ISSN 0289-6257

技報

UNIVAC TECHNOLOGY REVIEW

1988年2月　第16号

論　文

技術動向

日本ユニバック

NASTRANなどの有限要素法解析ソフトウェアでは，入力データの作成・修正のためのプリプロセッサと出力を評価するポストプロセッサの充実が求められる．中村暢樹と野田安則の**汎用有限要素法プリ・ポストプロセッサAPPEX**は，カラー表示やCADソフトウェアとのインタフェースなどに特色をもつプリプロセッサと，各種の評価図を作成しその保存・再表示を行うポストプロセッサで構成されるAPPEX（Advanced Pre-Post System for Expert）の概要を紹介している．

自動車の内板部品のような複雑な形状のモデリングでは，稜線に沿った複数のフィレット面が集まる頂点で，ボカシ面（すべての周辺面に滑らかに接続する曲面）を創成する必要が生じる．一般に現在の商用CADシステムでは，有効なボカシ面の創成機能が用意されておらず，そのために多大な操作工数を要している．藤井省の***N*辺ボカシ面創成機能の開発**は，任意個の辺を持つボカシ面を周辺面データのみ入力して自動的に創成する手法について報告する．本機能によって，工数を大幅に削減できたと述べている．

3次元編集設計とは，3次元データをそのままの形で入力し，各種の操作を加えて図面等を出力する設計法を言う．とくに既存の形状部品を組み合わせて形状を合成する点が特徴的なので，編集設計と名付けている．新田光義の**汎用CAD/CAMシステムUNICAD(R)/SURFACEにおける3次元編集設計**は，日本ユニバックの汎用CAD/CAMシステムUNICAD/SURFACEに組み込んだ3次元編集設計コマンドについて紹介している．

枠組壁工法による住宅の建設戸数は増大の一途を辿っている．渡辺寛の**枠組壁工法住宅における構造の自動設計システム**は，枠組壁構造の自動設計システムの事例を紹介している．同システムは，構造体の自動認識や構造部材の自動選択・自動創成等を行うもので，技術者のもつ構造設計の知識を抽出し構造記述言語で記述し，それを構造データベースに蓄積する方式を採っているのが特徴となっている．

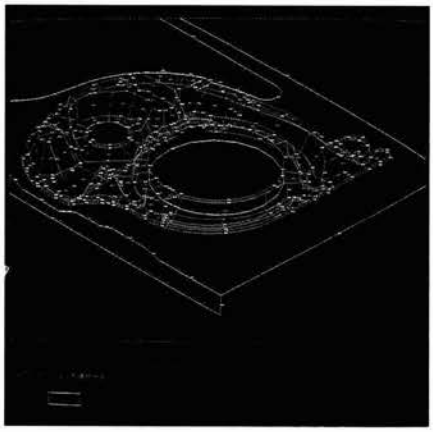
日本ユニバックのソフトウェア工学味見会では，企業でのソフトウェア開発に活用するためにソフトウェア工学の味見を行ってきた．その第1報は「仕様記述の味見」であり，本稿はその第2報である．峰尾欽二の**ソフトウェア設計法の味見**は，彼等の評価した二つのソフトウェア設計法の結果について報告している．ひとつは，表示的意味記述の枠組による仕様からのCOBOLコードの作成法であり，もうひとつはCSPによる仕様からのModula-2コードの作成法である．これらの評価の結果として，産業界においても十分実用に耐えるとの感触を得たと述べている．

加藤潤三の**JSD仕様の直接実行**は，ジャクソン・システム開発法（JSD: Jackson System Development）の支援環境（JSE: Jackson System Development Environment）の一貫として開発したJSD仕様直接実行系を報告している．なお，JSDの仕様記述のためにJSL（Jackson System Language）と呼ぶ言語を導入した．これによって，JSDを適用したシステム開発におけるラピッド・プロトタイピングが可能になったと述べている．

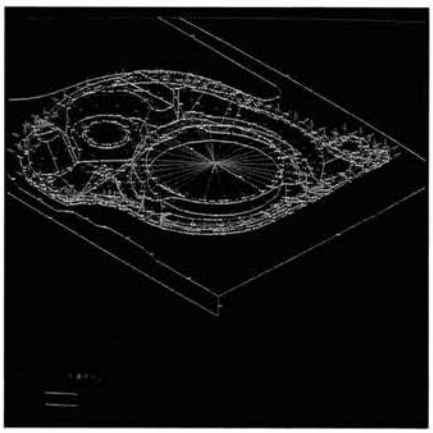
金融機関においては，顧客サービスの充実を目的にオンラインの平日時間延長や休業日運転が行われている．村田豊彦の**24時間運転**は，24時間運転を目指したシステム開発および運用について述べている．なお，24時間運転のためのシステムの目標として，①オンライン終了の意識の不要化，②データベース更新の停止の不要化，③処理量の平準化，④処理の効率化を掲げている．同報告では，これらの目標を踏まえて，問題点の把握と解決例の紹介を行っている．

