネットワークの処理に対して発揮する．VNMSはNAPの中核を成すソフトウェア群である．VNMSにより提供される豊富なコマンドライブラリによって，従来の数値や文字の処理と同等のプログラミング・ノウハウで音声処理アプリケーションが開発できる．

NAPを利用したシステムは，すでに米国においてはPacific Bell社などの電話会社で，またわが国においても日本経済新聞社の「日経テレ・ファックス」システムが稼働中であり，新しい情報処理メディアに対する市場のニーズとも相俟って多様な展開が考えられる．

Abstract The Network Application Platform (NAP), which serves to integrate a digital private branch exchange and a voice database into the A Series computer, is a platform to provide high-level languages including COBOL and fourth-generation languages with capabilities of communicating with telephone sets and facsimile equipment. It has made it possible for voices and facsimile images to be directly fed on the A Series products.

The Voice Network Management System (VNMS), the repertoire of software which is the core of NAP, provides for the processing of voices and voice networks the same effects as DBMS and DCMS have had on that of databases and data communications respectively. Its rich command library easily leads to the development of vocie processing applications with the same amount of programming know-how as required for the conventional processing of numerics and texts. Practical systems for which the NAP is being used are already seen at Pacific Bell Corporation in the United States and in the 'Nikkei Tele Fax' system by Nihon Keizai Shimbun in Japan, both operational at present. Also spurred by market needs for new information processing media, the NAP is considered to develop into a variety of new applications.

1. はじめに

電話とファクシミリ，これらは今日最も広く使用されている通信機器といえる．現在，日本国内で電話回線は5000万回線，ファクシミリ設置台数は400万台を数え，廉価版，携帯型等種々の多様化，高機能化とも相俟ってさらに普及すると予測されている．

また，単なる人対人の通信手段としてだけでなく，機械化されたシステムとの通信手段としての利用が可能となってきており，その利便性が今後ますます高まるととも

に現代社会のインフラストラクチャとしての重要性を一層高めるものと思われる．

電気通信事業法の施行により通信事業にも競争原理が導入された．今日，通信事業は原則的には誰でもが参入できる事業である．この事業で成功するには当然他事業者との差別化が求められる．通信料金の差だけでなく他事業者にないサービスの提供が求められる．

電話やファクシミリのサービスで一層の競争力を得られる企業は通信事業者だけではあるまい．公共企業体，金融，流通，種々のサービス業等「エンドユーザは一般大衆」と言える事業では，電話やファクシミリを活用したシステムにより，サービスの向上，業務の効率化・省力化が考案されうる．さらには，この最も普及した双方向の通信媒体，電話を活用することで需要を喚起し企業の飛躍的な発展に結び付けるような戦略的事業展開を図ることも可能ではなかろうか．

本稿では，ネットワーク・アプリケーション・プラットフォーム (Network Application Platform, 略称：NAP)，を構成するハードウェアとソフトウェアの機能を述べ，簡単な音声メールシステムを想定して NAP における音声処理の動作について解説する．

NAP を十分理解するには多方面に亘る技術分野の知識が必要となる．電話機と電話網，交換機の知識，汎用コンピュータを取り巻くデータベース，データ通信知識，音声系の通信プロトコルから端末機器技術基準適合認定申請等の法令に至るまで関連知識の枚挙に暇がない．本稿が NAP を理解する上での一助となれば幸甚である．

2. NAP とは

NAP は，交換機と EDP のハードウェア，ソフトウェアからなる基盤システムであり，NAP 上で動作するアプリケーションレベルプログラムに電話交換の機能と通話の端点としての機能を与えるものである．

2.1 稼 働 環 境

図 1 にネットワーク全体から見た NAP の稼働環境を示す．NAP の稼働環境は電話網とともにある．NAP システムの設置者が通信事業自身であれ，電話網の加入者であれ，概念的にはネットワーク内部に存在するシステムと見なせる．ネットワークを利用するシステムというより，ネットワークの一部であってネットワークに高度な付加価値を与えるシステムととらえることは，適用業務を考案するに当たってのヒントとなる．

2.2 適 用 業 務

NAP は，NIU (Network Interface Unit) と AP (Application Processor) から成る複合システムである．NIU により電話網を構成する種々の電気通信設備・端末機器とのインタフェースを取り，AP によりサービスアプリケーションを実行する．サービスアプリケーションは，その論理に基づき NIU を制御することで多種多様な付加価値通信処理，業務処理を行う．

NAP の適用業務は，処理過程の観点から次の 3 種に分類できる．

1) 呼制御のサービス (Call Control Services)……中継処理の過程で交換システムが (NAP のような) サービス処理系に対し制御を求める．サービス処理系は，交

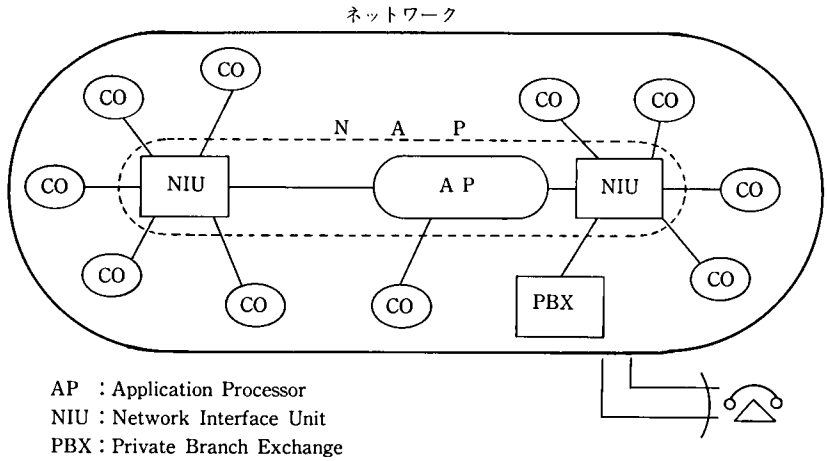


図 1 NAP 稼働環境概念

Fig. 1 NAP Conceptual operation environment

換システムが呼の確立に必要な処理を加える．呼の確立自体を目的とする付加価値中継処理である．

例としては，クレジットカード払いの通話における精査業務，情報料課金代行業務，ディレクトリサービスに類するものが挙げられる．

2) 発信者対話サービス (Caller Interaction Services)……サービス提供のために交換システムが (NAP のような) サービスノードに接続を行い，サービスノードと発信者の間にエンドツーエンドの呼が設定される．

サービス例としては，音声/FAX メールシステム，音声応答システム等が挙げられる．

3) 複合サービス (Hybrid Services)……前述 1) と 2) の複合サービスである．中継の過程で処理系が関与し，発信者との対話または中継交換機との情報交換を行う．対話により得られた情報に基づき，処理系自信が呼の端点となったり中継点となったりする．

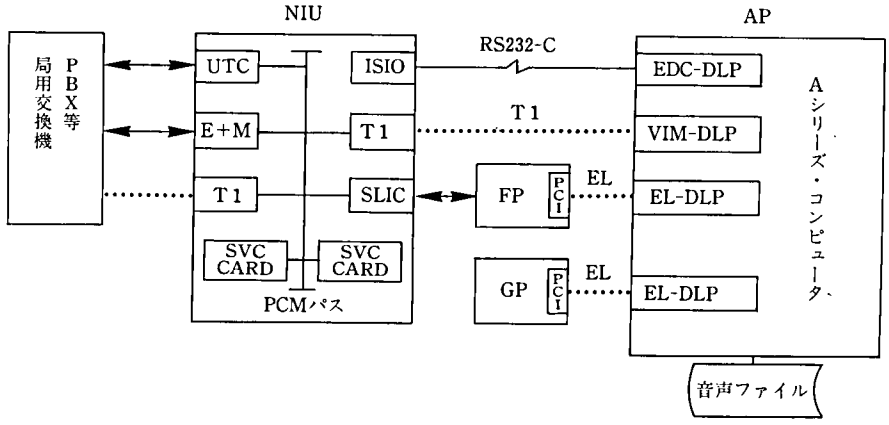
3. ハードウェア構成

図 2 にプラットフォームを構成するハードウェア要素を示す．プラットフォームは，① NIU 制御のためのデータ通信インタフェース，② デジタル音声のための T1 インタフェース，③ アナログ音声のために SLIC, E+M, UTC カード，④ パソコンとの高速データ転送のための Express Link を総合的に制御，管理するものである．

3.1 NIU (Network Interface Unit)

NIU は，AP の制御に基づいて動作し，局用交換機 PBX，あるいは内線電話機とのインタフェースの機能（インタフェースファシリティ）を提供する．また，物理的電気的インタフェース機能だけでなく音声認識，電話会議等，ハイレベルの付加機能（サービスファシリティ）も NIU 機能として備えている．

これらの NIU ファシリティをアプリケーションプログラムが相互に接続制御し，



UTC ：Universal Trunk Card
E+M：Ear and Mouth Card
T1 ：T1 Card
ISIO ：Integrated Serial I/O
SLIC ：Subscriber Line Card
SVC CARD：Service Card
FP ：Fax Processor
GP ：Graphics Processor
PCI：PC Interface Card
EL ：Express Link
EDC-DLP ：Enhanced Data Comm-Data Link Processor
VIM-DLP ：Voice Interface Module-Data Link Processor
EL-DLP ：Express Link-Data Link Processor

図 2　NAP ハードウェア構成
Fig. 2　NAP hardware component

多種多様な付加価値通信を提供する．

3.2　FP (Fax Processor)

FP は AP と NIU 間に位置付けられ，AP から転送されてくる TIFF 3 フォーマット* の FAX イメージファイルを G 3 ファクシミリに向けて送信する機能を持つ．また，G 3 ファクシミリから受信する FAX イメージを一時蓄積し，通信完了後 AP に転送する機能を持つ．換言すれば，CCITT 勧告 T. 4，T. 30 および V. 27，V. 29 プロトコルを実装し AP にファクシミリ送受信機能を与える外付けプロセサである．

3.3　GP (Graphics Processor)

G 3 ファクシミリ機の解像度は，3.85 本/mm（標準解像度）または 7.7 本/mm（オプショナル解像度）であり，画素数は 215 mm（A 4 サイズ）当たり 1728 画素である．A 4 判 1 ページ当たり約 180 万ビット（標準解像度）または 370 万ビット（オプショナル解像度）の情報量となる．この大量のビットマップ情報をホストシステムの CPU で直接扱うのはコストパフォーマンス面で問題がある．単純だが甚大なインストラクション数が必要となる．GP はこの処理をホストシステムの CPU からオフロードするために開発された．

NAP 納入先事例である日本経済新聞社では，情報表現形式にビデオテックス向けに標準化された NAPLPS (North American Presentation Level Protocol Syntax) を採用している．GP はホストシステムで NAPLPS コード化された文字画像情報を

* Tagged Image File Format Type-3, イメージファイル交換用に使用されるデファクトスタンダードの一種

ビットマップ展開し，圧縮操作を施した後，TIFF 3 フォーマットでホストシステムに返送する外付けプロセサである．

3.4 EL（Express Link）

FP/GP と AP 間は EL で接続される．EL はバイトパラレルの入出力チャネルであり，最大 3 M Byte/sec.の転送能力を持つ．1 本の EL 上には最大 12 台の FP/GP をディジーチェーン形式で接続することができる．

EL の採用により FAX イメージのような大量の情報（圧縮後 50 KB/ページから 400 KB/ページ）を FP や GP のような外付プロセサとホストシステム間で高速に転送できるようになった．

3.5 T1 トランクと VIM-DLP（Voice Interface Module-Data Link Processor）

NAP システムの大きな特徴の一つに汎用コンピュータによる音声の蓄積と入出力の機能を挙げることができる．デジタルコード化された音声は MCP(Master Control Program，A シリーズ標準 OS）制御下のデータファイルとして A シリーズ・コンピュータに蓄積され，システム内の情報資源として利用することが可能となった．VIM-DLP は T 1 インタフェースを実現し，A シリーズ・コンピュータに音声の入出力機能を与える．

T 1 とは，主としてデジタル交換機相互間の伝送路として北米を中心に使用されている伝送方式であり，1 回線当たり 1.544 Mbps の容量を持つ．TDM 方式により 64 Kbps の PCM 音声チャネル 24 本が多重化される．VIM-DLP 1 セットで 1 本の T 1 トランクが接続される．

VIM-DLP では，PCM 音声の入出力機能に加え，以下の付加機能が実現されている．

1) 32 Kbps および 16 Kbps への圧縮/伸長の機能
   録音処理時でのオプションとして動的に選択可能である．
2) プッシュボタン信号（DTMF-Dual Tone Multi Frequency）を検出して文字コードへの変換を行う．
3) 自動音量調節機能．
   入力音量が−6 dB に満たない時−6 dB まで増幅する．

4. プラットフォーム・ソフトウェア

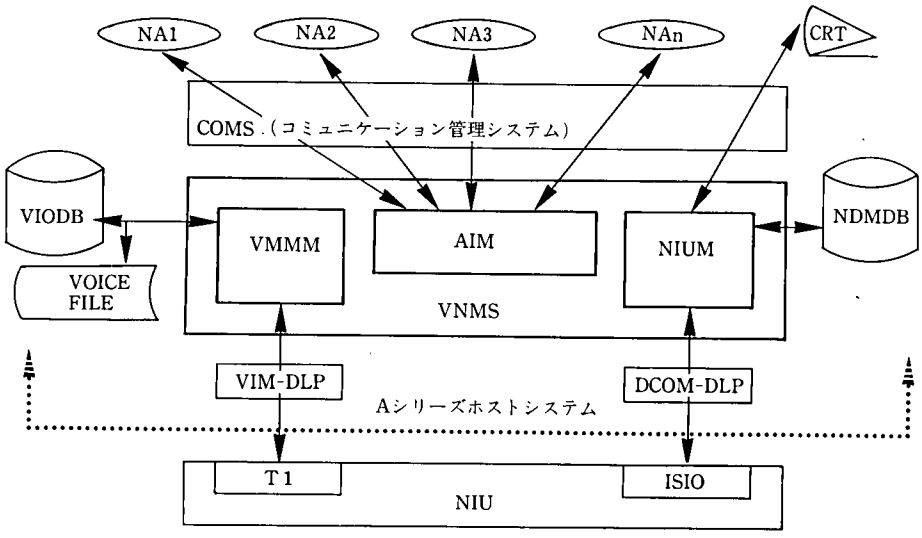
NAP を構成するハードウェアとして，A シリーズ・コンピュータ，NIU，FP，GP を紹介してきたが，本章ではプラットフォームを構成するソフトウェアについて述べる．

4.1 音声ネットワーク管理システム

音声ネットワーク管理システムとして VNMS（Voice Network Management System）が提供される（図 3）．

VNMS の機能によって業務処理プログラムは，低位の物理的インタフェース特性やハードウェア構成から独立し，抽象化されたコマンド/レスポンスを VNMS と交換することで所与の業務処理を実現することができる．

1) アプリケーション・インタフェース・モジュール（AIM）……コミュニケーショ



VIODB：音声入出力データベース
VOICE FILE：音声ファイル
VMMM：音声メッセージ管理モジュール
AIM：アプリケーション・インタフェース・モジュール
NIUM：NIU制御モジュール
NDMDB：ネットワーク構成定義データベース

図 3　音声ネットワーク管理システム
Fig. 3　Voice network management system

ン管理システム（COMS）経由で業務処理プログラム（NA）とプログラム間通信を行い，コマンド/レスポンスの組み立て，分解および経路付けを行う．

2) NIU 制御モジュール（NIUM-Network Interface Unit Manager）……NIU の管理を行う．NIU と AP 間のデータ通信リンクを介し NIU に対するコマンド発行および NIU からのレポートの解釈を行う．NIU とのコマンド/レポートの交換に基づき以下の処理を行う．

① 呼の状態管理．

② ネットワーク構成定義データベース（NDMDB-Network Data Model Data Base）のアクセスおよび保守．

③ VIM-DLP へのオンフック（終呼：電話機を置く操作），オフフック（起呼：電話機を取り上げる操作）指示．

また，NAP 運用管理者に CRT による会話型メニュー形式でのインタフェース機能を提供する．

3) 音声メッセージ管理モジュール（VMMM-Voice Message Management Module）……T 1 チャネルの音声入力操作および音声ファイル保守を行う．

4.2 音声ファイル

NAP システムでは，音声は PCM（CCITT 勧告 G. 711（μ-law）準拠）または ADPCM（CCITT 勧告 G. 721 準拠）フォーマットでホストシステムに蓄積される．こ

れを音声ファイル（VOICEFILE）という．音声ファイルへのアクセスキーはDMS IIデータベースに格納される．これを音声入出力データベース（VIODB）という．音声ファイルと音声入出力データベースは，1ホストシステムで一対であり唯一つである．

音声ファイル中の音声のアクセスは，それを録音した時の単位となり，個々の単位音声をメッセージ（MESSAGE）という．メッセージの録音時に圧縮率を指定することが可能である．圧縮率は個々のメッセージの属性として音声ファイルに格納される．したがって，音声ファイル内に種々の圧縮率のメッセージが混在することがあり得る．

利用可能な圧縮率と密度は，以下のとおりである．

① CCITT勧告G.711準拠
　1：1（圧縮なし）……64 Kbit/秒　　28.8 MB/時
② CCITT勧告G.721準拠
　1：2（1/2圧縮）……32 Kbit/秒　　14.4 MB/時
③ 米国ユニシス社
　1：4（1/4圧縮）……16 Kbit/秒　　7.2 MB/時

音声ファイル中のメッセージには，個々のメッセージにユニークな番号が与えられる．これをメッセージ番号という．メッセージ番号はそのメッセージの録音時に音声ネットワーク管理システムVNMSによって付与され，業務処理プログラムに通知される．

業務処理プログラムはメッセージ番号を指定することで，再生を行うことができる．

音声ファイル中のメッセージの絶対アドレスとメッセージ番号の対は音声入出力データベースに保持される．

4.3 ネットワーク構成定義データベース

ネットワーク構成定義データベース（NDMDB）にネットワーク全体の構成定義を行う．

ネットワーク構成情報として定義される要素には以下のものがある．

① 業務処理プログラム
② NIU
③ VIM-DLP
④ リソースグループ
⑤ ポートグループ
⑥ 着信呼経路テーブル
⑦ 発信呼経路テーブル

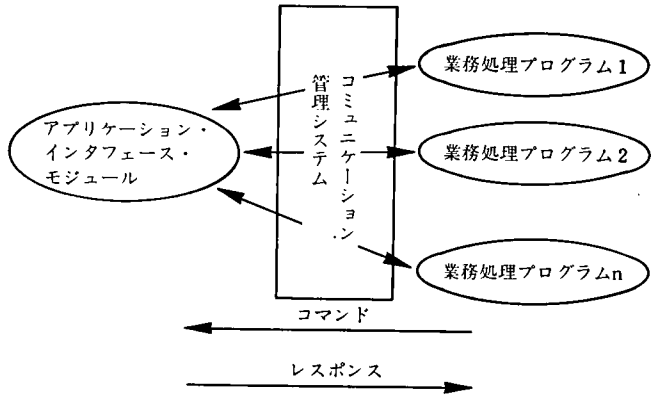
ネットワーク構成定義データベースに定義される情報としては，上記ネットワーク要素の他に業務処理プログラム単位に指定可能な変換論理がある．音声ネットワーク管理システムVNMSでは，アプリケーション開発の単純化と業務処理プログラムのシステム間での過般性を高めるためにシステムリソースの物理的識別子（たとえばポートアドレス）と業務処理プログラムによる参照値を独立させている．ネットワーク構成定義データベースには相互の変換のためのマッピング情報も含まれる．

5. アプリケーション・インタフェース

5.1 コマンドとレスポンス

業務処理プログラムは，アプリケーション・インタフェース・モジュールとコマンド/レスポンスを交換しながら業務処理を行う．コミュニケーション管理システムCOMS制御のTP対TP通信方式（COMSの上位でトランザクション処理を行うアプリケーションプログラム（TP）が他のTPとプログラム間通信を行うための方式）が使われる．

業務処理プログラムからアプリケーション・インタフェース・モジュールに向かうフレームがコマンドと定義され，逆がレスポンスと定義される（図4）．



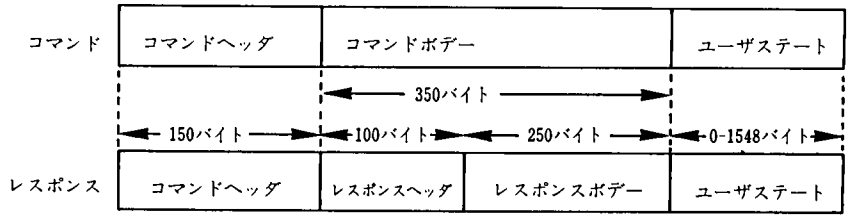


図4　コマンドとレスポンス

Fig. 4　Command and response

コマンドは，ヘッダ部，ボデー部，ユーザステート部から成る．レスポンスには，コマンドのヘッダ部とユーザステート部が変更なしに返される．ユーザステート部は業務処理プログラムが任意に定義できるため，通常のCRT等データ通信端末を制御するトランザクション処理プログラムに比し，業務処理プログラムは極めて単純な構造となる．端末との対話の状態等，処理の途中経過を自己の管理する作業メモリ（ターミナルテーブル等）に保持する必要がなく，替りにユーザステート部を利用すれば良い．トランザクション処理の制御に必要となる状態変数と状態遷移要因が1回の入力トランザクション中にすべて表示されているものとなる．とくにアプリケーションレベルの負荷分散のために業務処理プログラムの同一コピーを複数本同時に多重実行する時，各業務処理プログラムで共通に参照すべき作業領域を排他制御する必要がなくなる．

また，コマンド/レスポンスは，電話あるいはデジタル交換機に固有の専門知識を有

しない者でも容易に理解できるよう単純化が図られている．業務処理プログラム開発の生産性は音声ネットワーク管理システム VNMS の設計目標の一つである．

主なコマンド，レスポンスを以下に列挙する．

コマンド

Enable Application	業務処理プログラムの初期化完了通知
Send Voice Message	音声再生
Delete Voice Message	音声削除
Connect Call	接続
Terminate Dialog	会話終了
Collect Digit	PB 桁収集
Repeat Poll	アイドル状態通知
Initiate Call	発信
Terminate Call	切断
Get Message Numbers	メッセージ番号リスト要求
Get Voice Message	音声ファイルからのメッセージ取り出し
Create Voice Message	外部メッセージの音声ファイルへの追加
Send From File	外部メッセージの再生
Pivot Call	出力ポート切り替え
Get Message Attributes	メッセージの属性要求

レスポンス

Call Connected	接続完了
Message Sent	再生完了
Message Received	録音完了
Message Deleted	メッセージ削除完了
Voice Discontinued	再生中割り込み
Voice Error	再生時誤り発生
Delete Error	削除中誤り発生
Execution Error	パラメタ等に誤り
Command Executed	実行完了
Message Created	外部メッセージ追加完了
Command Rejected	コマンドプロトコルエラー
Attributes Obtaind	メッセージ属性報告
Incoming Call	着信報告
Start Application	業務処理プログラム開始報告
Poll Application	アプリケーション・インタフェース・モジュールのアイドル状態報告
Application Request	業務処理プログラムコマンド受付け可能

5.2 業 務 処 理

本節では，以上のコマンド/レスポンスが実際の処理においてどのように交換されるのかを述べる．説明のために，簡単なボイスメールシステムを考える．図5に全体シーケンスを示す．

1) 業務処理プログラムの起動……NAP 運用管理者が業務処理プログラム名を指定し起動する．起動命令は NIU 制御モジュールに対して CRT 端末から行う．NIU 制御モジュールは，指定された名称の業務処理プログラムについてネットワーク構成定義データベースを検索し，種々の属性を得た後，アプリケーション・

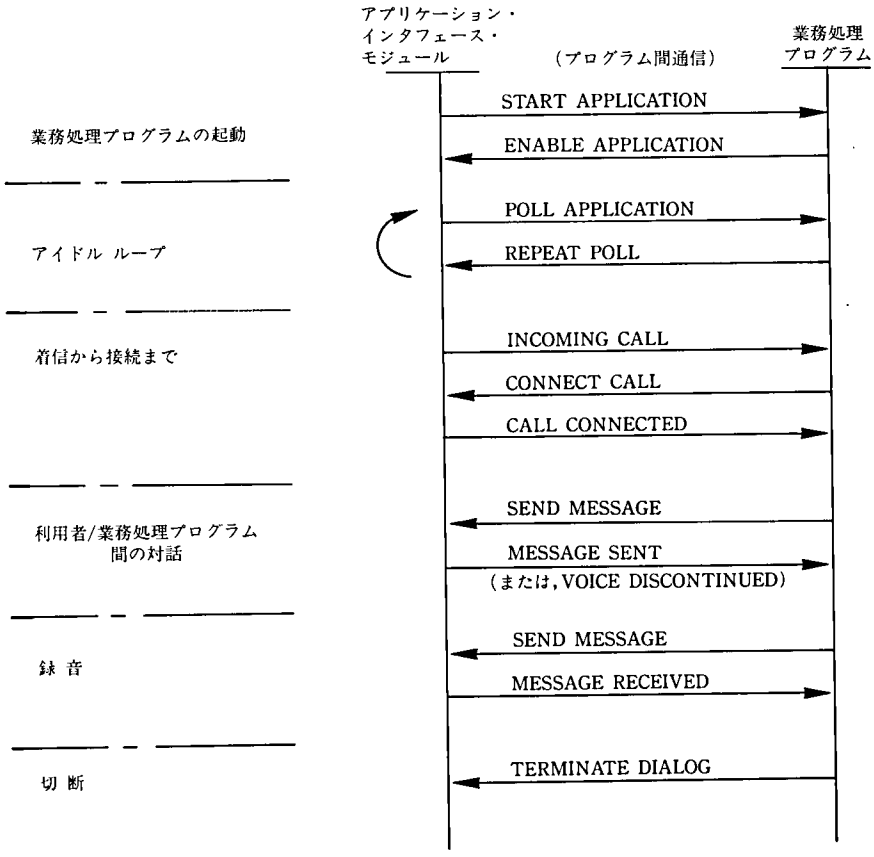


図 5　コマンド・レスポンスシーケンス例

Fig. 5　Example of command and response sequence

インタフェース・モジュールに渡す．アプリケーション・インタフェース・モジュールは，TP 対 TP の通信に必要なウィンドウ名，アジェンダ名を指定し，コミュニケーション管理システム COMS に対して START APPLICATION レスポンスを発行する．コミュニケーション管理システム COMS の機能により業務処理プログラムが実行を始める．

2)　アイドルループ……着信あるいは発信の必要性等，状態に変化がない限り，アプリケーション・インタフェース・モジュール――業務処理プログラム間は POLL APPLICATION レスポンスと REPEAT POLL コマンドを所定のインターバルで交換しあう．「レスポンス」という用語から違和感があるが，アプリケーション・インタフェース・モジュール――業務処理プログラム間のプロトコルはアプリケーション・インタフェース・モジュール主導のポーリング方式である．業務処理プログラムは常にアプリケーション・インタフェース・モジュールからの何等かのレスポンス（たとえば POLL APPLICATION）の受領を契機にコマンドを発行する．

3)　着信から接続まで……利用者が伝言を残すために NAP システムに電話をかけると，NIU のあるポートに呼び出しがかかる．NIU は制御リンクを介し AP へ着

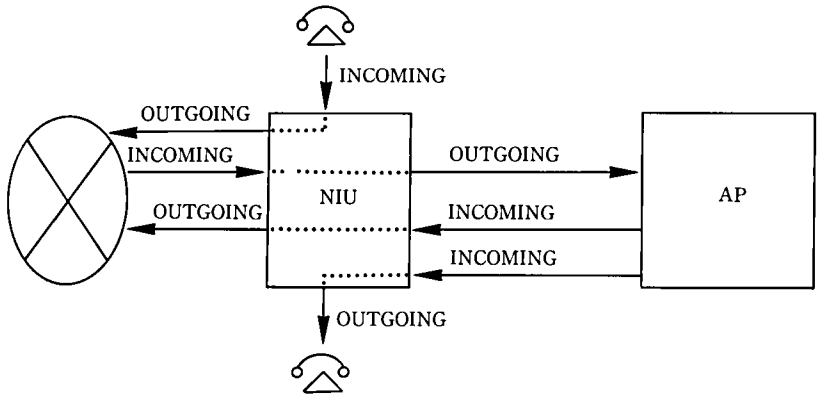


図 6　呼の方向
Fig. 6　Direction of calls

信報告を行う．

呼の方向の用語について説明する（図 6）．

呼の方向は NIU を中心にみて，NIU に着信したポートを INCOMING PORT，NIU から発信したポートを OUTGOING PORT と言う．ホスト主導型システムに馴染んだ者にとっては，ホストに入る方向を「イン」，ホストから出る方向を「アウト」と考えがちだが，「交換」を一つの主機能とする NAP システムにおいては概念を改める必要がある．この例では，利用者から着信した INCOMING PORT と VIM-DLP の接続された T 1 トランクとを接続するのであるから，業務処理プログラムは T 1 トランク側を OUTGOING　PORT として指定した CONNECT CALL コマンドを発行するものとなる．

業務処理プログラムから CONNECT CALL コマンドを受けとった音声ネットワーク管理システム VNMS は，自己の管理するリソースの中から利用可能な VIM-DLP ポートと T 1 ポートを割り当て NIU/AP 間の中継呼を確立する．

これで利用者の電話機と業務処理プログラムの間に通話路が確立された．VNMS は，当通話路に ID を与える．これをダイアログ ID と言う．ダイアログ ID は当通話路が解放されるまで保持され，当通話路上の以降のコマンドとレスポンスのヘッダ部に常に表示される．

4)　利用者/業務処理プログラム間の対話……業務処理プログラムは，CALL CONNECTED レスポンスの受領により通話路の確立を知る．ここからエンドツーエンドの対話が可能となる．まず，「こちらは，XX ボイスメールサービスです．ID 番号を入力してください．」といった音声ガイダンスを出力し，利用者から ID 番号を期待することを仮定しよう．

業務処理プログラムは，SEND MESSAGE コマンドを発行する．コマンドフレーム中に音声ガイダンスのメッセージ番号を指定し，音声メッセージ管理モジュールに音声ファイルからの読み出しと VIM-DLP への送出を指示する．1 回の SEND MESSAGE コマンドで 50 個のメッセージ番号をリスト指定でき，これらの音は間断無く再生される．

音声処理業務システムの設計においては，メッセージ番号とその音との関連(どのメッセージ番号で，どんな音声が再生されるか）を保つ機構を考慮することが必要となる．通常，この関連はDMS IIまたはISAMによって実現され，レコード様式は次のようなものとなる．

項目1——メッセージ番号（単項目または配列）

項目2——プロンプト・コード

項目3——音の意味

項目4——その他の属性（録音日時等）

メッセージ番号は，一録音音声に対しNAPシステムが与えるユニークな番号である．また，一録音音声に対しユーザが指定する番号をつけたい時，その番号をプロンプト・コードと呼ぶ．

たとえば，株式に関わる音声の登録の場合，銘柄名（変更が考えられるため世代管理が必要）の音声自身をメッセージ番号，銘柄コードをプロンプト・コードで管理する，などである．

なお，メッセージ番号とプロンプト・コードとを関連付ける機構を，プロンプトDBと呼ぶ．

業務処理プログラムがSEND MESSAGEコマンドを発行するとき，そのコマンドフレームの中に再生音を指定すると同時に，利用者からの応答方法を指定する．指定可能な応答方法について以下に概説するが，これらを理解する上で必要な前提について述べる．

(前提1) 業務処理プログラムの発行するコマンドとアプリケーション・インタフェース・モジュールからのレスポンスは1対1であり，また完全な同期が保たれる．コマンドの先送りは，エラーレスポンスで応答される．

(前提2) 音声ネットワーク管理システムVNMS内に通信中の各電話機からの入力プッシュボタン音声(PB)用バッファが存在する．入力されたすべてのプッシュボタン音声は，業務処理プログラムの状態とは無関係に一旦このバッファに格納される(プッシュボタン音声はEBCDICコードに変換され，文字としてプッシュホンボタン音声用バッファに格納される)．

プッシュボタン音声バッファのパージ，バッファ内プッシュボタン音声の取り出しは，業務処理プログラムの裁量に任される．

プッシュボタン音声応答仕様を定めるパラメタを以下に列挙する．

① DIGIT RULE

利用者からのプッシュボタン音声入力桁数を定める機能で，次のレパートリがある．

「Single」…業務処理プログラムは一桁のプッシュボタン音声応答を受領する．

「Count」…業務処理プログラムは指定桁数のプッシュボタン音声応答を受領する．固定桁数のプッシュボタン音声入力に使う．

「Delimiter」…業務処理プログラムは指定プッシュボタン音声が入力されるまでのプッシュボタン音声を受領する．可変長桁数のプッシ

ュボタン音声入力に使う．

「None」…業務処理プログラムはプッシュボタン音声入力に応答しない．応答にプッシュボタン音声は含まれない．

② BREAK LIST

利用者が音声再生を中断させることのできるプッシュボタン音声キー種別を指定する．利用方法に慣熟した利用者は，既知の音声プロンプトを最後まで聞いてからプッシュボタン音声を入力するよりも，音声プロンプトに先行する形でプッシュボタン音声を入力する．また，間違って入力したプッシュボタン音声によって再生されているプロンプトを最後まで聞くよりも中断して再入力する．