☆

［有限要素法プリプロセッサ APPEX の実行例］



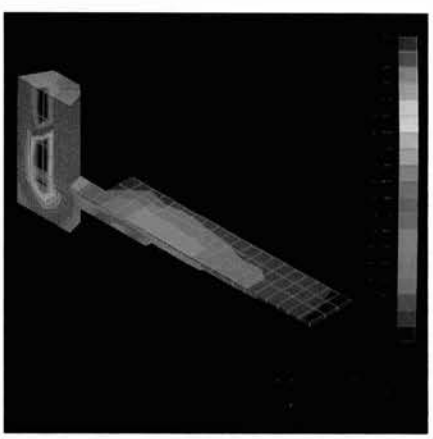
口絵 1　形状創成



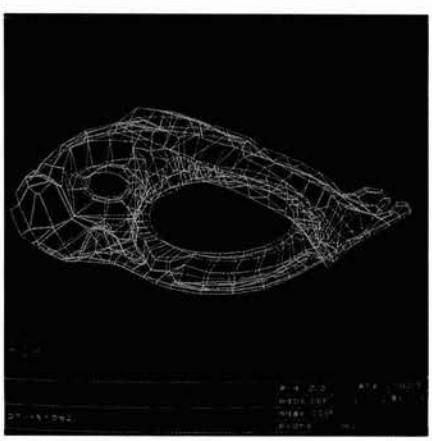
口絵 2　有限要素分割および荷重拘束

［有限要素法ポストプロセッサ APPEX の実行例］

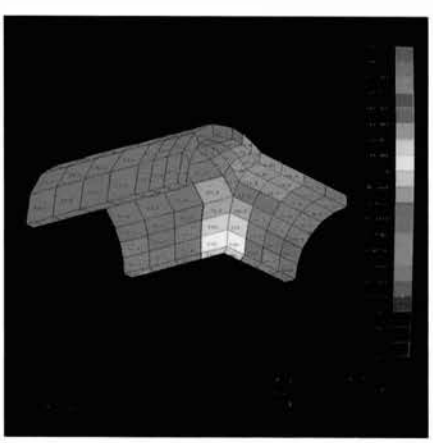
口絵 3　主応力のテンソル矢印図



口絵 4　凝固時間の等高線図



口絵 5　変形図と原形図の重ねがき



口絵 6　主応力の有限要素色分け図(数値の重ねがき)

詳細は，本文「汎用有限要素法プリ・ポストプロセッサ APPEX」に掲載．

論文 汎用有限要素法プリ・ポストプロセッサ APPEX

FEM Pre/Postprocessor APPEX

中村暢樹, 野田安則

要約 有限要素法ソフトウェアの入力データ作成と解析結果の表示を会話型に行うプリ・ポストプロセッサ APPEX (Advanced Pre-Post System for FEM EXpert) を紹介する. プリプロセッサは, 形状定義と有限要素定義を行う. このプリプロセッサは, CAD ソフトウェアとのインタフェースをもつので, CAD データを取り込んだ後に有限要素の定義を行うこともできる. ポストプロセッサは, 各種有限要素法ソフトウェアとインタフェースをもち, 変形図, 等高線図, 矢印図など豊富な評価図の作成とその保存・再表示ができる.

Abstract APPEX is a pre and post processor for FEM packages to make input data and display results.

Pre-processor defines FEM object shapes by using various APPEX commands, as well as obtains the data from other CAD systems.

Post-processor displays, saves, and redisplays the calculated pictures such as deformation, contour line, stress vector chart , etc.

APPEX, a useful interactive system with many functions, will reduce manpower in object shape modeling, FEM data definition and evaluation of analysis results.

1. はじめに

有限要素法プログラム NASTRAN の入力データは, 構造物を細分化した有限要素とそれを構成する節点から成り, 構造物が大型化・複雑化するに従って入力データの作成・修正作業にかかる期間・工程数が増大する. このため, その短縮・削減が要求されてきた.

また, NASTRAN の解析結果は節点・有限要素単位に変位量・応力が数値で出力されるので, そのままでは傾向・特徴をつかむのは不可能である.

これらを解決する日本ユニバックのプログラムには, 入力データ作成プリプロセッサ SIGMA, 解析結果表示ポストプロセッサ NASPLOT, 両方を兼ね備えた GIFTS があり, 長年にわたって使用されてきた. しかし, 近年これらのプログラムでは, 以下のような課題を解決すべきことが明らかになってきた.

プリプロセッサの問題点:

1) バッチ処理プログラムである.
2) 構造物形状を定義するコマンドとその機能が少ない.
3) 定義できる節点と有限要素数に制限がある.
4) 荷重・拘束・有限要素特性が定義できない.
5) カラー表示・隠線消去など表示機能が不足している.
6) 限られた有限要素法解析プログラム (以下では解析プログラムと略す) や CAD プログラムとしかインタフェースをもたない.

ポストプロセッサの問題点:

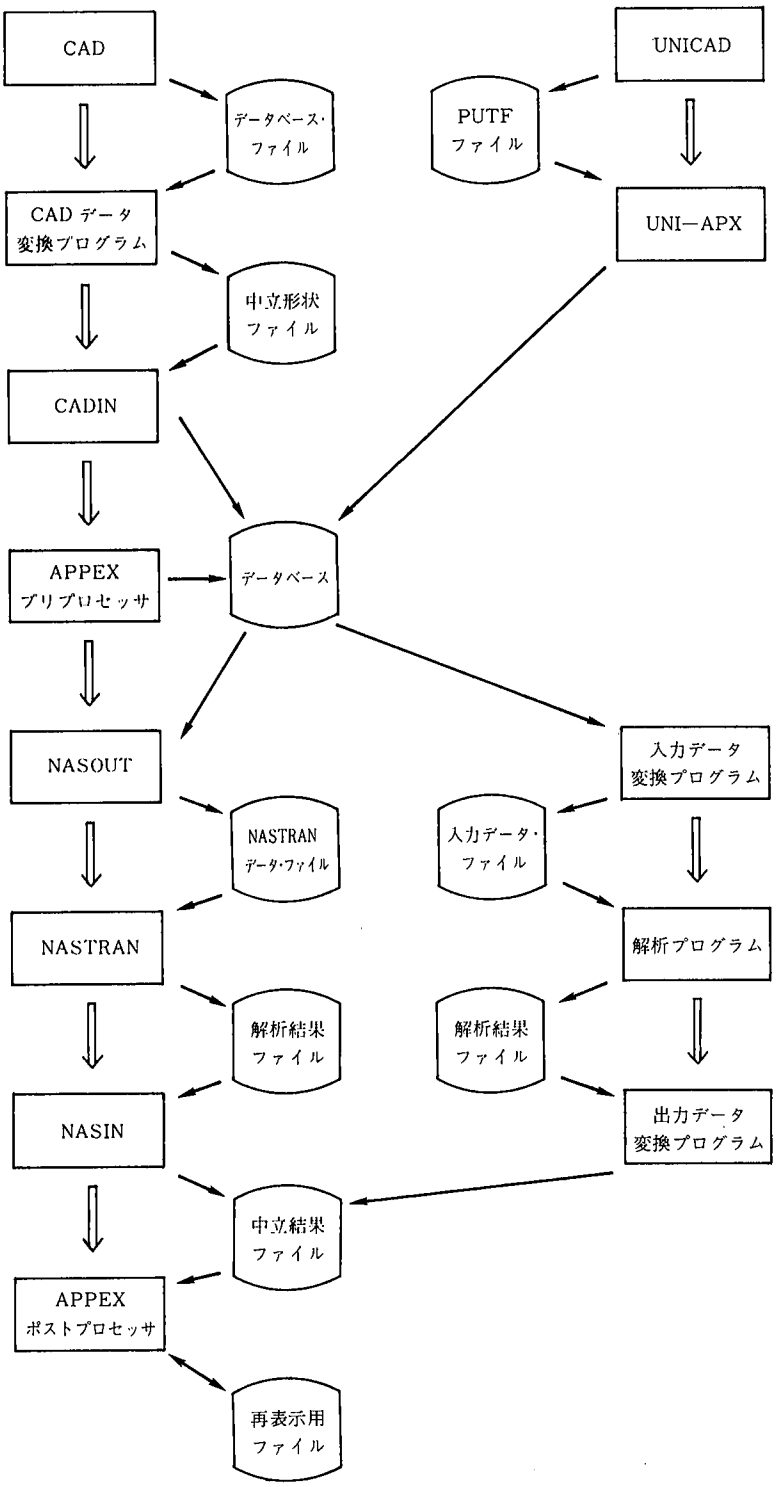


図1　APPEX の概要

Fig. 1　Outline of APPEX system

1)　解析結果の表示種類や内容が限られている.

2)　立体要素や荷重・拘束の表示ができない.

3)　カラー表示や隠線・隠面消去など表示機能が不足している.

4)　限られた解析プログラムの結果しか表示できない.

上記問題点を解決するため APPEX を開発した. プリプロセッサは 1986 年 8 月, ポストプロセッサは 1987 年 2 月に発表され, 現在も機能追加・改良を行っている.

本稿では, プリプロセッサとポストプロセッサごとに各々の概要や機能について説明するが, 全体の処理とデータの流れは図 1 のようになる.

2. プリプロセッサ

2.1 プリプロセッサ概要

プリプロセッサ（以下プリと略する）は, 有限要素法プログラムの入力データ作成をグラフィック端末から会話型で行うシステムである. プリは, モデル創成モジュール (APPEX), NUK-NASTRAN 専用インタフェース (NASOUT), UNICAD® 専用インタフェース (UNI-APX) から構成されており, 他の CAD プログラムや有限要素法プログラムとのインタフェースを追加できる構造になっている（図 2）.

1)　形状モデルの創成……図形要素（端点・線要素・面要素・立体要素）を組み合

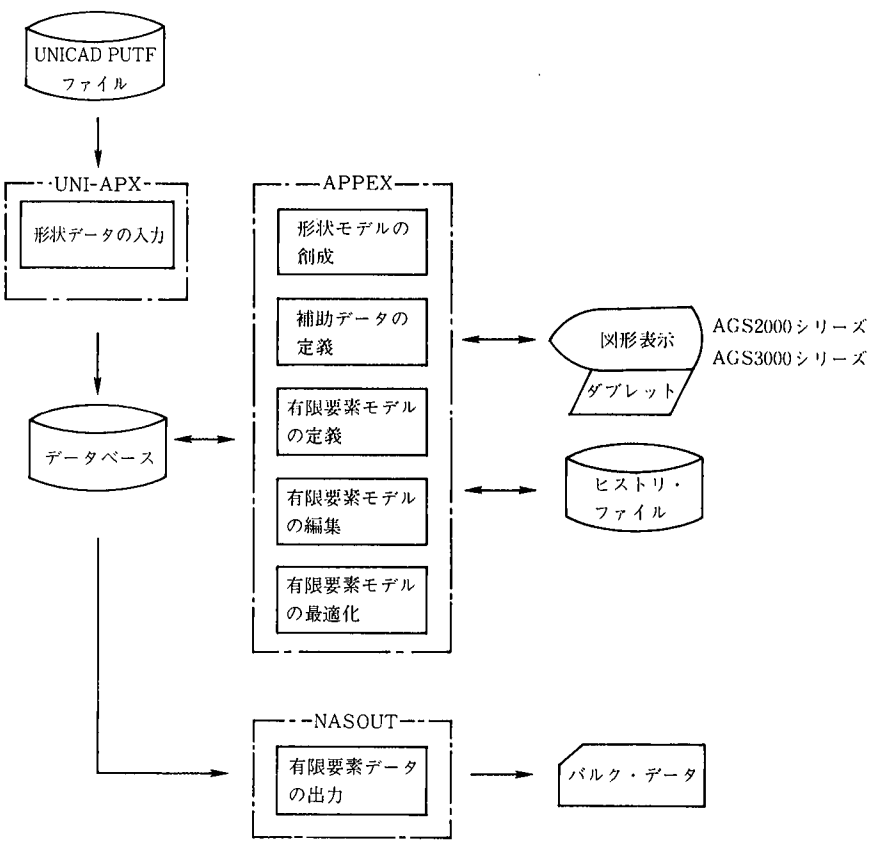


図2　APPEX（プリプロセッサ）の機能概要

Fig. 2　Function of APPEX (preprocessor)

わせて有限要素法解析の対象構造物形状を創成する．

2) 補助データの定義……局所座標系や形状関数などの形状モデルや有限要素モデルを定義するのに必要な補助的なデータの定義を行う．

3) 有限要素モデルの定義……形状モデルに解析プログラムのデータとなる節点と有限要素を定義する．重複節点の除去を行い，荷重・拘束条件と有限要素特性を定義する．

4) 有限要素モデルの編集……有限要素モデルを微調整するために，節点と有限要素の削限，追加および節点の移動を行う．

5) 有限要素モデルの最適化……どの有限要素とも接続のない節点を除去したり，有限要素法解析時のバンド幅が最小となるよう節点番号の入れ替え再付番を行う．

6) 有限要素データの出力……有限要素モデルを解析プログラムのデータ形式で出力する．

7) 形状データの入力……UNICAD システムより提供される UFIO ファイルを読んで，その中に定義されている点・直線・円・円弧・自由曲線・自由曲面のデータをプリのデータベースへ登録する．

1)から 5)は会話型で順次実行されるが，1)から 3)までは任意の回数だけ繰り返される．

2.2 形 状 定 義

最初に解析対象構造物の形状を定義する．対象構造物は，図形要素つまり，端点，パラメトリック 3 次の線要素・面要素・立体要素を組み合わせて表現する（図 3）．図形要素は，図形要素創成コマンドによって創成され，データベース上では名称で識別・管理される．

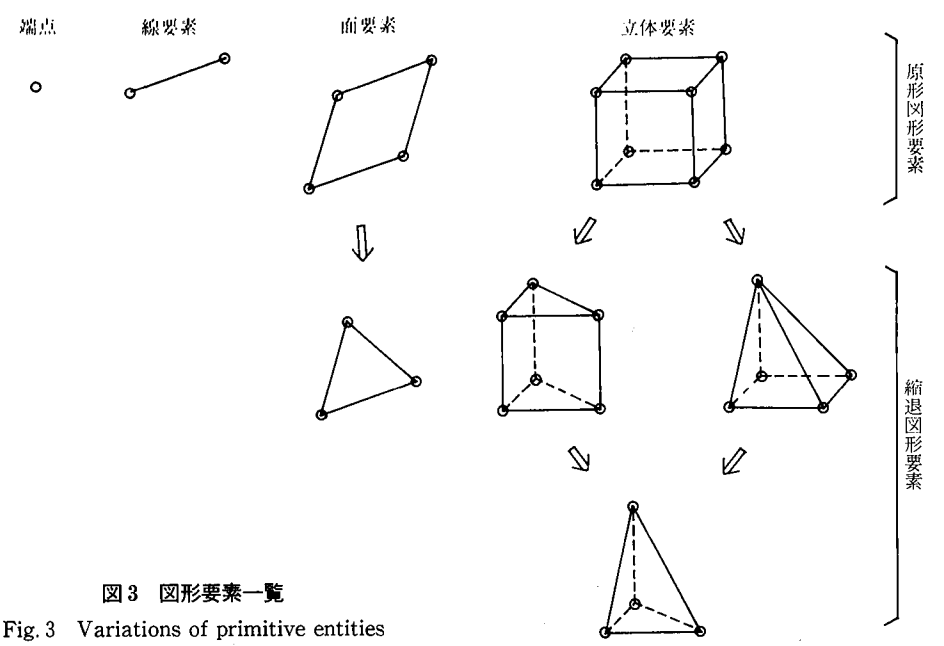


図3 図形要素一覧

Fig. 3 Variations of primitive entities

コマンドは創成方法により次のように分類される．

1)　位相的に低次元の図形要素から高次元の図形要素を創成する．ただし，図形要素の次元の順位は次のようになる．

端点＜線要素＜面要素＜立体要素

［例］　指定した端点を直線で結んで面要素を創成する．

他にスプライン補間，オフセット，スイープ等がある．

図4　面要素創成

Fig. 4　Example of patch definition

2)　位相的に同次元の図形要素を座標変換し複写する．

［例］　面要素を平行移動させる．

他に回転移動，鏡面対称移動，拡大・縮小がある．

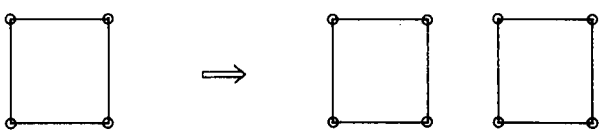


図5　面要素移動

Fig. 5　Example of patch translation

3)　位相的に高次な図形の一部分を取り出して図形要素とする．

［例］　面要素の辺を線要素として取り出す．

他に交点・交線，面上点・面上線等がある．

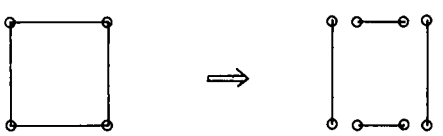


図6　線要素創成

Fig. 6　Example of line definition

4)　他の図形要素に接する図形要素を創成する．

［例］　指定した端点を通り，線要素に接する接線を創成する．

他に接円弧や共通接線等がある．

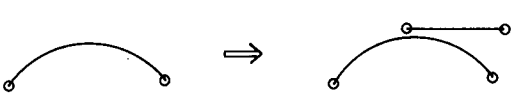


図7　接線創成

Fig. 7　Example of tangent definition

三角形要素・四面体要素・五面体要素は，面要素・立体要素の縮退形として取り扱うので，データベース上は特別扱いしない（図3参照）．

創成される図形要素は，要素の各頂点に置かれた端点を介して他の図形要素と相対関係をもつ．この頂点と端点の関係を接続性とよぶ．隣り合う図形要素が同じ名称の端点を介していると図形要素同志は接続しているが，同座標・異名称の端点を介しているときは図形要素同志は接続していない(図8，図9)．なお，この接続性が後出の重複節点除去で重要となってくる．