業務処理プログラムは，音声再生コマンド中にブレーク信号の機能を持たせるプッシュボタン音声を指定することができ，レスポンスでブレークの発生を知ることができる．コマンド発行時にすでにプッシュボタン音声バッファ中にブレーク信号として指定されたプッシュボタン音声がある場合，音声の再生出力は一切行われず，即座にレスポンスが返されることとなる．

もちろん，ブレーク機能を使用せず必ず最後まで再生させることも可能である．ブレークによってSEND MESSAGEコマンドが終了した場合，レスポンスはVOICE DISCONTINUEDとなる．

③ TIME LIMIT

プッシュボタン音声入力待ち時間の上限値を指定する．

利用方法に不慣れな利用者は，自分が次に入力すべきプッシュボタン音声がシステムにとってどのようなものか忘れる．または迷ってしまうことがある．このようなとき，一定の時間，通常5秒後位に直前の音声プロンプトを繰り返す．または，より丁寧なプロンプトを自動的に再生する．

業務処理プログラムはコマンド中にプッシュボタン音声入力待ち時間の上限を指定することができ，指定時間内にプッシュボタン音声入力が無かったとき，タイムアウトであった旨のレスポンスを受領できる．タイムアウト応答を受けとった業務処理プログラムは，直前のプロンプトまたは直前のプロンプトのより詳しいものを再度送出する．

④ DELIMITER LIST

DIGIT RULEがDELIMITERである時有効であり，デリミタとして機能すべきプッシュボタン音声キー種別を指定する．

業務処理プログラムが，数量や金額のように可変長のプッシュボタン音声入力を期待するとき利用する．音声ネットワーク管理システムVNMSは当該プッシュボタン音声入力があるまで利用者からの入力を受付け，入力のあった時点で業務処理プログラムに応答をかえす．

デリミタプッシュボタン音声は，”#”キーを標準として使用する事が通例である．

⑤ PURGE DIGIT

音声ネットワーク管理システムVNMS内の入力済プッシュボタン音声バッ

ファをクリアする．

VNMS内に入力プッシュボタン音声用バッファがあるために，利用者は音声プロンプトを聞く前に先行してプッシュボタン音声を入力することができる．一方業務処理プログラムとしてはバッファ内のプッシュボタン音声を順次処理するばかりでなく利用者との間で同期点を設定することが必要となる時がある．たとえば，例外処理に分岐するようなとき，業務処理プログラムは例外処理となったことを通知する音声プロンプトの再生コマンドでPURGE DIGITをセットし，VNMS内のプッシュボタン音声バッファをクリアし，利用者に例外発生時点からのプッシュボタン音声入力を促す．

業務処理プログラムは，以上のプッシュボタン音声入力仕様を状況に応じて選択し対話を進めることとなる．SEND MESSAGEコマンドによる音声プロンプトの再生とMESSAGE SENTまたはVOICE DISCONTINUEDレスポンス中に返される利用者からの入力プッシュボタン音声の解釈は，業務処理プログラムのプロセスの大部分を占めるものとなる．

さて，電話利用者は，正しく加入者IDを入力し伝言を残す処理を選択したものとする．

5) 録　音……録音の処理は，音声の再生コマンド（SEND MESSAGEコマンド）の1オプションとして実現されている．再生コマンドフレーム中に再生処理に続いて録音処理を行うか否かの指定を行う．ここでの再生メッセージは通常300ミリ秒程度の信号音とし，利用者に「録音始め」の契機を与えるものとするのが通例である．

録音されたメッセージは，音声メッセージ管理モジュールにより直接音声ファイルに格納される．MESSAGE RECEIVEDレスポンスフレーム中にメッセージ番号が表示される．

6) 切　断……業務処理プログラムからの切断は，TERMINATE DIALOGコマンドにより行う．利用者側からの切断は，レスポンスヘッダ中に表示される．レスポンスヘッダに切断表示があった場合，業務処理プログラムはTERMINATE DIALOGコマンドを発行する．TERMINATE DIALOGコマンドの発行を以てダイアログは消滅する．

6. 音声ファイル保守

前章では，NAPとNAP上で実行される業務処理プログラムによる音声の入出力処理について述べた．音声の入出力（録音，再生）は音声ファイルに対して行われる．音声ファイルはシステムで唯一つであり，録音・再生のためのアクセスは音声メッセージ管理モジュールが一括して行う．

次に，音声ファイル中の音声（メッセージ）の保守方法について述べる．

6.1 抽出・投入・削除

音声の録音と再生が，業務処理プログラムからのコマンドに基づき音声ネットワーク管理システムVNMSによって処理されるのと同様に，個々のメッセージを音声ファイルから外部ファイルへ抽出する処理，逆に，投入する処理もまた業務処理プログ

ラムからのコマンドにより実行される．GET MESSAGE コマンド，CREATE MESSAGE コマンド,，DELETE MESSAGE コマンドがそれぞれ抽出，投入，削除の機能を持つ．

抽出処理の概念を図 7 に示す．

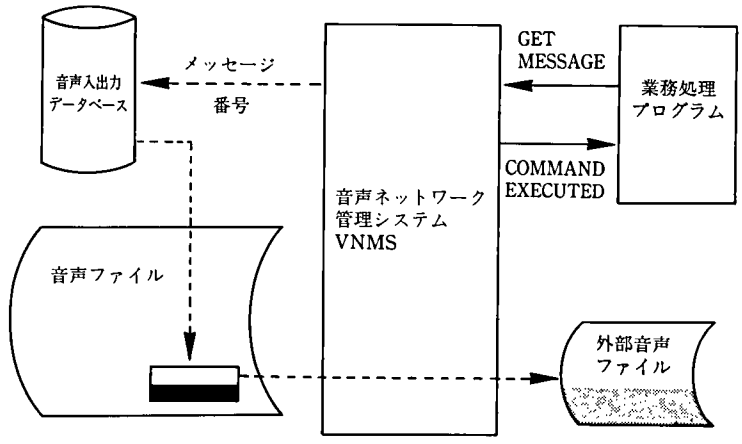


図 7　メッセージ抽出

Fig. 7　Message extraction

業務処理プログラムは GET MESSAGE コマンドに，抽出したいメッセージ番号と抽出後の外部ファイル名を指定する．VNMS は，指定されたメッセージを音声ファイルから読み出し，指定されたファイル名で生成する．外部ファイル名は，50 文字以内で任意に指定できる．

外部音声ファイルの音声ファイルへの投入では，業務処理プログラムは CREATE MESSAGE コマンドにおいて，外部ファイル名を指定し VNMS に読み取らせる．VNMS は，音声ファイル内に格納の後に，新しく付与されたメッセージ番号をレスポンスに表示して返す．

音声ファイル中のメッセージの削除では，業務処理プログラムは削除コマンドにメッセージ番号を指定する．

6.2　デジタル音声フォーマット

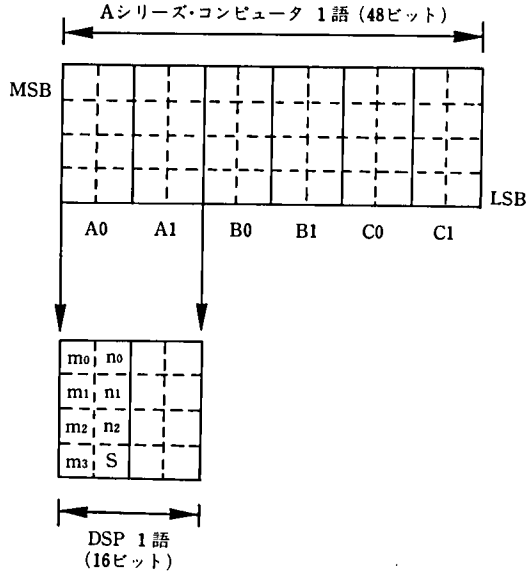
外部音声ファイルには次の情報が含まれる．

① 圧縮率

② 音声長のバイト表示

③ デジタル音声

ここで，圧縮率 1：1 のデジタル音声は A シリーズ・コンピュータ 1 語内で右図の構成を取る．



音声のレベル（強さ：X）は，次式で得ることができる．

$$N=2^2\cdot\bar{n}_2+2^1\cdot\bar{n}_1+2^0\cdot\bar{n}_0$$

$$M=2^3\cdot\bar{m}_3+2^2\cdot\bar{m}_2+2^1\cdot\bar{m}_1+2^0\cdot\bar{m}_0$$

$$X=(-1)^s[(M+16.5)\cdot 2^N-16.5]$$

VIM—DLP に採用された DSP（Digital Signalling Processor）は，16 ビット/語であり，上位 8 ビットと下位 8 ビットが反転している．A シリーズ・コンピュータ 1 語中でも DSP 1 語は，反転のままである．したがって，次系列上で A シリーズ・コンピュータ 1 語中の各バイトは，次のように再生される（A 1 から C 0 方向）．

A 1—A 0—B 1—B 0—C 1—C 0

←——6/8000 (SEC) ——→
再生時間

音声レベル（X）を時間上にプロットすることにより近似的なアナログ波形を求める事ができる．適当な区間における X の絶対値の累計値により有音部/無音部の判定が可能となる．

圧縮率 1：2 または，圧縮率 1：4 のデジタル音声のフォーマットついては今後の調査課題である．

6.3 プロンプトの録音

NAP システムの出力する音声は以下のようなカテゴリに分類する事ができる．

1) 利用者の録音した音声
   伝言等
   「利用者メッセージ」と呼ぶ．
2) 利用者の操作を勧誘するガイドとしての音声
   メニューの紹介，操作の誤りの指摘等
   「ガイダンス」と呼ぶ．
3) 再生時に動的に組み立てる音声
   数値，日付などの要素となる音（いち，に，ひゃく，びゃく，かようび等）
   「フラグメント」と呼ぶ．

ガイダンス，フラグメントについては，体系化し適当なコードシステムを設定することが必要となる．コード体系を確立した上で録音システムを構築する．

ガイダンス，フラグメントの録音に際しては，録音された音声が高品質であること，とくにフラグメントについては本体となる音の前後に無音や雑音がないことが望まれる．また，システムの初期導入時に行う録音では，バッチ処理的な録音方法が望まれる．初期録音工程の一例として，次の手順が提案できる．

① 保守の時期となっても録音の依頼が可能なアナウンサの選定
② 保守の時期となっても利用可能な録音スタジオの選定と確保
③ 次のフォーマットでの音声テープの作成
   [無音] [プロンプトコードのプッシュボタン音声] [無音] [本体の音] [#トー

ン］（繰り返し）

④ 録音用業務処理プログラムによる音声ファイルへの投入（図8）

ここで重要な事は，システムと源音再生機器（図8では，テープレコーダ）間を専用回線で接続し，録音音質の均質化を図る事である．公衆アナログ網を経由すると接続の度に物理的な経路が変わることになり，録音レベル，雑音レベルの変動が避けられない．再生機器を遠隔地に設置せざるを得ない場合は音声級の専用回線，システムと同一宅内に設置可能な場合は，NIU 直結の内線電話インタフェースの使用が勧められる．

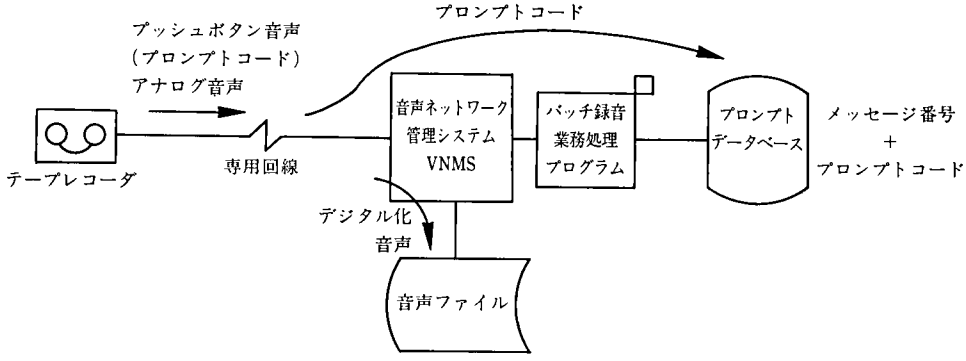


図 8 バッチ録音システム

Fig. 8 Voice recording system

6.4 プロンプトの確認および編集

録音されたプロンプトを検証し，とくにフラグメントについては本体音の前後にある無音や雑音を除去する必要がある．

日本経済新聞社向けに開発した編集システムでは，G 711 フォーマットのデジタル音声について以下の機能を具体化した．

① 指定プロンプトコードの連続再生機能

② 前部音声の削除機能

③ 前部への無音部追加機能

④ 後部音声の削除機能

⑤ 後部への無音部追加機能

編集業務処理プログラムは CRT からプロンプト・コードを入力し，それぞれのプロンプトに対して，指定された長さの切り取りと無音の付加を行った後，これらを連結して再生する．長さの指定は，理論的には 8000 分の 1 秒単位（1：1 圧縮の場合）まで可能であるが，実務上は 10 ミリ秒単位程度で十分である（人間の耳での識別範囲を越える）．

6.5 スタンドアロンシステムによる音声ファイル保守

前述のように NAP システムで直接音声編集システムを構築することは可能であるが，パソコンやワークステーションを利用することも一案としてあげられる．音声ボードを搭載するパソコン，ワークステーションが利用可能となってきており，ソフトウェアにより音声波形をスクリーン上に表示し増幅，減衰機能を含む編集機能を提供

している．スタンドアロンシステムで編集を行い，ファイルトランスファによって音声をデジタルのままNAPシステムへアップロードすれば品質劣化のない音声編集システムを構築できる．

7. おわりに

本稿では，NAPの構成要素とそれらの機能およびアプリケーションとのインタフェースを概説した．

今後，基礎技術の進歩と市場のニーズに伴ないネットワークのマルチメディア化，インテリジェント化はますます進展するであろう．プッシュボタン式電話機はもとよりワープロ，パソコン，ファクス等は，すでに一般家庭への普及もみられ，高度情報化社会の礎となりつつある．このような状況下で汎用コンピュータの機能を一切デフォルメすることなく，データベース技術，データ通信技術，システム開発ツールをそのまま適用可能なプラットフォームの出現は我々に大きな期待を抱かせる．

参考文献　[1] Summa Four, Inc., SDS-1000 System Description (1991)
[2] Summa Four, Inc., SDS-1000 System Software Overview (1991)

執筆者紹介　三　輪　次　郎 (Jiro Miwa)

昭和49年鈴鹿工業高等専門学校 金属工学科卒業．同年日本ユニシス(株)入社．主に客先ネットワークシステムのSEサービスに従事．現在，社会公共システム第2本部情報サービスシステム部に所属．

汎用的な知識ベースを持つ 変電所停電操作支援システム：QUALTES

QUALTES: An Expert System with a Generic Knowledge Base for Substation Stoppage Sequence Operations

鈴木常彦, 寺野隆雄, 工藤隆司
上林俊之, 伊神克典, 飯田 賢

要 約 今日，多くのAI応用システムはそれぞれが一からの手作りのシステムとなっている．その背景として，「知識獲得のボトルネック」が実用的なシステムの開発の問題点として浮かんできた．すなわち問題解決のための知識を将来にわたって利用していけるような形に抽出できるかどうかという問題点である．

このような問題を解決するには，目的に特化した開発ツールを開発するのが一つの方法である．ツールの開発にはタスク指向型ツールと領域特化型ツールの二つのアプローチが考えられる．

筆者らの開発したエキスパートシステム「QUALTES」では領域特化型のアプローチにより，適度な一般化レベルを見極めた上で，汎化知識ベース(GKB：Generic Knowledge Base)を構築し，対象領域における可搬性を高めた．

QUALTESの主たる目的は，変電所の停電操作においてあらかじめ記述される指令操作票の作成と検証を支援することにある．エンドユーザは，あらかじめ操作の対象となる変電所の系統を，簡単なグラフィカルインタフェースを利用して定義しておくことだけで，利用されるほとんどの指令操作票の作成や検証が容易にできる．結果として，実際の設備に対する保守性と操作の信頼性が一層向上した．

Abstract Today, most AI applications are developed from very initial stages. As a result, so-called "Knowledge acquisition bottleneck" make it difficult to develop practical systems. The knowledge for problem solving should be formulated to be used for future applications.

One way to solve this problem is to develop special purpose development tools. There are two approaches to develop such tool: the task specific approach and the domain specific approach.

The latter is effective when the characteristics of target domain tasks are well specified and when there are serious needs to develop similar applications. Therefore, we have adopted the domain specific one to develop QUALTES.

The main purposes of QUALTES are to support human operators to make and verify "Command Table Sheets", which define the proper operation sequences while executing maintenance tasks at a substation. Using the Graphical Interface of QUALTES, end users can easily define the configuration of an arbitorary substation. Command Table Sheets are automatically generated from the definition, and they can be easily verified by the simulator.