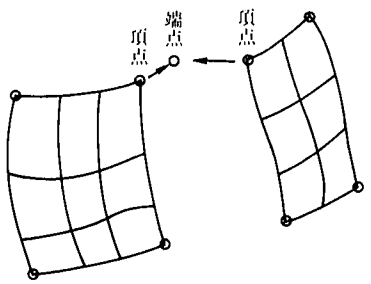


図8　接続する面要素

Fig. 8　Two patches to be connected

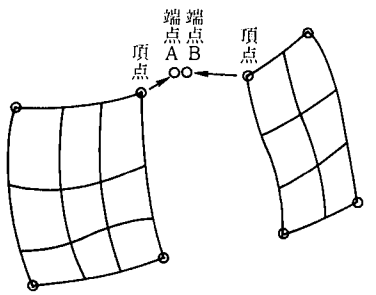


図9　接続しない面要素

Fig. 9　Two patches not to be connected

以上のようにプリを使って形状定義を行うが，他のCADシステム（UNICAD, UNICAD-SOLID等）を使って形状定義を行った場合は，そのデータをプリ用データに変換後使用できる．

2.3　有限要素データの定義

形状定義の後で，解析プログラムの入力データとなる節点・有限要素・要素特性，荷重・拘束を順次定義していく．

1)　節点……節点は図形要素単位に定義する．各パラメタ方向の分割点数と分割比を与え，パラメタ空間上の等比間隔格子を図形要素空間に写像することにより節点を発生させる（図10）．

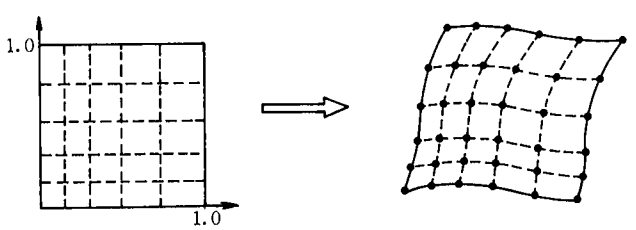


図10　節点の発生

Fig. 10　Example of node generation

2)　有限要素……指定された種別の有限要素によって，あらかじめ節点が定義済みの図形要素を覆って定義する．取り扱える要素種別は，図11に示すように6種類あり，それぞれに1次，2次（中間節点1個），3次（中間節点2個）の要素があ

る．解析プログラムが用意している多種の有限要素種別と対応させるために要素サブタイプがある．要素サブタイプが異なれば，表示上は同じ種別で同じ次数の有限要素であっても別有限要素として扱う．また，有限要素の覆い方には複数種あり，方向オプションによって制御する(図 12)．方向オプションを変えることにより，図形要素に有限要素を多重定義することができる．

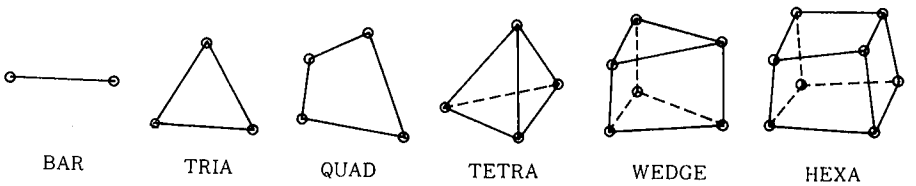


図 11　有限要素種別一覧

Fig. 11　Variations of elements

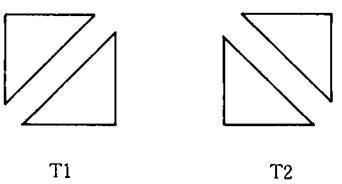


図 12　方向オプション

Fig. 12　TRIA element generation patterns

3)　重複節点の除去……節点と有限要素の定義は，図形要素単位に処理されるので，図形要素境界では同座標・異名称の節点が多数存在したり，分割節点数が異なったりして，境界に隣接する有限要素同志が力学的に関係付けられていない．このままでは有限要素モデルとしては不完全なので，次のような手順で完全なモデルにしていく．

①　図形要素の各辺（面）を調べ，辺（面）の接続性が同じ図形（辺・面を構成する端点名が同じ図形）同志の境界における分割節点数が異なる場合は節点数の少ない方に合わせる．有限要素はそれに従って自動的に縮退される．これを境界一致作業という（図 13)．