1. はじめに

今日，多くのAI応用システムはそれぞれが一からの手作りのシステムとなってい

る．電気・電力の分野においても各種の業務に対して，それぞれ別個の似通ったシステムが繰り返し開発されてきている[12][13]．その理由は，現状の知識ベース構築ツールでは，知識表現やユーザインタフェースの構築環境が提供されているのみで，知識の再利用といった枠組みがそれほど実現されていないという現状があり，「知識獲得のボトルネック」が実用的なシステムの開発の問題点として浮かんできた．すなわち問題解決のための知識を，将来にわたって利用していけるような形にしなければならない．

このような問題を解決するには，目的に特化した開発ツールを開発するのが一つの方法である．ツールの開発にはタスク指向型ツールと領域特化型ツールの二つのアプローチが考えられる．前者は診断型や計画型の市販ツールにいくつか例がある．また，後者は対象とする分野の業務の性質が明確になっていて類似のシステムをいくつか必要としている時には有効な方法である．ただし，知識の再利用においては，目的に特化する深さでその使い勝手が決定される．過度な一般化では応用範囲は広がるがカスタマイズしなければならない量が多くなり，過度な特殊化では非常に狭い範囲の応用しかできなくなり，適度な一般化のレベルの決定自体がむずかしい．

著者らの開発したエキスパートシステム「QUALTES」* では領域特化型のアプローチにより，適度な一般化レベルを見極めた上で，汎化知識ベース（GKB：Generic Knowledge Base）を構築し，対象領域における可搬性を高めた．こうした試みは他にも見られ，なかでも米国の EPRI (Electric Power Research Institute) の Plexsys プロジェクト等は注目に値する．Plexsys は原子力発電所での応用に特化したシステムであり，プラント機器に関する領域知識を持っている．この例でも開発ツールとしては KEE** が使用されており，QUALTES の汎化知識ベースの試みは，Plexsys に類似している．

Plexsys も QUALTES も基本的なプラントモデルと，特定のプラントモデルを構築するためのグラフィカルなエディタ，モデルベース推論，プラントシミュレーションといった機能を備えている．しかしながら QUALTES は Plexsys と比較すると，以下のような違いがある．

1) 対象が指令操作票の作成と検証業務という狭い領域に絞られている．このため GKB には極めて具体的なプラントモデルを持たせることができた．一方 Plexsys では，多くのアプリケーションで使用できるようにプラントの部品がかなり一般化されている．
2) 生成された個別のエキスパートシステムはスタンドアロンである．データベースやオンラインシステム，あるいは制御システム等の複雑な他システムとつなぐ必要がない．
3) QUALTES は指令操作票を検証するためにプラントのシミュレータを持っている．変電所というプラントモデルでは，シミュレータはひたすら多種多様な構成機器間の制約条件をチェックするのみである．こうした機能は ATMS を利用することにより容易に実現できた．

---

* Substation Stoppage Sequences operation Support Expert System より四つの S，または四つの subsystem の意より転じて「QUALTES」と命名した．

** 米国 IntelliCorp 社の登録商標である．

2. QUALTESの開発

変電所において点検等の作業が行われる時には，安全のため適切な停電区域を設定する必要がある．その設定のためには，多くの機器の複雑な操作が要求され，手順を間違うと大きな事故につながりかねない．このため変電所の保守を担当する部署（電力センター）では，作業に先立って機器の操作手順を記した「指令操作票」を作成する．担当者が作成した指令操作票は，何人かの熟練専門家の承認を受ける．この作成と検証には多くの知識と時間が費やされるが，それでも間違いは生じうる．指令操作票の作成から検証までの作業を省力化し，信頼度向上に寄与するシステムへの期待は大きいものがあるが，70余りある対象変電所の異なった設備，運用の形態に合わせられるシステムを構築するのは困難であった．

筆者らはこうした背景のもと，指令操作票の作成と検証を支援するエキスパートシステム「QUALTES」を開発した．QUALTESは最初LISPワークステーションKS-303を用いて研究された中部電力管内の岡崎変電所の専用システム[1][3]~[5]であった．後にこのシステムは，前述の領域特化型ツールとして汎用化しUNIX*ワークステーションへ移植された[2][6]~[9]．ツールにはKEE3.1を用いている．QUALTESはGKBを持ったツールとしてのカーネルシステムと，それを使って変電所ごとに生成される個別システムに分けられる．

3. 個別システムの基本機能

変電所ごとに生成された個別システムは次の三つの機能を持つ．

3.1 模擬操作盤サブシステム

本サブシステムは，ディスプレイ上で変電所の機器をシミュレートする模擬操作盤である．ユーザは模擬操作盤上の機器の操作をマウスによるアイコンのクリックによって行ない，装置の入切状態や電圧のかかっている範囲（加圧範囲）等をビジュアルに確認することができる．システムには，しゃ断器，断路器，現場アース，機械ロック，DCロック等の操作対象となる機器のほぼすべてが組み込まれている．

3.2 指令操作票エディタ

ユーザはこの知的エディタを使って指令操作票を作成できる．本サブシステムは二つのフェーズからなる．第一は指令操作票を記述する前に，その記述フォーマットをユーザが定義する機能である．いくつかの変電所を管理する事業所ごとにそれぞれ異なった運用がなされており，同様の作業に対して多種多様な記述形態が求められる．筆者らはこれらのひな形をGKBにクラスライブラリとして定義した．これによりエンドユーザは，単に各種のひな形から最も適したフォーマットを選択すれば準備が整えられる．

第二が指令操作票のエディタである．事前に定義された個別変電所の知識ベースをもとにポップアップメニューが生成され，ユーザはマウスオペレーションだけでほとんどの指令操作票を記述することができる．

本機能で作成された指令操作票は次に述べるサブシステムと連携して自動的にシミ

---

* UNIXオペレーティング・システムは，UNIX System Laboratories, Inc.が開発し，ライセンスしている．

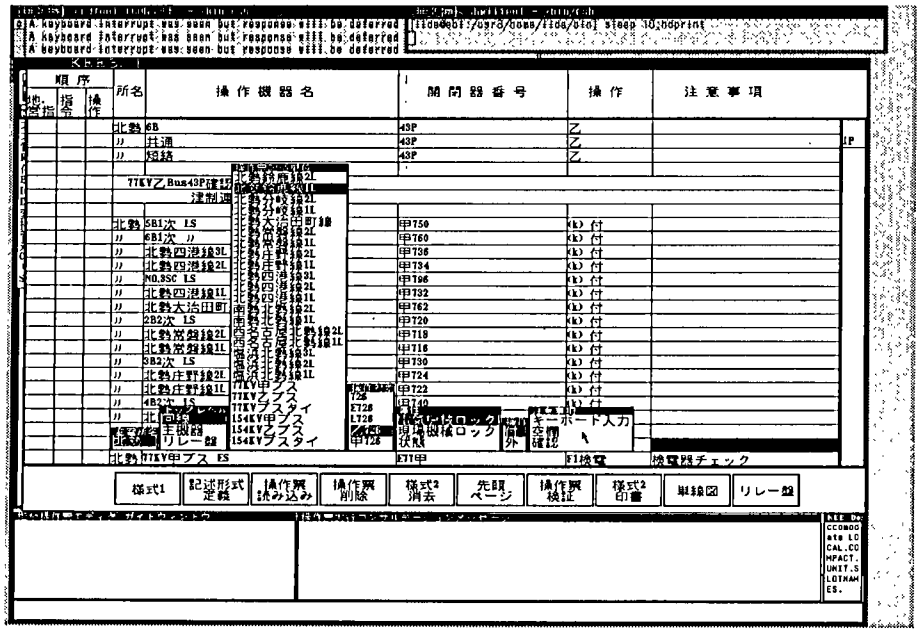


図 1 指令操作票エディタ
Fig. 1 Command table editor

ュレーションされ，手順等に間違いがあればメッセージが出されるため，ユーザは容易に間違いを発見し修正することができる．

図 1 に指令操作票エディタの使用画面を示す．

3.3 シミュレーション/コンサルテーション機能

エンドユーザはマウスで模擬操作盤を直接操作することや，指令操作票に書かれた手順を自動実行することができる．このとき，手順に誤りがあれば何らかのコンサルテーションが呈示される．これらのコンサルテーションは単に電気的・物理的な制約事項だけではなく，運用ルールに基づいても行われる．これらの機能により，ユーザは操作手順の問題点を発見し，実際の操作時のトラブルを事前に回避できる．図 2 にシミュレーションの様子を示す．

4. QUALTES における汎化知識ベース

QUALTES の汎化知識ベースは 2 層から構成される．第 1 層は変電所システムの核となる「変電所 Kernel KB」と，一般化された操作手順を定義した「変電所グローバルルール」から構成されており，以下のような構造を持っている．第 2 層は，ユーザの生成する個別の変電所である．

変電所 Kernel KB
- 機器ライブラリ
- 保護方式ライブラリ
- 図面ライブラリ

変電所グローバルルール
- 主機器操作手順ライブラリ
- リレー操作手順ライブラリ

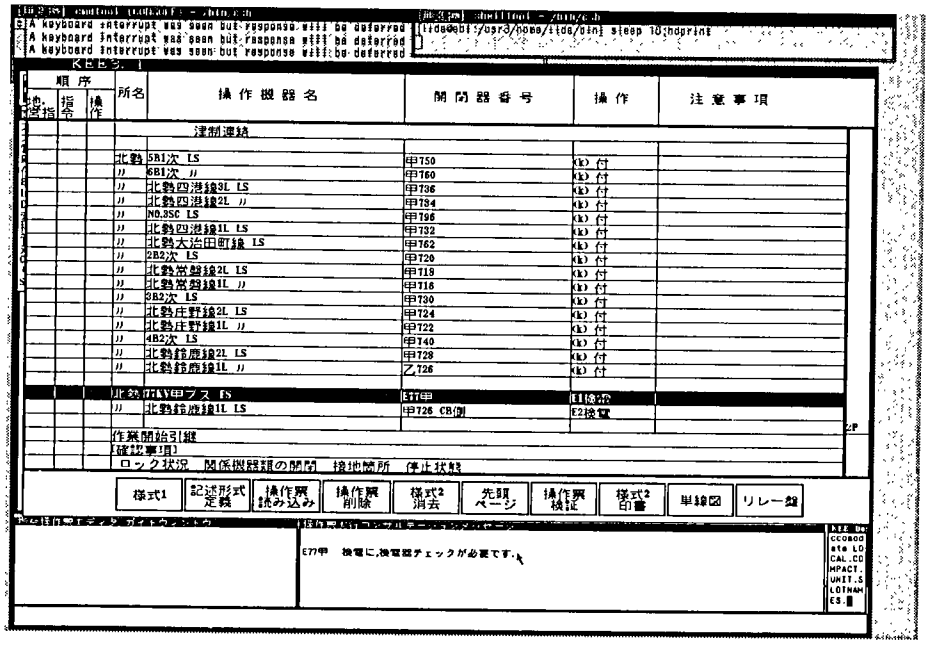


図 2　指令操作票のシミュレーション

Fig. 2　Simulation on command table sheet

・操作票表記ライブラリ

「変電所 Kernel KB」は「機器ライブラリ」,「保護方式ライブラリ」,「図面ライブラリ」から構成される.「機器ライブラリ」には,変電所の一般化された主機器のクラスが定義されており,「保護方式ライブラリ」には,中部電力管内で利用されている保護方式が定義されており,「図面ライブラリ」には,変電所の系統を表現する単線結線図用の各種機器の絵から,機器を一まとめにしたブロック単位の絵までが定義されている.

「変電所グローバルルール」は,「主機器操作手順ライブラリ」,「リレー操作手順ライブラリ」,「操作票表記ライブラリ」から構成される.「主機器操作手順ライブラリ」には,主機器の操作に関する手順の規則が定義されている.手順は,物理的な操作の制約の他に,運用時に安全を確保するために行う運用上の操作手順を含む.断路器の開閉操作には,隣接するしゃ断器の「閉」状態をインターロック条件としており,これは前者の物理的な制約にあたる.変電所内にある停電区間を確保する際には,断路器や,しゃ断器等の開閉器を操作して行うが,運用上の規定から開閉器の回路切断は1段では安全が確保できないので,2段で「切り」操作を行ったり,アースの設定を行ったりして,多重の安全確保を行っている.これは後者の運用上の操作手順に関する制約である.「リレー操作手順ライブラリ」には,リレー操作に関する操作手順が定義されている.「操作票表記ライブラリ」は,操作票を記述するための表記法を定めている.

これらの第1階層を用いることで,第2階層に属する各個別の変電所を容易に定義できる.

5. 個別の変電所知識ベースの生成

個別変電所を定義する際には，エンドユーザはインテリジェントなグラフィカルインタフェースを利用して個別の変電所を定義できる．汎化知識ベースは変電所の機器をオブジェクト指向の枠組みを用いたり，変電所の系統状態を宣言的知識表現を用いているため，ポータブルに実現できている．

5.1 電圧とアースの表現

系統の状態表現である電圧は，ATMS を用いることで効果的に表現できる．効果的と言う意味は，個々の隣接する機器間の関係だけに着目して宣言的に記述すればいいということである．宣言的に記述しておけば，系統が変更されても宣言を変更するだけで状態表現できてしまう．アルゴリズムを用いて同様の表現を行った場合には，系統の変更に伴ってアルゴリズムの変更が必要となる．

たとえば，L562 という断路器の加圧状態は，ATMS 宣言を用いて次のように宣言される．

（宣言（L562　電圧　あり）
：－（L562　状態　in）　（北勢分岐 1LA　電圧　あり））

この宣言は，「もし，L562 の状態が in で，かつ北勢分岐 1LA に電圧があるならば，L562 に電圧がある」ことを宣言的に表現している．この宣言を ATMS では「正当化」といい，宣言の中の「(L562　電圧　あり)」等のように括孤で囲まれた三組の記述を「命題」という．

ここで北勢分岐 1LA に電圧「あり」を仮説として与えると，L562 の「in」の状態の時には，L562 に電圧が伝播する．もし L562 の状態が「in」でなくなるか，北勢分岐 1LA の電圧が「あり」の仮説がなくなると，ただちに L562 の電圧「あり」が成立しなくなる．一般のルールベースシステムでは，このようにデータの依存関係に基づいて推論をダイナミックに行う機構は備っていない．このように，宣言は隣り合った機器間で宣言的に定義しておけば，系統全体の表現ができる．なお，この宣言は系統の機器構成から自動生成されるので，変電所の構成が変更されても自動的に反映される．

また，変電所の設備構成は信頼性向上のために，2 重の設備を構成するのが一般的となっている．2 重の設備にすることで，「廻り込み」の回路構成ができてしまう．図 3

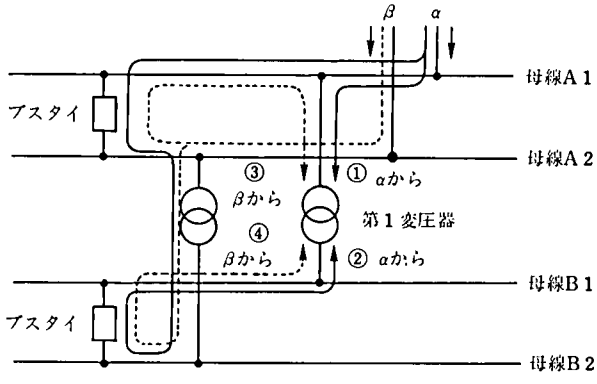


図 3　第 1 変圧器への電圧の伝播
Fig. 3　Voltage propagation to Bank # 1.

がその簡単な例である．電力の引き込み線，母線，トランスがそれぞれ2重に構成され，母線間はしゃ断器でつながれる．このような構成で，電圧の供給を α, β で行うと，第1変圧器には四つの経路から電圧の伝播がある．これは「第1変圧器の電圧あり」という命題が，四つのコンテクスト（文脈）により成立していることを意味する．四つのコンテクストは，それぞれ電圧伝播の四つの経路そのものである．図3で第1変圧器に伝播してきている四つの経路のうち，①と②は α からの導出であり，③と④は β からの導出であるが，導出の源が同じ（α や β）でも，コンテクストによって区別されるので，「廻り込み」も表現できる．

このシステムでは，アースの表現にもATMSを用いている．電圧がある点にアースが設定されたら，地落あるいは短絡事故が発生する．模擬操作盤上の系統のすべての箇所に設定されているデーモン（検知器）は，電圧「あり」とアース「付」が同時に発生するのを監視しており，発生した時点で事故の発生をシミュレーションする．また，シミュレータは，このとき発生した事故の原因を，宣言を用いてコンサルテーションする．

5.2 模擬操作盤作成サブシステム：MDB (Mimic Diagram Board)

QUALTESは第1階層のKernel KBと，第2階層の個別変電所からなることは述べた．Kernel KBの各クラスや個別変電所の各機器はオブジェクトで表現されており，個別変電所の各オブジェクトは，上位クラスのKernel KBから基本的な性質を継承している．

MDBは，しゃ断器や断路器，変圧器といった各機器の絵や機器群の絵をメニューとして保持しており，ユーザは各機器をマウスで選択して，MDB上のキャンバスに絵を描くことができる．各機器は内部的にオブジェクト表現されており，絵を描いた時点でオブジェクトが生成され，機器クラスに応じた分類がされた後，Kernel KBとのリンク関係が付けられる．

断路器のインターロック条件等の各種の制約条件も自動的に定義される．次に示すのは，「主機器操作手順ライブラリ」に定義されている母線間の断路器の「論理的インターロック条件」であり，機器のオブジェクトを生成しクラス分類されると自動的に継承定義される．

（ペアのLS　off）and（しゃ断器　off）
or（ペアのLS　in）　and（ブスタイ開閉器　in）

次に示すのが，機器の構成や各機器の名称が決定された後に定義される物理的インターロック条件の例であり，断路器「甲562」のものである．

（乙562　off）and（562　off）
or（乙562　in）　and（500　in）and（甲500　in）and（乙500 in）

断路器のインターロック条件は，設置される箇所により条件が異なるが，GKBを用いることで，機器の設置と同時に自動定義できる．ATMSを用いた系統状態の表現も，機器の構成と接続関係を用いて自動的に生成される．図4は個別変電所を構築している概念図である．図5はMDBの実際の画面である．

また，MDBにはリレー盤構築のためのサブシステムがある．エンドユーザは，MDBを用いることで複雑なリレー盤の構築が容易にできる．MDBでリレー盤定義の結果，

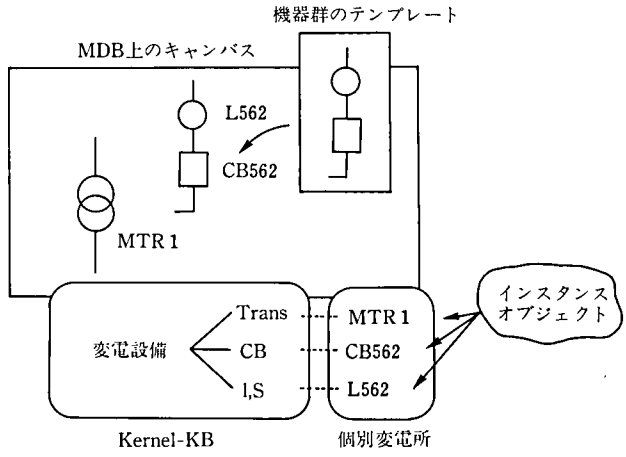


図 4　個別変電所の構築

Fig. 4　Construction of arbitrary substation

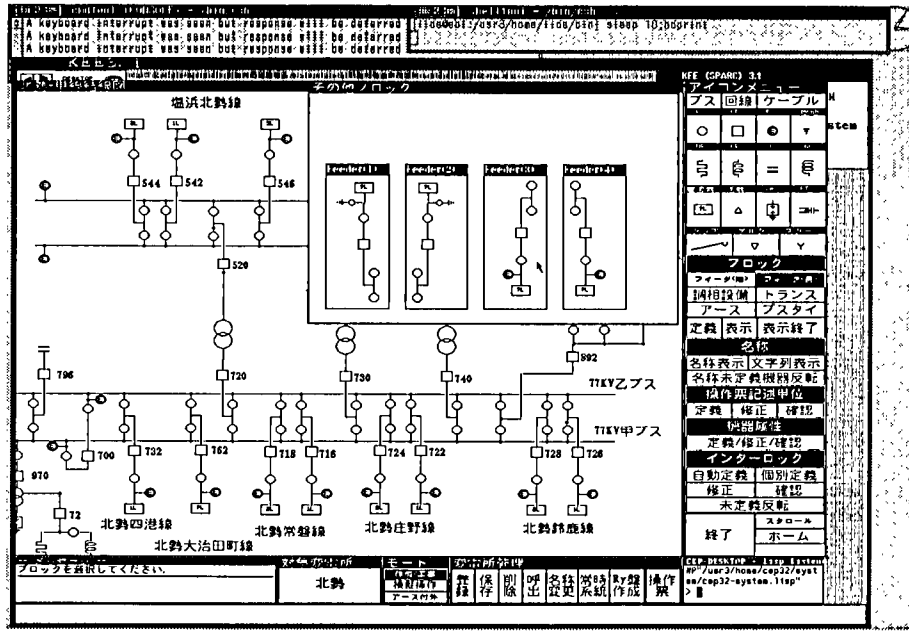


図 5　MDB の画面

Fig. 5　Mimic diagram board

操作に対する各種のコンサルテーションを出すことができる．岡崎変電所のシステムでは，各種の保護リレー回路をアルゴリズムで忠実に実現したリレー盤を用いていた．しかしこの方式では，複雑な保護回路をエンドユーザが定義しなくてはならない．また，近年アナログ方式の保護回路が，ディジタル方式に置き替りつつあり，保護回路をそのまま表現したのでは，機器の変更に耐えられず汎用性を欠くこととなる．

そこでわれわれは，複雑な保護回路を直接表現することなく，各種のコンサルテーションを得る枠組みとして，「リレースイッチ依存グラフ」を実現した[2]．リレースイッチ依存グラフでは，各変電所に設置されているリレー盤と，リレー盤の各種のスイ

ッチ類を定義し，各保護回路を構成しているリレー盤のグループを定義することで，すべてのコンサルテーションを得ることが可能である．

6. QUALTESの評価

QUALTESの開発にあたっては，通常の知識ベースシステムの場合と同様に，プロトタイピングの手法を用いた．そして，二つのバージョンを開発した．すなわち，開発の第1段階においては，岡崎変電所をモデルとして個別システムを開発し[1]，第2段階において，一般の変電所を対象としたシステムを開発した．

最初のバージョンは，GKBもMDB生成のための機構も備えていなかった．このシステムを利用して，基本的な機能やアルゴリズムを実証し，またプラントの各コンポーネントも検証された．

QUALTESの目的は，ドメインエキスパートである変電所操作員を支援することであり，彼等の経験的な知識を知識ベースに組込んでいるからである．次の各項目がエキスパートシステムの評価にとって重要である[10]．

① 知識表現
② 推論方法とその機構
③ 正解の比率
④ 正解の評価方法
⑤ 開発方法
⑥ 開発ツール/言語

結果は次のとおり満足すべきものである．

① オブジェクト指向によるプラントの表現は効果的である．
② メッセージ送信とATMSは効果的である．
③ ほとんどの操作票が扱えた．
④ 実際の変電所のテスト機能のためのシーケンステストは，このシミュレータに適用可能である．
⑤ GKBアプローチは有効である．
⑥ KEE/LISPは有効である．

KEEのようなハイブリッドツール（複数の知識表現を扱えるツール）は，優れたプロトタイプを作成するにはとても有効である．しかし，作られたシステムがその機能の良さにもかかわらずプロトタイプのままであることも多い．その一つの理由は，ハイブリッドツールの高機能性が，エンドユーザにとって複雑すぎるのと，知識ベースも修正するにはむずかしすぎることである．この問題の一つの解は，GKBのような領域指向のツールを構築することである．このような結果を踏えて，第2段階ではQUALTESにGKBを組込んだ．

このバージョンを利用して，想定利用者によるシステム評価を実施した．変電所システムに関して45名の専門家に対し，その機能を説明し，デモンストレーションも行った．専門家の意見は，「エキスパートシステム評価ガイドライン」[14]の「エンドユーザによる評価項目」に従って評価された．

ここで対象となる重要な評価項目は以下のとおりである．

① 推論時間
② 知識ベースのわかりやすさ
③ ユーザインタフェースの評価
④ ユーザはどこに期待しているか
⑤ 現場採用の要求

エンドユーザの評価は，質問票の形式で集計された．結果は以下のとおりである．

① 有効だが初期設定時間が長い
② 簡単（30%） 普通（70%） むずかしい（0%）
③ 簡単（12%） 普通（85%） むずかしい（3%）
④ 操作票の生成（62%） 操作票の検証（46%） 教育ツール（35%）
⑤ すぐにでも使いたい（19%） 一部手直しした後使いたい（65%）
システム化する必要はない（15%）

現在は，評価を継続している段階である．QUALTESにより生成された個別変電所は，現在2か所の変電所においてフィールドテストを実施している．評価フェーズを完了した後，QUALTESを全社的に広げる計画である．

7. おわりに

本稿では，QUALTESの基本的な機能を紹介すると共に，評価の中間報告を行った．これから次のことが結論づけられる．

1) ATMSの手法を用いることにより，変電所の複雑な機器の相互関係を，宣言的なフォーマットで容易に記述することができる．またその記述に基づき，ATMSは操作によって変化する変電所の状態を自動的に管理してくれる．結果として知識ベースの信頼性と保守性は格段に優れたものが得られることとなった．
2) GKBにより生成されたユーザインタフェースは，極めてよい操作性を持っている．これにより，個別変電所の生成という複雑な操作が容易になり，各変電所におけるエンドユーザによる知識ベースの構築が可能となった．

今後の課題としては，エンドユーザの要望の中でとくに多かった配電変電所まで範囲を広げることと，システムの動作をより軽快にすることがあげられる．

本システムの開発に惜しみない協力をしていただいた，中部電力(株)岡崎電力所ならびに四日市電力センターの方々に感謝の意を表する．

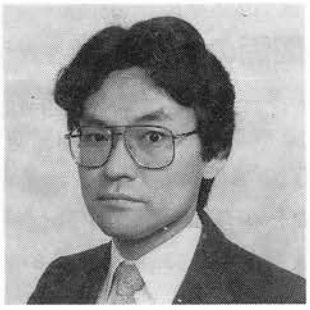
参考文献 [1] T. Suzuki, M. Shizawa, R. Kudo, "An Expert System for Verification of Switching Sequences at Electric Power Substation.-Simulation of Electric Power System by OOP & ATMS-",Proceedings of the Pacific Rim International Conference on Artificial Intelligence'90 (PRICA'90) , 1990, pp. 128～133.
[2] T. Suzuki, R. Kudo,"Approach to the maintenance of Knowledge Base by End-User-Substation Switching Sequence Support Expert System (QUALTES) -", Proceedings of the lEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 1991, pp. 557～580.
[3] 鈴木常彦，工藤隆司，"中部電力における変電所停電操作支援エキスパートシステム"，ユニシス技報，日本ユニシス(株)，Vol. 10. No. 3, 1990. 11, pp. 51～62.
[4] 鈴木常彦，矢田四郎，田中庸平，志澤通正，工藤隆司，三宅孝子，「変電所停電操作

支援エキスパートシステム-その(1)-」, 電気学会, 平成元年度全国大会論文集, 1989,pp. 8-115～18-116.
[5] 鈴木常彦, 田中庸平, 矢田四郎, 志澤通正, 工藤隆司, 塩崎慎治, 伊神克典, 「変電所停電操作支援エキスパート・システム-ATMS による系統の状態表現-」, 情報処理学会, 第 39 回全国大会論文集, 1989,pp. 223～224.
[6] 鈴木常彦, 伊神克典, 飯田賢, 工藤隆司, 「変電所停電操作支援エキスパート・システム-エンド・ユーザによる指令操作票の記述形式定義・作成・検証-」, 情報処理学会, 第 43 回全国大会論文集, 1991,pp.2-187～2-188.
[7] 鈴木常彦, 飯田賢, 伊神克典, 工藤隆司, 「変電所停電操作支援エキスパート・システム-エンドユーザによる知識ベースの構築-」, 情報処理学会, 第 43 回全国大会論文集, 1991,pp.2-289～2-190.
[8] 鈴木常彦, 工藤隆司, 伊神克典, 飯田賢, 「汎用的な知識ベースを持つ変電所操作支援システムの実現」, 人工知能学会, 全国大会論文誌, 1992, pp. 681～684.
[9] T. Suzuki, R. Kudo, K. Ikami, K. Iida T. Terano, "QUALTES : An Expert System with a Generic Knowledge Base for Substation Stoppage Sequence Operations", Proc. ESAP' 93 (4th Int. Symp. on Expert Systems Applics to Power Systems) , 1993, pp. 642～646 a.
[10] 寺野隆雄 (編著) : エキスパートシステム評価マニュアル. オーム社, 1992.
[11] 石塚満, 小林重信(編), "エキスパートシステム", 丸善, 1991,
[12] An International Survey of the Present Status and the Perspective of Expert Systems on Power System Analysis and Techniques. CIGRE SC38, WG 38.02 TF07, Expert System for Power Analysis and Techniques Final Report, 1988.
[13] Zhang, Z. Z., Hope, G. S., Malik, O. P., "Expert Systems in Electric Power Systems-A Bibliographical Survey.", Proc. IEEE Power Engineering Society 1989 Winter Meeting, WM 212-2 PWRS, 1989.
[14] T. Terano, et al., "Developing a Guideline for Expert System Evaluation-A Midterm Report.", Proc. PRICAI '90, 1990. (also available in Technical Report of Graduate School of Systems Management, The University of Tsukuba, Tokyo, No. 90—10,1990.)
[15] T. Terano, "Towards Domain-Specific Al Tools for Electric Power Applications", Proc. ESAP'91 (3rd Int. Symp. on Expert Systems Applics. on Power Systems), 1991, pp. 736～743.
[16] T.Terano, "A Checklist-Based Guideline for Expert System Evaluation.", Proc. ESAP'93 (4th Int. Conf. on Expert Systems Applics. on Power Systems), 1993, pp. 656～661.
[17] EPRI Report, "The Plant Expert System (PLEXSYS) Developmen Environments ; System Description and User's Manual", Version 2. EPRI NP-6410, 1989.
[18] J. de Kleer, "An Assumption-besed Truth Maintenance System", Artificial Intelligence., Vol. 28, No. 2, 1986, March, pp. 127～162.
[19] R. Fikes and T. P. Kehler, "The Role of Frame-Based Representation in Reasoning.", Comm. ACM, Vol. 28, No, 9, 1985, pp. 404～920.

執筆者紹介 鈴 木 常 彦 (Tsunehiko Suzuki)

昭和 37 年生. 60 年電気通信大学電気通信学部応用電子工学科卒業. 同年中部電力株式会社入社. 62 年 8 月技術開発本部電力技術研究所情報制御グループ配属. Al, CAD, ネットワーク等の研究に従事. 平成 4 年 8 月株式会社コンピュータ・テクノロジー・インテグレイタ出向. 現在, 技術計算事業本部応用技術第二部コンピュータグラフィックス応用グループにてマルチメディア関係のシステムの開発・研究に従事.

寺 野 隆 雄 (Takao Terano)



昭和27年生．53年東京大学情報工学科修士課程修了．53年から平成1年：電力中央研究所勤務，知識処理システム開発に従事．平成2年より筑波大学大学院経営システム科学専攻，現在，助教授．工学博士．知識工学，機械学習の研究に従事している．現在の興味は，知識システム開発方法論，事例ベース推論等．人工知能学会，計測自動制御学会，日本OR学会，情報処理学会，電気学会，経営情報学会，土木学会，IEEE，AAAIに所属．主要著書として「知識システムハンドブック」(1990)，オーム社(共編著)，「エキスパートシステム」(共著)，「エキスパートシステム評価マニュアル」(1992)，オーム社（共編著)，「知識システム開発方法論」(近刊）朝倉書店がある．

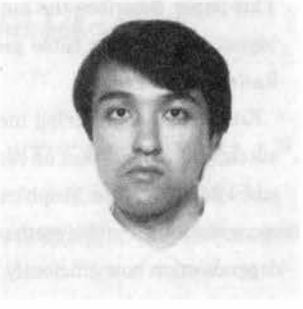
工 藤 隆 司 (Ryuji Kudo)

昭和29年生．55年中央大学経済学部卒業．同年日本ユニシス(株)入社．社内システム教育インストラクタを経た後，59年通産省工業技術院 電子技術総合研究所へ国内留学，61年米国Unisys社にて人工知能のトレーニングを受ける．現在，知識システム部に所属し，エキスパートシステムの研究開発業務に従事する傍ら，筑波大学大学院経営システム科学専攻に存学．人工知能学会，情報処理学会各会員．

上 林 俊 之 (Toshiyuki Uebayashi)

昭和37年生．62年大阪大学大学院理学研究科修士課程修了．平成元年日本ユニシス(株)入社．LISP言語の開発，エキスパートシステムの開発等を経て，現在，中部支社システム技術部に所属し，UNIX関連のシステム開発に従事．

伊 神 克 典 (Katsunori Ikami)



昭和36年生．59年名古屋大学理学部化学科卒業．63年中部ソフト・エンジニアリング(株)入社．現在システム4部に所属し，エキスパートシステムをはじめとするアプリケーション・ソフトウェアの開発に従事．情報処理学会会員．

飯 田 賢 (Ken Iida)

昭和40年生．平成2年東京大学教養学部基礎科学科第一卒業．同年中部ソフト・エンジニアリング(株)入社．LISP，KEEによるエキスパートシステム開発を行う．現在，システム4部に所属し，分散処理，データベース関連のシステム開発に従事．情報処理学会会員．

図面の自動認識
——東日本旅客鉄道(株)の信号連動図表作成システムにおける例

The Automatic Recognition of Handwritten Drawings
——The Signal-interlocking Table Generating System at East Japan Railway

蔵田幸一, 藤城雄二, 東野康臣

要　約　手書き図面の自動認識技術は,CADデータの初期入力負荷を軽減できることから注目されている.