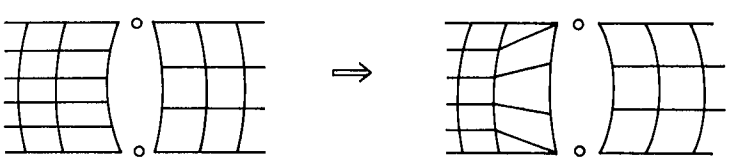


図 13　境界一致作業

Fig. 13　Connecting process of boundaries

②　共有する境界上の対応する節点同志の距離が有意であっても無視して同一視する（図 14)．

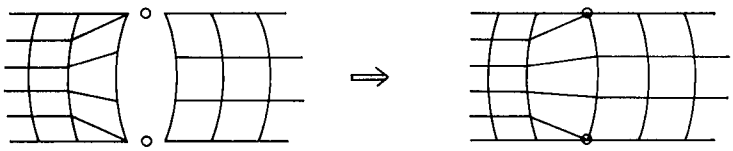


図 14　重複節点除去

Fig. 14　Elimination of redundant nodes

以上の方法を位相的重複節点除去とよんでいる．また，幾何的重複節点除去，つまり同座標節点群を同一視するだけで境界一致作業を行わないことも可能である．

4)　要素特性・荷重・拘束……定義した有限要素に板厚・断面性能・材料特性名などの要素特性諸情報および荷重・拘束も付加できる．形状関数を使い各有限要素または節点のパラメタ空間上での位置を媒介として形状関数を評価し，その関数値を入力データとする．これによって不均一に分布する板厚や荷重を定義することも可能である（図 15）．

定義可能な荷重・拘束は次の 5 種である．

① 集中荷重：節点に直接加わる力とモーメント

② 分布荷重：物体表面に作用する分布力

③ 強制変位：節点強制変位，BAR 要素の初期変形および節点自由度拘束

④ 温度荷重：有限要素に加わる熱荷重，熱流束および節点・有限要素初期温度

⑤ 剛体拘束：節点間の自由度拘束

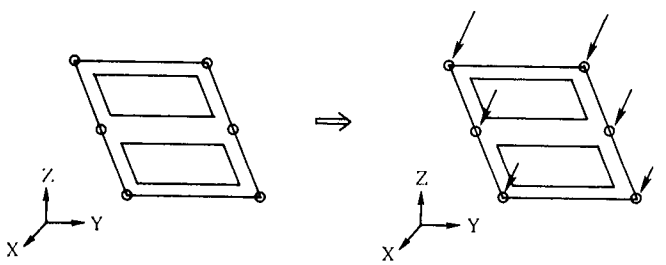


図 15　形状関数を使った集中荷重定義

Fig. 15　Example of concentrated load definition by using a shape function

2.4 操　　作

システムの操作は，コマンド・モードとグラフィック・モードの 2 種類がある．コマンド・モードでは，端末のキーボードからコマンド文字列を入力する．グラフィック・モードでは，コマンドをタブレット上のメニュー，データを表示画面上の図形ヒット，ロケータ入力，メニュー選択，キーボード入力などによって行う．グラフィック・モードでは，図形名称や座標値を直接入力する必要はほとんどない．

入力したコマンドの履歴は，ヒストリ・ファイルに順次記録されるので，操作の再現や障害時のデータベース修復のほか，類似操作の繰り返しに利用される．

また，マクロ機能が用意されているので，コマンドを組み合わせ，一連の手順を一つのまとまりとして登録することにより，定型作業の操作が簡略化できる．

2.5 今 後 の 課 題

CAD プログラムとの対応について考えてみると，それらのプログラムでは各々が独

立した専用のデータベースを持っているので，プリで形状を修正しても CAD プログラムにはその修正が反映されない．CAD プログラムのデータベースを共通化すればこの問題は解決されるが，個々のプログラムに合わせたプリの改良が必要となり，汎用性が失なわれてしまう．このため CAD プログラムからデータをとり込むだけでなく，CAD プログラムへデータを渡すインタフェースが必要である．

3. ポストプロセッサ

3.1 概　　要

ポストプロセッサ（以下ポストと略す）は，解析プログラムの処理結果をグラフィック端末を用いた会話型作業によって編集するためのシステムである．

汎用のポストであるためには，解析プログラムへの依存性をなくす必要がある．本システムでは，3.3節に述べるようなデータの抽象化を行うことにより，これを実現している．有限要素法において基本となるデータは，解析対象物の幾何学的形状を記述する節点，節点間の結合関係を記述する有限要素，および節点または有限要素に対して出力される解析結果である．これらを，かりに《FEM 的位相の入ったデータ》と呼ぶことにするならば，有限要素法による解析プログラムであるかどうかを問わず，「《FEM 的位相の入ったデータ》に対して APPEX のポストは適用可能である」と言える．

解析プログラムとのインタフェースをとるのが変換プログラムである．変換プログラムは具体的なデータの抽象化を行い，《FEM 的位相の入ったデータ》を作成する．

ポストの位置づけは図1を参照のこと．ポストはプリとは独立に使用できる．

3.2 評　価　図

ポストで作図可能な評価図は以下のものである（表1）．

1) 変形図……解析結果のうち節点の並進変位を用い，解析対象構造物の変形を変位量を強調して形状図として表示する（口絵5参照）．

2) 有限要素色分け図……有限要素に関して計算された任意のスカラー量の値の分布状況を多色で塗り分けることにより表示する．スカラー量の値域をいくつかの区間に分け，値が同一の区間内にある有限要素を同一のグループとして，その区間に割り当てられた色で塗り分ける（口絵6参照）．

3) 等高線図……節点に関して計算されたスカラー量について，その値の連続的な分布状況を等高線図で表示する（口絵4参照）．

4) 円図……節点または有限要素に関して計算されたスカラー量について，その値の分布状況を円により表示する．中心，半径，線種または色で，それぞれ計算された位置，値の絶対値，および値の正負を表す．