本稿では,東日本旅客鉄道(株)東京電気工事事務所の「信号連動図表作成システム」(以下,本システム)の開発において採用した図面の自動認識の手法について述べる.

図面の自動認識はエキスパートシステムを代表とする知識工学の手法が有効と考えられている.プロトタイピング手法をとるエキスパートシステムは,知識の追加(認識図形の追加)が容易に行えることが重要である.また,図面の自動認識は処理結果の検証方法に課題があり,ユーザがいかに効率よく確認/修正できるかで,全体の処理時間が変わる.

このように,自動認識システムを実用化するためには,認識率とともに以下の点がポイントになる.

1) 認識図形の追加が容易であること
2) 認識エラーに対してユーザが容易に対応できること

これに対し本システムでは,シンボル辞書の活用,図形指示によるエラー内容確認等の手法を適用することで,実用化に成功している.

Abstract　The automatic recognition technology for handwritten drawings has been a focus of attention because of its advantage in cutting down on initial data input loads for computer aided design systems. This paper describes the automatic drawing recognition technique adopted for the development of the 'signal-interlocking table generating system' for the Tokyo Electric Construction Office of East Japan Railway Co., Ltd.

Knowledge engineering methodology including expert systems has been regarded as effective for the automatic recognition of drawings. It is important that prototyping-based expert systems be easy to add knowledge (or graphics to be recognized) onto. What is more, the point of automatic drawing recognition lies in the method by which to verify the output; that is, the whole transaction time greatly depends upon how efficiently users can verify and/or modify the output of their systems. In like manner, the following, besides recognition ratio, are also requisite for the implimentation of an automatic drawing recognition system:

1) Easiness of addition of graphics to be recognized
2) Easiness of user reaction to recognition errors

In contrast, the system developed for East JR has proved to be a successful one through the use of a symbol dictionary, the adoption of graphics-directed error reference, and so forth.

1. はじめに

図面の自動認識は，設計工程のシステム化を進める上で重要視されている．設計図面は，過去の類似図面を参考にして作成する，あるいは既存図面を修正してそのまま使用することが多く，設計図面の電子化（CAD 化）の有効性は広く認識されている．

一方，CAD システム導入にあたっては，既存する膨大な量の手書き図面をいかにして CAD データとしてシステム内に取り込むかが課題となる．手書き図面の自動認識は，とくに CAD データの初期入力負荷を軽減できることから注目され，昨今さまざまな試みがなされている．

本稿では，東日本旅客鉄道株式会社東京電気工事事務所の「信号連動図表作成システム」（以下，本システム）の開発において採用した図面の自動認識の手法について述べる．

本システムは単純な図面の清書システムではなく，図面から取り出した論理情報をもとに，別の設計図面を自動作成することを目的としている．このため，論理情報を取り出す処理（これをパターン理解という）が不可欠である．また，その認識結果の精度が自動作成する図面の質に影響を与えるので，システム全体の性能を左右する重要な処理になっている．

パターン理解はエキスパートシステムを代表とする知識工学の手法が有効と考えられている．プロトタイピング手法をとるエキスパートシステムは，知識の追加（認識図形の追加）が容易に行えることが重要である．また，図面の自動認識は処理結果の検証方法に課題があり，ユーザが効率よく確認/修正できるかどうかで，全体の処理時間が変わる．

このように，自動認識システムを実用化するためには，認識率とともに以下の点がポイントになる．

1)　認識図形の追加が容易であること

2)　認識エラーに対してユーザが容易に対応できること

本稿では，これらの点を中心に本システムで採用した手法を述べる．

2. システムの概要

本システムは，連動図（図 1）を入力し，連動表（図 2）を出力するシステムである[1]．

「連動図」は，駅の線路の配線や信号装置の配置を表した 2 次元の線画である．

「連動表」は，列車の進路設定を行うために連動図内の各種装置にほどこされる制

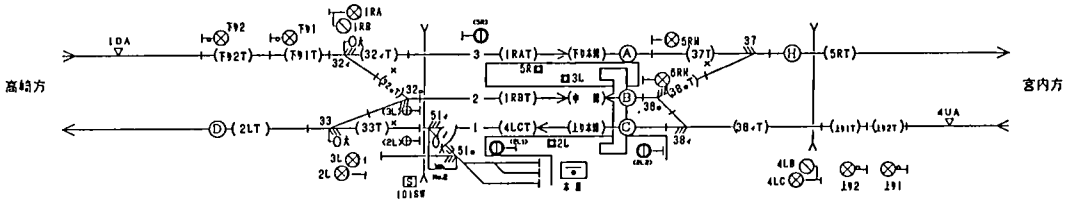


図 1　連動図の例

Fig. 1　Sample of interlocking drawing

名称	番号	鎖錠	信号制御又はてっ査鎖錠	進路鎖錠	接近鎖錠又は保留鎖錠	自動制御又は進路制御
場内信号機 場 時 方一下 り 本 線	1R	Ⓐ				
同 上 場 時 方一中 線		Ⓑ				
出発信号機 上 り 本 線一高 崎 方	2L	Ⓒ				
出発信号機 中 線一高 崎 方	3L	Ⓓ				
場内信号機 宮 内 方一上 り 本 線	4L	Ⓔ				
同 上 宮 内 方一中 線		Ⓕ				
出発信号機 下 り 本 線一宮 内 方	5R	Ⓖ				
出発信号機 中 線一宮 内 方	6R	Ⓗ				
鎖錠てこ 2 L T～中 線	101L					
鎖錠てこ 2 L T～ I						
電てつ器 (2 線)	82					
電てつ錠	93					
電てつ錠	97					
電てつ器 (2 線)	36					
電てつ錠 (2 線)	51					

図 2 連動表の例

Fig. 2 Sample of interlocking table

御の内容を表形式で表した図面である．

連動図と連動表を合わせて連動図表と呼ぶ．連動図表は，列車の安全運行には欠かせないものであり，信号設備設計の基本となるものである．

連動表作成処理の流れを図3に示す．なお「進路表」とは，列車の進路をリストアップしたものである．

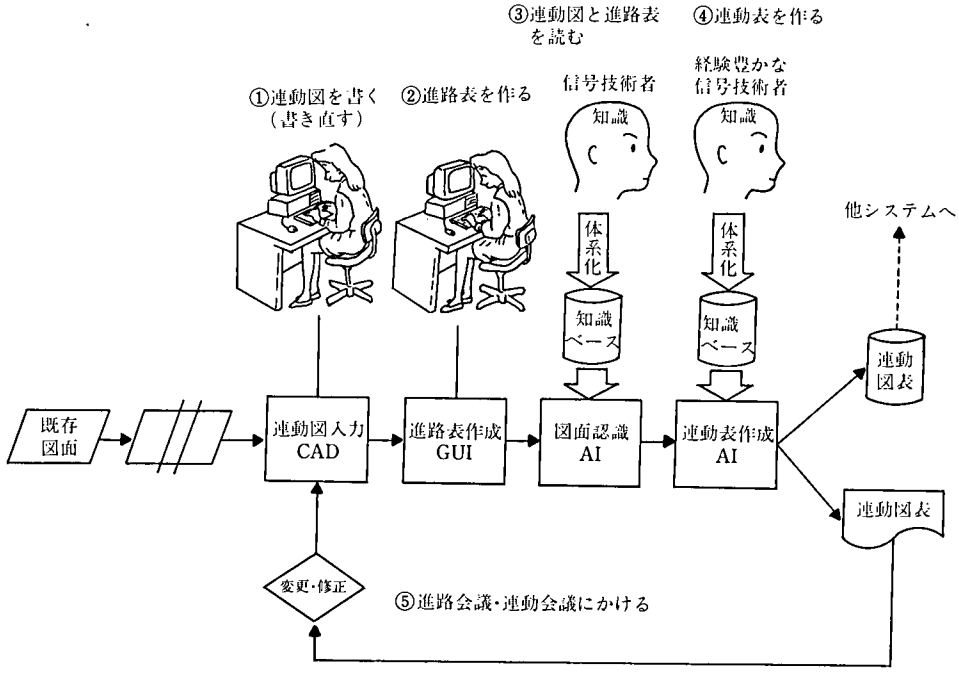


図 3 連動表作成処理の流れ

Fig. 3 Flow of making a interlocking table

駅の工事は，駅の新設よりも既存駅の改造の方が圧倒的に多い．このため，信号設計業務のシステム化の際には，既存の連動図の電子化が不可欠である．

電子化の手法として，原図をトレースする方式ではユーザの負荷が大きい．また連動図は論理図であり，線路の実測値を反映したものではないが，個々の装置の位置関係や配線形状は，現場に忠実に描かれている．CAD システム上で自由に作図する方式では原図を逸脱した図面になりやすく，官庁に認可されない危険がある．このような

理由により，本システムでは自動認識の手法を採用している．

3. 図面の自動認識

自動認識の一般的な処理手順は以下の通りである．

処理1　手書き図面をスキャナで入力する．図面はラスタデータ* として，システムに入力される．

処理2　ラスタデータをベクタデータ** に変換する(ベクタ変換)．線分や文字列への変換，シンボルへの変換もこの段階で行われる．

処理3　図面から論理情報を取り出す（パターン理解)．

処理3は処理2と平行して行われる場合もある．また処理3はCADシステムで知的処理を行う上で不可欠な処理である．

処理1，処理2については，研究が進められ，多くの成果が発表されている[3][4][5]．単純な図形についてはハードウェアの進歩等により，ほぼ実用レベルに達しているといえる．一方，処理3や処理2のうちのシンボルの認識等は，入力図面に関する知識を必要とすることから，知識工学の手法が注目されているが，技術的に確立されたとは言えず，個別に対応しているのが現状である．

本システムでは処理1，処理2をPC上のCADに適用し，処理3をEWS上で開発した．PC上で行うベクタ変換は線分，円弧等にとどめ，文字列，シンボルは手入力する（トレースする)．

以下，本稿ではEWS上で行っている処理3のパターン理解について述べる．

4. パターン理解

図面から論理情報を取り出す処理をパターン理解という．

コンピュータシステムは，与えられた図形情報と対象領域の知識モデルとのマッチングを行い，内部にインスタンスを生成することで，図面を「理解」する．

4.1 ボトム-アップ処理とトップ-ダウン処理

パターン理解の意味決定の過程で一般的にとられる手法として，データ（線画）を解析してモデル候補を仮定するボトム-アップ処理と，モデルに関する知識を用いてデータを調べ，そのモデルの正当性を検証するトップ-ダウン処理がある[2][6]．本システムでは，図面中にシンボルを見つけると，モデル（フレーム）を生成し（ボトム-アップ処理)，周辺の図形要素がそのモデルに関する知識に適合しているか調べる(トップ-ダウン処理)ことを繰返す．すなわち，シンボルを認識のきっかけとして使用している．

モデルはより低レベルのモデルの集合であることがあり，階層構造をとる（図4)．また，それぞれのモデルに関する知識も，モデルと図形との関係，上位モデルと下位モデルの関係に関するもの等がある．階層構造のそれぞれのレベルのモデルに対して，ボトム-アップ処理とトップ-ダウン処理を行い，より上位のモデルを構築することで認識は進む．

---

* 図面のイメージを0，1に2値化したデータ．

** 線分，円弧等通常のCADシステムが扱えるデータ．

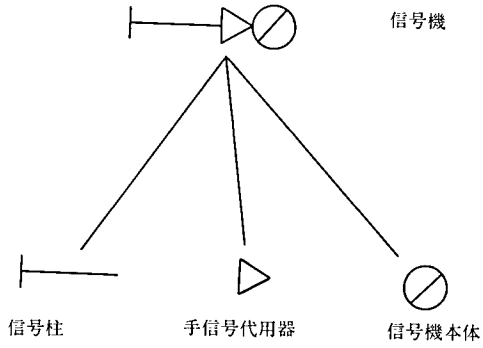


図 4 信号機の階層構造

Fig. 4 Hierarchical structure of a signal

パターン理解は対象領域に適した制御戦略を必要とする．これは，トップ-ダウン処理の際に，関係するモデルのボトム-アップ処理が終わっていない場合，マッチングが失敗する可能性があるためである．汎用的な手法の研究も進められているが，決め手となるものはない．

本システムでは，モデル間の依存関係を積極的に処理手順に取り込むことで対応している．具体的な処理手順は概略次のとおりである．

1) 不要な線分の整理，文字列の連結等の前処理を行う（ノイズの除去）．
2) 軌道回路* を認識する．
3) その他のモデルを認識する．

処理手順で軌道回路の認識を優先している理由は次のとおりである．

① すべての信号設備は，軌道回路上を走行する列車を制御するために設置される．このため，軌道回路と何らかの関係を持っている（軌道回路の認識が終わらないと軌道回路との対応付けができず，認識エラーになる）．一方，軌道回路はそれ単独で認識が可能である．

② 軌道回路は線分として構成されるが，線分はそれ単独では論理的な意味を決定できない．認識の初期段階で線分をノイズかそうでないか判定することにより，処理効率向上が望める．

4.2 離散的緩和法とプロダクションシステム

離散的緩和法[2] は，パターン理解の代表的手法の一つである．本システムでは，プロダクションシステムを緩和法の代替として使用している．

離散的緩和法は，局所的な特徴（モデル）の集合に対して整合的なラベル（属性値）付けを行うことを目的とするアルゴリズムである．

特徴の集合を X，要素 x_i に対する可能なラベルの集合を $L(x_i)$ とする．さらに関数 f と g を以下のように定義する．

$$f(x_i, \mu_j)=\begin{cases}1 & \text{特徴 } x_i \text{ にラベル } \mu_j \text{ を付けえる場合}\\ 0 & x_i \text{ に } \mu_j \text{ を付けえない場合}\end{cases}$$

* 線路を適当な位置で絶縁で区切り，電気回路を構成したもの．列車を電気的に検知する機能を持つ．

$$g(x_i, x_j, \mu_k, \mu_l)=\begin{cases}1 & x_i, x_j \text{ にそれぞれ } \mu_k, \mu_l \text{ を付けえる場合}\\ 0 & x_i, x_j \text{ にそれぞれ } \mu_k, \mu_l \text{ を付けえない場合}\end{cases}$$

このとき，隣接する特徴の組を取り出し，

$f(x_j, \mu_l) * g(x_i, x_j, \mu_k, \mu_l)$

の値を調べることにより，整合的なラベル付けを見出そうとするものであり，x_i にラベル μ_k を割り当てたとき，x_j に与えるラベルが存在しないならば，ラベル μ_k を x_i の候補からはずす処理を行う．

以上の処理は，より明示的にプロダクションルール* で表現できる．

```
if    ?xi∈X
  and  ?xj∈X
  and  隣接(?xi, ?xj)；隣接する X の要素 xi, xj を取り出し
  and  ?μk∈L(?xi)  ；xi にラベル μk を割り当てたとき
  and
    not(?μl∈L(?xj)and                    ；xj に与えるラベルが存在
      f(?xj, ?μl) * g(?xi, ?xj, ?μk, ?μl)=1)    ；しない
then
  remove  ?μk, L(?xi)；?μk を ?xi の候補からはずす．
```

本システムでの具体例を示す．

図5において信号機（円形の図形）に名称を対応付けることを考える．作図規則上，名称は信号機の前後左右どこに書いてもよい．

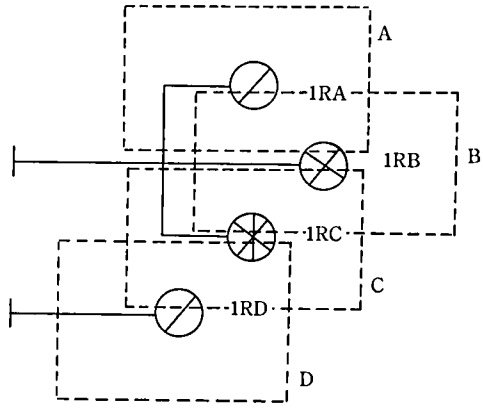


図 5　信号機と名称

Fig. 5　Signals and their names

初期状態でそれぞれの特徴（信号機）のラベル（名称）の候補は，

A：1RA

B：1RA，1RB，1RC

C：1RC，1RD

---

* 本システムでは日本ユニシスのエキスパートシステム構築支援ツール「GNOSIS-II」を使用している．なお便宜上，ルールの構文を一部簡略化して記載している．

D：1RD

である．対応付けをするために，次のルールを適用する．

```
if    信号機（?信号機 i)
  and  信号機（?信号機 j)
  and  ?信号機 i≠?信号機 j；信号機を二つ取り出し
  and  名称候補（?信号機 i，?名称 k)；?信号機 i に?名称 k を割当てたとき
  and not（名称候補（?信号機 j，?名称 l)；?信号機 j に割当てる
            and  ?名称 l≠?名称 k)      ；名称が存在しない．
then
  remove  名称候補（?名称 k，?信号機 i)；?名称 k を?信号機 i の候補からはずす．
```

この結果，正しい対応付けが得られる．

推論エンジンは，緩和法の繰り返しサイクルと同等の機能を持つ．また，明示的なルール表現で条件判定用の関数の代用ができるので有効である．

5. シンボル辞書

4.1節で述べたように，本システムではシンボルを認識の手掛りとしている．

連動図のシンボルは数百種類にのぼり，プロトタイピングの初期モデルですべてのシンボルを対象とすることは困難である．また信号装置の技術革新により，新たなシンボルが追加されることがある．

このため，本システムではシンボル辞書を設け，シンボルの追加に対応している．フレーム名等の静的な認識属性は辞書に登録するだけでよい．

一方，連動図のシンボルは静的な属性だけでなく，認識処理そのものの制御にかかわる性質を持ったものがある．

5.1 複数のフレームに対応するシンボル

図6のシンボル（三角形）は，同一の形状をしているが，それぞれ意味（対応するフレーム）がまったく異なる．

このようなシンボルに対しては，本システムでは以下の手法で対応している．

1) 1シンボルから複数のフレームを生成……各々のシンボルは認識モデルとしてのフレームに対応付けてある（シンボル辞書に登録してある）．シンボルをフレームに変換する際（シンボル辞書を調べたとき），類似形状のシンボルが他に存在する場合，考えられる候補のフレームをすべて生成する．

2) モデルが確定したら他の候補を削除……それぞれの候補は，モデルに矛盾が生じない限り，有効なデータとして処理される（矛盾が生じると抹消される）．最終的に一つのモデルが正しく認識されたとき，そのモデルが確定したものとみなし，他の候補を削除する．

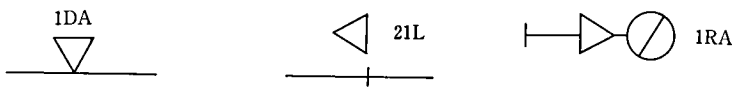


図 6 同一形状シンボルの例

Fig. 6 Sample of same figure symbols

本方式は，以下の点で有効である．

① 候補モデルすべての認識の経過を保存：候補の決定に失敗した時でも，候補の認識経過を保存しているので，その内容を見ることができる．場合によってはその中から，最も妥当な候補を認識結果として採用することができる．

② 保守が容易：新たなシンボルが追加され，モデルの形状に曖昧さが発生する場合でも，シンボル辞書に登録するだけですむ．

5.2 回転するとフレームの属性が変わるシンボル

信号機のシンボルは，その形状が現示（表示）する信号を表しており，連動表の論理に直結しているが，図7のように周囲の図形との相対的な配置が変わると（回転させると）意味が変わるものがある．シンボルとしてはそれぞれ個別のものとして登録してあるが，システムの作図規制としてしまうと操作性が低下する（タブレットでシンボルを探すよりも，画面近傍にあるシンボルをコピー/回転する方が楽なことがある）．

図 7　回転すると属性が変わるシンボル

Fig. 7　Symbols whose attributes are changed when rotated

本例における手法は以下の通りである．

1) シンボルに接続点と認識基準点を設ける……個々のシンボルは連動図の作図規則により，他の図形と接する点（位置）が決まっている．この［点］を接続点とする．さらに図形の基準点とは別に，シンボルがその本来の意味を表す点を接続点の中から選び認識基準点とする（180度回転すると元の形状に戻るシンボルは認識基準点が2個となる）．この認識基準点は，シンボルの性質により一意に定まる点とする．図8の信号機を例に説明する．

・信号機は特定方向に進行する列車に対応しており，その列車の進行方向に合わせて作図される．このため，信号機自身が方向(列車の進行方向と等しい)を持っている．このベクトルを信号機シンボルに重ねたとき，ベクトルの始点となる点を認識基準点とする．

信号機のように，特定のベクトルを持つシンボルはその始点を認識基準点とすればよい．逆にベクトルを持たないシンボルは作図角度が認識結果に無関係なので，認識基準点は不要である．

2) 認識基準点を決定し，シンボル辞書と照合する……まず，図形の配置とモデルに関する知識を用いてデータとしての認識基準点を決定する．図8の信号機のように辞書に認識基準点を複数持つシンボルは，単独で認識基準点を決定することはできない．一方，信号柱はその形状から認識基準点は1個しかない．このように認識基準点が不定のシンボルと，既定のシンボルが要素となり，図4に示すようにより上位の［信号機モデル］を構成する．

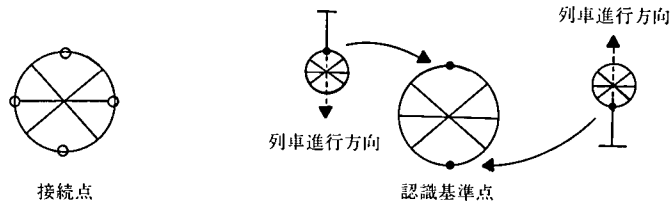


図 8　接続点と認識基準点

Fig. 8　Contact point and recognition base point

［信号機モデル］は単一のベクトルを属性として持つ．言い換えれば，構成要素であるそれぞれのシンボルのベクトルは共通になるはずである．この性質を利用し，シンボル間で属性を伝播させることにより，認識基準点を決定する(図 9)．

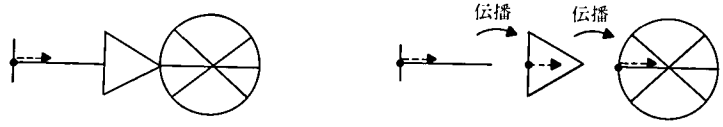


図 9　認識基準点の決定

Fig. 9　Determine the recognition base point

次に，データの認識基準点とシンボル辞書の認識基準点を比較する．データの認識基準点とシンボル辞書の認識基準点が一致すればそのままでよい．一致しない場合，対応するシンボルが辞書に登録してあるので，属性値をそれの内容に置き換える（図 10)．

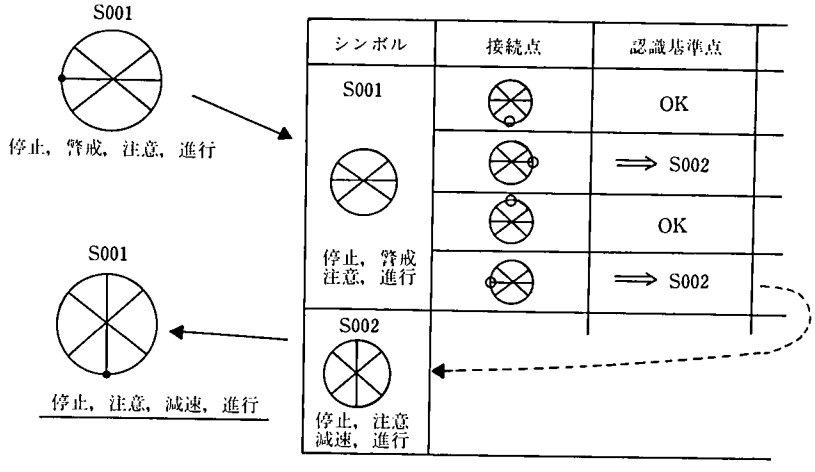


図 10　シンボル辞書との照合

Fig. 10　Comparing with the symbol dictionary

本方式の特徴は以下の通りである．

・接続点は作図規則そのものである．作図規則をプログラム内に埋め込むのでなく辞書として外付けにするので，わかりやすく，また保守も容易である．

・作図エラーを容易に判定できる（接続点以外で他の図形と接する場合は作図エラーである）.

・シンボルの追加に容易に対応できる.

このように，本システムではシンボル辞書を活用することにより，知識の追加/変更に備え，認識精度向上を計っている.

6. 認識エラーへの対応

手書き図面を100%完全に認識することは困難である.

自動認識の弱点として,「認識結果のチェックに時間がかかる」点が指摘されている.とくに，ベクタ変換の処理で言われていることであるが，誤認識された図形を検出する有効な方法がない場合，結果を原図と付き合わせてしらみつぶしにチェックすることになり，ユーザに大きな負荷がかかる.

これに対応するため，以下のような対策がとられている[3].

1)　原図と変換後の図面をペアで画面上に表示する.

2)　認識不能あるいは曖昧な図形に印を付ける.

ベクタ変換時に発生するエラーは，ユーザはエラーの場所さえわかれば，原図（手書き図面）と変換後の図面を見比べることにより修正できる．一方，パターン理解では，図面の見た目が正常でもエラーになることがある.

パターン理解で発生する認識エラーには，次のものがある.

・作図エラー

　　システムが前提としている作図規制を逸脱したもの.

・論理エラー

　　認識結果に論理的な矛盾があるもの.

・誤認識（システムで検知不能）

　　システムは正常終了する．最も見落しやすい.

ベクタ変換の出力がCAD図面であるのに対し，パターン理解の出力はシステムの内部に生成されたインスタンス，あるいはそれが加工された後に出力される論理情報である．このため，図面のように直接見比べてチェックするわけにいかず，システムの出力するメッセージが重要となる．システムがなぜ認識できないのか，その理由を正確にユーザに伝達できない場合，システムに対する不満/不信を招くことになる．逆に，適切なエラーメッセージを伝達するシステムの場合，ユーザが容易にシステムのくせを把握でき，修正が容易になる.

いずれにしても，認識エラーに対して，ユーザが効率よく対応できるかどうかが，自動認識システムの性能を左右するといえる.

本システムではパターン理解の認識エラーに対するユーザへの支援機能として，以下の機能を提供している．なお機能1は作図エラーおよび論理エラーへの，機能2〜機能4は論理エラーおよび誤認識への対策である.

機能1　認識エラー図形の強調表示と原因の説明

　　認識処理でエラーとなった図形は画面上で強調表示される．さらに，その図形をクリックすると，メッセージが表示され，原因を知ることができる（図

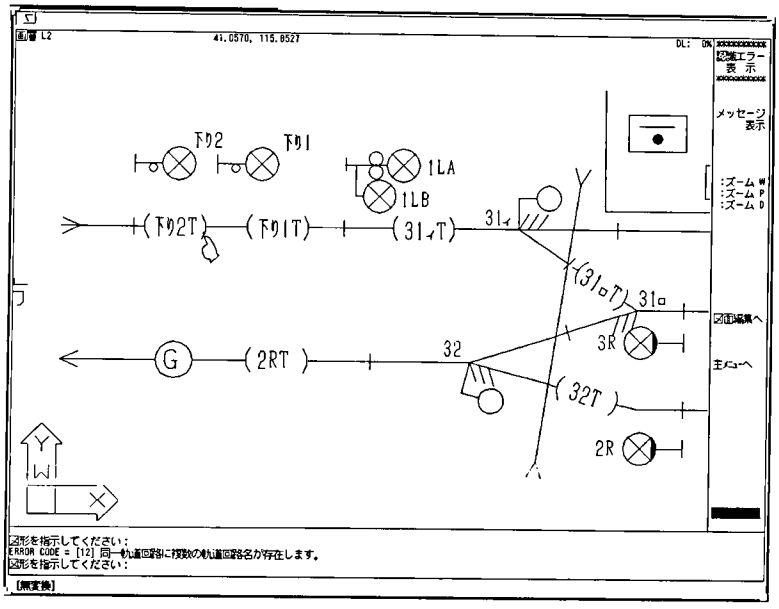


図 11　認識エラーの表示例

Fig. 11　Sample of recognition error message

11)．ユーザはそのメッセージに従って図面を修正すればよい．システムがエラー原因をユーザに伝えることにより，システムのくせを容易に把握することができ，ユーザの作図精度向上が期待できる．さらに，認識処理は連動図の作図規則に基づいて行うので，図面品質も向上する．

機能 2　シンボルの強調表示

本システムのパターン理解はシンボル化が 100%行われていることが前提である．シンボルを強調（着色）表示することで，シンボル化されていない図形を容易に見つけることができる．

機能 3　軌道回路線分の強調表示

軌道回路は連動図内の線分として最もデータ量が多く，かつ認識処理上，最も重要な図形である．処理実行上，論理的矛盾が発生する場合は，システムが自動的に指摘することが可能であるが，図形処理の許容誤差等の理由で認識からもれる図形についてはエラーの検知が困難である．このため，システムが軌道回路として認識した図形を強調表示し，ユーザがチェックできる機能を提供している．

機能 4　設計知識によるエラーの指摘

本システムは，連動図の認識が最終目的ではなく，認識結果ともう一つの入力データである進路表をもとに，設計知識を用いて連動表を作成することを目的としている．図面の認識が正常に終了しても，連動表の論理を生成する段階で矛盾が発生する場合がある．逆にいえば，認識結果を設計知識を用いて検証することができ，図面認識だけでは見のがす誤りを検知することができる．

認識エラーの場所と原因がわかると，ユーザはCADの機能を使用して図面を修正する．CADの標準機能なので操作性は良い．

なお，自動認識が困難な部分については，図面内に補助シンボルを書込むことで対応している．

7. 評　　価

本システムについての評価を以下に述べる．

1) 処理時間……本システムのパターン理解の実行時間はルート数が100程度の中規模駅で約30分である．図面の認識システムとしては，単純に実行時間だけを評価するわけにいかず，認識結果の質や，ユーザのエラー対応時間を含めた1枚の図面が認識できるまでのグロスの処理時間を評価すべきであろうが，本システムの場合，十分実用に耐えるとの評価を得ている．

課題は，図面修正後の再実行時にも認識を最初から行う点である．現在，再認識時に図面の修正内容を解析し，不要な処理をスキップするような機能を検討中である．

2) 操作性……認識エラーへの支援機能については意見の分かれるところであろう．知的会話処理により直接認識結果を補正する方法も考えられるが，本システムの方式に比べ，グロスの処理時間を短縮できるという長所がある反面，図面の内容（モデルの内容）を熟知したユーザにしか操作できないという欠点がある（本システムの場合，図形の配置に関するメッセージを出力するので専門家でなくても図面を修正できる）．また，モデルそれぞれに対して会話機能を用意しなければならないので，開発コストがかさみ保守性も悪い．

3) 保守性……シンボル辞書を活用することにより，良好な保守性を得ている．

8. お わ り に

図面の自動認識技術は確立されたとはいえず，処理対象図面に関する知識をどれだけシステムに移植するかが認識率を左右する．自動認識システムとして，知識の追加が容易な構造にすることが必要である．また，認識率と同程度あるいはそれ以上に認識エラーへの対応機能が実用化する上で重要である．

本システムでは，シンボルに接続点，認識基準点を導入してシンボル辞書を活用する手法，図形クリックによるメッセージ表示機能等を適用することで，実用に耐える自動認識システムとすることができた．

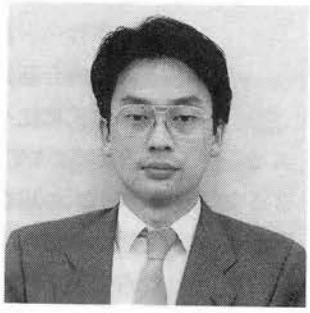
これらの手法は，連動図の自動認識システムを実用化する上で有効であるだけでなく，十分汎用性があると考えている．認識基準点を例にとると，本システムでは列車の進行方向のベクトルをそれの決定に使用したが，この手法は他の分野の図面にも適用できるであろう．たとえば，プラント装置系統図では，配管内の流体のベクトル，電気回路図では，電流のベクトル等がそれである．

最後に，本システムの開発に尽力された関係者各位に紙面を借りて謝意を表する．

参考文献 ［1］ 田島外幸 他，“信号連動図表作成システムの開発”，第28回「鉄道におけるサイバネティクス利用国内シンポジウム」論文集，日本鉄道技術協会，1991.
［2］ 安西祐一朗，認識と学習，岩波講座ソフトウエア科学，16，岩波書店，1989.
［3］ 加瀬信次 他，“プラント装置系統図における図面自動入力の活用”，PIXEL，No. 84，1989.
［4］ 恒川尚，“図面の自動読取りとその問題点”，PIXEL，No. 84，1989.
［5］ 坂内正夫，“これからの図形の自動読取り技術”，PIXEL，No. 84，1989.
［6］ 松原仁 他，“プロダクションシステムによる線画の解釈”，人工知能学会誌，Vol. 1，No. 1，1986.