5) ベクトル矢印図……節点または有限要素に関して計算されたベクトル量について，その値と方向の分布状況を矢印により表示する．

6) テンソル矢印図……節点または有限要素に関して計算されたテンソル量について，その分布状況を矢印により表示する．テンソルの成分（スカラー量）を双頭の矢印にて表示し，計算された成分の座標軸の方向を矢印の方向で，絶対値を長さで，正負値を矢頭の向き（内/外）でそれぞれ表す．これは，応力の評価に主に

表1　評価図一覧

Table 1　Summary of calculated pictures

評価図	目　　的	対象とする物理量のデータ型	対象とする物理量の節点量・要素量の区別	使用例
変形図	解析対象物の変形状態を表示する	ベクトル	節点量	変形状態 固有モード
有限要素色分け図	スカラー量の要素ごとの分布状況を表示する	スカラー	要素量	歪の特定成分
等高線図	スカラー量の連続的な分布状況を表示する	スカラー	節点量	等応力線 等温線
円図	スカラー量の節点または要素ごとの分布状況を表示する	スカラー	節点量 または要素量	温度 応力の特定成分
ベクトル矢印図	ベクトル量の節点または要素ごとの分布状況を表示する	ベクトル	節点量 または要素量	速度ベクトル 熱流束
テンソル矢印図	テンソル量の節点または要素ごとの分布状況を表示する	テンソル	節点量 または要素量	主応力テンソル 主歪テンソル
梁・トラス図	1次元有限要素に対する解析結果のうち，軸力，剪断力，ねじりモーメントおよび曲げモーメントを表示する	ベクトル	要素量	軸力 剪断力 ねじりモーメント 曲げモーメント
XYグラフ図	二つのスカラー量間の相関関係を表示する	スカラー	節点量 または要素量	時刻歴応答 荷重応力図 歪・応力図
荷重・拘束図	解析条件を表示する	スカラー またはベクトル	——	荷重 拘束 強制変位

注：データ型および節点量，要素量については3.3節を参照．

使用される評価図である（口絵3参照）．

7)　梁・トラス図……1次元の有限要素を用いた有限要素法による構造解析（すなわち，骨組構造解析）の結果を評価するための図である．軸力，剪断力，ねじりモーメントおよび曲げモーメントを表示できる．

8)　XYグラフ図……スカラー量とスカラー量の相関関係（時刻歴応答・荷重応力関係など）をXYグラフ上の折れ線として表示する．

9)　荷重・拘束図……他の評価図とは異なり，解析結果ではなく解析条件を表示する．荷重・拘束および強制変位を，スカラー量の場合は円で，ベクトル量の場合は矢印で，作用または課されている位置に表示する．

3.3　データの抽象的な形式

ポストへの入力は，解析プログラムが扱う（解析プログラムの入力または出力となる）データである．これらは，標準的には，節点および有限要素，解析条件および解析結果であるが，その扱い方は解析プログラムにより異なる．そこで，ポストが解析プログラムに依存しないために，これらに対する抽象的な扱い方を以下のように設定する．

1)　有限要素……形状および節点の配置具合にのみ着目し，分類されている（プリと同様，図11）．

2)　荷重・拘束……構造解析における用語を使用し，解析条件のことを荷重・拘束

とよび，次の4種類に分類する．

・集中荷重：節点に作用する荷重
・分布荷重：面積に作用する荷重
・拘束　　：節点に課す拘束
・強制変位：節点に課す強制変位

3) 物理量……解析結果，および一般に評価図作成の対象となる量のことを物理量とよぶ．

・節点量：節点に関して計算される量
・要素量：要素に関して計算される量
・評価点：物理量が計算される位置（節点量に対する評価点は節点であり，要素量に対する評価点はたとえば要素図心・積分点などである．）
・層　　：面要素に対する板厚方向の位置（面の表裏，多層板における各層の位置）
・ケース：物理量を各解析条件ごとに分類するための識別子
・データ型：物理量の意味を捨象し，数値としての型に着目したもの（スカラー，ベクトル，テンソル）

物理量に対しては単項演算を施すことができる．この演算処理が可能であることにより，不要な物理量を保存する必要がなくなる．

3.4 部分図形

評価図は解析対象物全体に対して作成することもできるし，特定の部分に限定することもできる．このような解析対象物の部分のことを部分図形という．部分図形として扱えるものは以下のとおりである．

1) 節点名または有限要素名の集合……節点名または有限要素名の集合を指示し，そこに含まれるものを部分図形とする．
2) 有限要素のタイプ……有限要素のタイプ（種類）を指示し，そのタイプのものすべてを部分図形とする．
3) 領域……領域を指示し，そこに含まれるものを部分図形とする．
4) 任意断面……任意の平面を指示し，その平面によって切り取られる解析対象物の断面を部分図形とする．
5) 表面……解析対象物が立体有限要素を含む場合，その表面を部分図形とする．

3.5 表示オプション

評価図は，XYグラフ図以外はすべて3次元の解析対象物に重ね合わせて表現される．表示オプションは，評価する目的に応じて表示方法を指示するものである．表示オプションの一覧および各評価図に対する指示の可否を表2に示す．

表示オプション以外に，物理量の値を評価図中に表示することや，表示の対象となる物理量の範囲を指定することもできる．

3.6 表示制御機能

XYグラフ図以外の評価図は，任意の視点から見た平行投影図としてグラフィック端末上に表示される．表示制御機能には以下のものがある．

表2　表示オプション一覧
Table 2　Summary of display options

表示オプション ＼ 評価図	変形図	有限要素色分け図	等高線図	円図	ベクトル矢印図	テンソル矢印図	梁・トラス図	XYグラフ図	荷重・拘束図
隠面消去	○	◎	◎	○	○	○			○
陰影づけ	○	○	○	○	○	○			○
表示位置の選択（節点・要素図心・評価点）				○	○	○			
変形図・原形図の選択		○	○	○	○	○	○		○
輪郭線図・要素分割図の選択	○	○	○	○	○	○			○
原形図との重ねがき	○								
荷重の表示	○	○	○	○	○	○	○		○
拘束・強制変位の表示	○	○	○	○	○	○	○		○
全体の輪郭線図との重ねがき	○	○	○	○	○	○	○		○
シュリンク表示	○	○	○	○	○	○	○		○