執筆者紹介 蔵 田 幸 一 (Koichi Kurata)
昭和21年生．47年東京都立大学電気工学科卒業．40年日本国有鉄道入社．62年より東日本旅客鉄道(株)に所属し，主に電気関係工事，システム開発に従事．現在，東京電気工事事務所 技術企画室長．電気設備学会会員．

藤 城 雄 二 (Yuji Fujishiro)
昭和33年生．56年名古屋工業大学工業化学科卒業．61年日本ユニシス(株)入社．航空システムの開発に従事．現在，システム技術本部 知識システム部に所属．AI応用システムの開発を担当．



東 野 康 臣 (Koshin Higashino)
昭和24年生．46年芝浦工業大学工業経営学科卒業．同年日本ユニシス(株)入社．主にOR関連のシステム開発に従事．現在，システム技術本部 知識システム部 企画課課長．人工知能学会会員．

U6000H手順（JCA-H手順）対応ソフトウェア

1. 背　　景

JCA手順（J手順）が1980年7月に，日本チェーンストア協会により制定されてから10年以上が経過した．この間流通業界では受発注業務を中心に，主に公衆回線を使用したオンライン企業間データ交換システムが普及し，流通業務の合理化に大いに貢献してきた．

しかし，オンライン企業間データ交換の普及に伴って，取引先の増大，取引データ量の増加等，ネットワーク規模は拡大し，そこを流れる事務処理は多様化してきており，従来の流通業界標準手順であるJCA手順では，以下の理由から対応が困難なケースが顕著になってきた．

① 通信接続時間が長くなったことにより，業務処理の遅延や通信コストが増加してきたこと．

② 漢字データ伝送や，セキュリティの強化等新たな機能が求められてきたこと．

JCA手順で規定している公衆回線やDDX-Cでは，回線の高速化への対応には限界がある．また，異業種間でのオンライン取引も増加してきており，特定の業界標準であるJCA手順で業種間の合意形成を図ることは困難である．

こうしたことから，回線の高速化と業界横断的に利用できるプロトコルの確立を目的に，日本チェーンストア協会内のOSIプロジェクトで，ISDN回線を使用し，国際標準通信手順であるOSIに対応した，「新手順」の開発を行うこととなった．そして，平成3年4月に日本チェーンストア協会よりJCA-H手順が発表され，平成4年3月に通商産業省により，JCA-H手順を基本としたH手順が策定されるに至った．

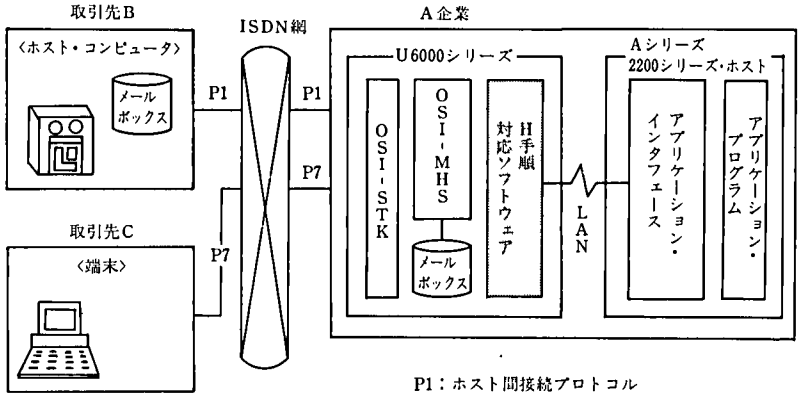
2. 概　　要

H手順は，通信回線にISDNを使用した新しい通信標準プロトコルである．また，データ表現形式と転送方式(OSI参照モデル：高位層)にはOSI-MHSを採用しており，メッセージ交換をベースにしたデータ交換機能を持つ．

H手順対応ソフトウェアは，このH手順のホスト側機能を支援するソフトウェアで，U6000シリーズ上に通信サーバとして位置付けられ，Aシリーズ/2200シリーズ等ホスト上の業務アプリケーション・システムに接続される（図1）．

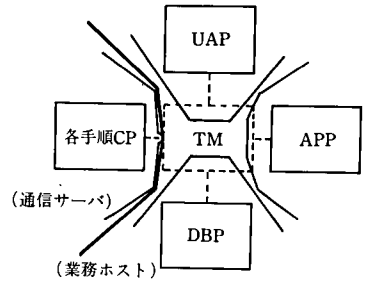
3. 特　　徴

1) H手順の採用で得られるメリット



P1：ホスト間接続プロトコル
P7：端末ーホスト間接続プロトコル
OSI-STK：OSIプロトコル・スタック
（OSIモデル4層～7層）
OSI-MHS：開放型システム間相互接続
電子メールシステム

図1 接続形態



UAP： ユーザ・アプリケーション・プロセス
APP： アプリケーション・プロセス
(API,運用管理,セキュリティ,障害管理等)
DBP： データベース・プロセス
TM： プロセス間インタフェース(トランザクション管理)
各手順CP：各種手順対応コミュニケーション・プロセス

論理的な分散単位
実装での分散例

図2 EDIコンセプトの分散システム構造（論理的構造）

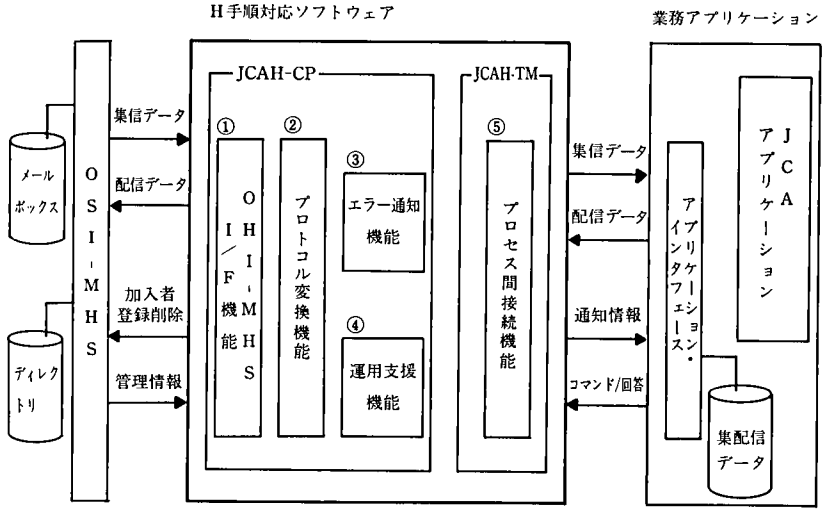


図3 システム構成図

- ISDN回線を使用することで高速通信が可能
  これにより通信コストが軽減．また受発注処理等，業務の効率もアップ
- OSI-MHSを採用しているため，電子メールを用いたメッセージ交換により，社内メール等きめ細かい業務の支援が可能
- 漢字データの伝送が可能
- JCA手順との互換性では，JCA手順でのデータ交換フォーマットがそのまま使用でき，また運用方式も従来のものを活かすことが可能
- EDI（Electronic Data Interchange：電子データ交換）のための標準フォーマットにも，トランスレータを導入するだけで容易に対応可能

2) 通信サーバとしての特徴

- 四つのプロセス(各手順CP，DBP，APP，UAP)を最小単位に，論理的に分散されたシステム構造(EDIコンセプト*)に準拠している（図2)．
- そのため，通話サーバとしてU6000シリーズ上でH手順対応を行う(H手順CPに相当)だけで，業務ホスト上の既存JCAアプリケーションに接続が可能

4. 構　　成

本ソフトウェアは，二つの機能体と五つの機能要素から構成される（図3)．

1) JCAH-CP……JCAH-CPはH手順のホスト側対応ソフトウェアである．業務アプリ

* EDIコンセプト：ビジネスプロトコルから通信プロトコルまでを包含したEDIに対して，Aシリーズ，2200シリーズ，Uシリーズ等各プロダクトが個別に対応するのではなく，分散システム構造を前提にした各プロダクト共通的な考え方．

ケーションとOSI-MHSの中間に位置し，その間でメッセージを中継するプロセスである．業務アプリケーションから渡された取引先宛のメッセージをOSI-MHSのUA (User Agent：利用者機能体)に渡す．他のホスト宛のメッセージは即座に転送され，端末宛のメッセージは該当端末のメールボックスに格納される．また，取引先から自社ホスト宛に送信されたメッセージをOSI-MHSのUAから取り出し，業務アプリケーションに渡す．

JCAH-CPは，以下の四つの機能要素から構成される．

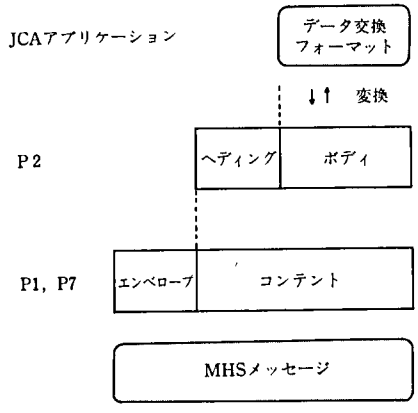
① OSI-MHS I/F

OSI-MHSのUAのUAL (User Agent Layer：OSI7層のサブレイヤ) 関数とコマンド・インタフェースを使用し，OSI-MHSに対してメッセージの集/配信処理，およびOR (Originator/Recipient：発信者/受信者) 名等のMHS管理情報のアクセス・インタフェースを持つ．

ⓐ メッセージの集信
ⓑ 中間ファイルの生成
ⓒ 集信プロセスの起動
ⓓ 集信メッセージの削除依頼
ⓔ メッセージの配信
ⓕ MHS管理情報の検索
ⓖ OR名の登録/削除

② プロトコル変換機能

OSI-MHSと受け渡しするメッセージのフォーマットは，P2 (個人間メッセージ通信プロトコル) で規定された形式であり，このP2形式からJCAデータ交換フォーマットへの変換，あるいはその逆変換を行う．



③ エラー通知機能

OSI-MHSが検出したエラー情報，および本ソフトウェアが検出したエラー情報を，接続ホスト上の業務アプリケーションに通知する．

〈エラー通知情報〉

ⓐ 本ソフトウェアで発生したエラー
ⓑ タイムアウト発生
ⓒ OSI-MHSで発生した集信時のエラー
ⓓ OSI-MHSで発生した配信時のエラー
ⓔ プロトコルエラー
ⓕ セッション確立不可
ⓖ 回線障害発生
ⓗ 加入者情報エラー
ⓘ 集信データ書き込みエラー
ⓙ 配信データ読み込みエラー

④ 運用支援機能（コマンド機能）

接続ホストから発行されたOSI-MHSおよび本ソフトウェアに対するコマンドを受信し，該コマンドの処理を実行する．

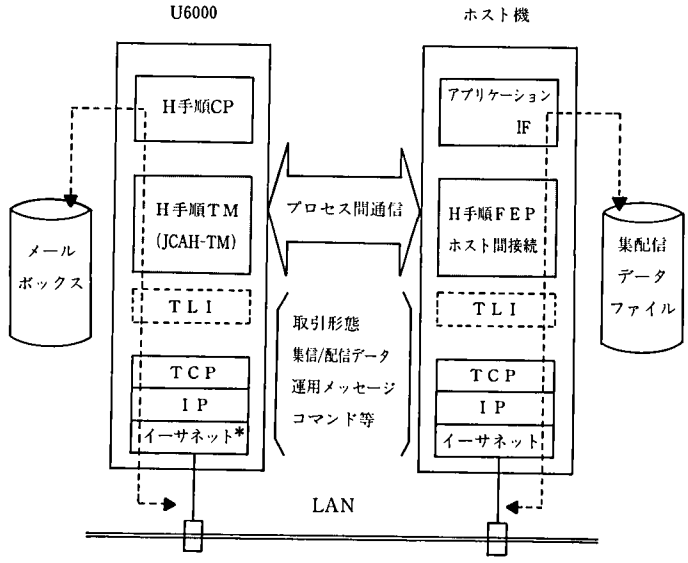
〈コマンドの種類〉

ⓐ 加入者（登録・削除）……H手順のプロトコル構成に応じてOSI-MHSに対し取引先（加入者）の登録・削除を行う．
ⓑ 配信状況検索……OSI-MHSに対して配信済みのメッセージの状態検索を行う．
ⓒ CP閉塞……本ソフトウェアに対し閉塞，および閉塞解除を行う．
ⓓ CPトレース……ログ情報の取得のため電文のトレース採取を行う．

2) JCAH-TM……FEP側にあり，集信データや配信データ等の各種メッセージを，ホスト側と送受信するためのソフトウェアである．ホスト側にもJCAH-TMに相当するソフトウェアが必要であり，FEP-ホスト上でそれぞれ動作し，双方で各種メッセージのプロセス間通信を行う．

⑤ プロセス間接続機能

OSIモデルのトランスポート層へのインタ



＊イーサネット（Ethernet）は米国ゼロックス社の登録商標である。

図4　プロセス間接続

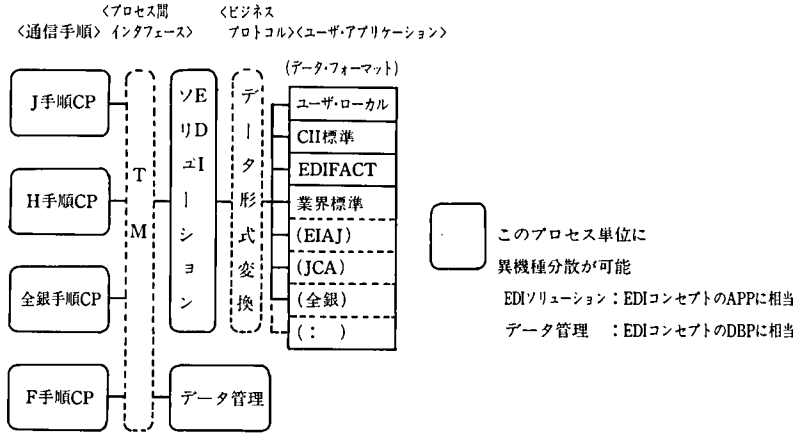


図5　統合 EDI システム概念図

フェースである TLI (Transport Layer Interface) を使用して，FEP—ホスト間で各種メッセージのプロセス間通信を行う(図 4)．

UNIX* System V リリース 4.0 では TLI 関数と共に，トランスポート・プロバイダに TCP/IP が提供されていることから，プロセス間接続のネットワーク・プロトコルには TCP/IP を採用している．

5. 今後の計画

H 手順対応ソフトウェアは当面 A シリーズ・ホストと接続されるが，2200 シリーズ・ホストとの接続機能の開発にもすでに着手している．さらには，U6000 シリーズでのスタンド・アロン型のシステムも将来実現したい．

最終的には，各種通信手順とビジネス・プロトコルを包含した統合 EDI 対応システムを目指しており，機種に依存しない，異機種分散が可能なシステム構造も合わせて実現したいと考えている(図 5)．

＊ UNIX オペレーティング・システムは，UNIX System Laboratories, Inc.が開発し，ライセンスしている．

掲載論文梗概

ネットワーク・アプリケーション・プラットフォーム（略称：NAP）は，デジタル交換機と音声データベースをAシリーズ・コンピュータに統合し，COBOL等の高級言語や4GLに電話機・ファクシミリとの通信機能を提供するプラットフォームである．NAPによって，音声やファクシミリイメージをAシリーズ・コンピュータで直接入出力することが可能となった．三輪次郎は，**NAP（音声データ処理）の機能と特徴**の中で，NAPを構成するハードウェアとソフトウェアについて述べ，簡単な音声メールシステムを想定して，NAPにおける音声処理の動作について解説している．

今日，多くのAI応用システムは手作りシステムとなっている．その背景として，「知識獲得のボトルネック」が実用的なシステム開発の問題点として浮かんできた．このような問題解決には，目的に特化した開発ツールの開発が一つの方法である．ツールの開発には，タスク指向型と領域特化型の二つのアプローチが考えられる．鈴木常彦・寺野隆雄・工藤隆司・上林俊之・伊神克典・飯田賢は，**汎用的な知識ベースを持つ変電所停電操作支援システム：QUALTES**の中で，エキスパートシステム「QUALTES」では領域特化型アプローチにより，適度な一般化レベルを見極めた上で，汎化知識ベース（GKB）を構築し，対象領域における可搬性を高めたことを報告している．

手書き図面の自動認識技術は，CADデータの初期入力負荷を軽減できることから注目されている．また，図面の自動認識にはエキスパートシステムを代表とする知識工学の手法が有効と考えられている．蔵田幸一・藤城雄二・東野康臣は，**図面の自動認識――東日本旅客鉄道（株）の信号連動図表作成システムにおける例**の中で，図面の自動認識システム実用化にあたって認識率とともにポイントとなる，①認識図形の追加（知識の追加）が容易であること，②認識エラーに対しユーザが容易に対応できること，等を中心に，採用した手法を述べている．

掲載論文梗概

ISSN 0914-9996

技　報
UNISYS TECHNOLOGY REVIEW
Vol. 12　No. 4　(No. 36)

発行日　平成5年2月28日
編集人　柳生孝昭
発行人　富田和夫
発行所　日本ユニシス株式会社
東京都江東区豊洲1-1-1 〒135
TEL(03)5546-4111（大代表）
印刷所　三美印刷株式会社