◎　：無条件に行われる
○　：指示可能
空白：指示不可

① ズーム（部分拡大）
② シフト（窓領域の移動）
③ チルト（視点の面外回転）
④ 多色カラー表示
⑤ 陰影づけ
⑥ 隠面消去
⑦ 表示・線種・塗りつぶしパターン等の属性の変更

また，塗りつぶしを伴わない評価図はXYプロッタへも出力できる．

評価図データは保存し，必要なときに再表示できる．したがって，バッチ処理で評価図データを作成しておき，それをグラフ端末上に会話型で表示できる．

3.7 操　　作

操作は，コマンドの入力によって会話型で行うことができる．プリと異なり，画面上に表示される評価図がすべてテンポラリ表示なので，アテンションを伴う入力操作はサポートされない．

入力したコマンドの履歴は，ヒストリ・ファイルに順次記録されるので，操作の再現に利用できる．

3.8 変換プログラム

ポストで扱うデータは，3.3節で述べた抽象的な形式である．これを外部に公開しているのが，カード形式の《中立結果ファイル》である．変換プログラムは，各解析プログラムに応じて存在し，それらの具体的なデータから抽象的な《中立結果ファイル》を作成する．

現在使用可能な解析プログラムは，

・NUK-NASTRAN（汎用構造解析プログラム）
・ARGUS（汎用非線型構造解析プログラム）*
・CAST（鋳物凝固解析プログラム）**
・MELT FLOW（プラスチック射出成型解析プログラム）***

である．

3.9 今後の課題

現在のポストは，次に述べるように“操作性”および“プリとの接続”に改良すべき点がある．これらに対処することが今後の課題である．

1) 操作性……ポストの操作は，すべて端末のキーボードからのコマンド入力で行っているので，メニュー入力およびアテンションを伴う入力に比べ操作しにくい．
2) プリとの接続……ポストはプリとは独立に使用できるものとして設計しているため，プリとポスト間でのデータの受け渡しが行いにくい．プリからポストに渡すことのできるデータは，現在，部分図形としての節点または有限要素の集合のみであり，ポストからプリへはデータを渡すことはできない．したがって，解析結果の有限要素分割や形状モデルの創成にフィードバックすることはむずかしい．

4. 適　用　例

ここでは，構造解析と凝固解析における適用例を示す．

4.1 プリプロセッサ

・形状創成（口絵1）
・有限要素分割および荷重拘束（口絵2）

4.2 ポストプロセッサ

・主応力のテンソル矢印図（口絵3　解析プログラム：NUK-NASTRAN）
・凝固時間の等高線図（口絵4　解析プログラム：CAST）
・変形図と原形図の重ねがき（口絵5　解析プログラム：NUK-NASTRAN）
・主応力の有限要素色分け図(数値の重ねがき)(口絵6　解析プログラム：NUK-NASTRAN)

5. おわりに

現在のプログラムは，プリ・ポストともに所期の目的を達したと思われる．すなわち，本稿の冒頭で指摘した問題点に対し，プリについては，有限要素法解析に必要なほとんどのデータを会話型で容易に作成することが可能となり，またポストについては，有限要素の種別を問わず多種の評価図を作成することが可能となった．また，プリは多くのCADプログラムおよび解析プログラム，ポストは多くの解析プログラムとインタフェースをとることができ汎用性を実現した．

今後とも，ユーザでの使用の経験を生かして，APPEXの機能の改良に取り組んでゆ

---

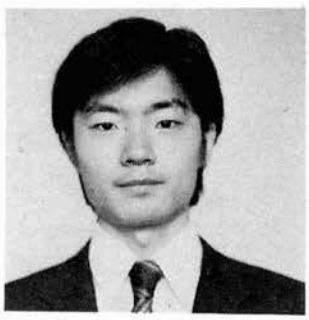
* ARGUS：米国 APPLIED COMPUTER METHODS CORPORATION 社のソフトウェア
** CAST ：日本ユニバックのソフトウェア
*** MELT FLOW：日本ユニバックのソフトウェア

きたいと思っている．

本システムの開発にあたり，指導・協力をいただいた日本ユニバックの渡部義維，稲葉聰，古賀正志，長島毅の各氏ならびに開発を担当した榎本誠，毛利明，北原義章，ユニ・ソフト・エンジニアリング(株)の向江幸次，渡辺美樹の各氏に感謝の意を表したい．

執筆者紹介　中　村　暢　樹 (Masaki Nakamura)

昭和32年生．56年学習院大学大学院自然科学研究科終了，同年日本ユニバック(株)入社．構造解析プリ・ポストプロセッサの保守を経てAPPEX (PRE) の開発に従事．



野　田　安　則 (Yasunori Noda)

昭和31年生．57年山梨大学大学院工学研究科終了，同年日本ユニバック(株)入社．数値計算および構造解析関連ソフトウェアの保守・開発を経てAPPEX (POST) の開発に従事．

論文

N 辺ボカシ面創成機能の開発

Development of a Capability for Generating a Smooth Corner Surface with Arbitrary Number of Boundaries

藤井　省

要　約　自動車の内板部品のような複雑な形状をモデリングすると，稜線に沿った複数のフィレット面が集まる頂点位置で，必ずボカシ面が必要になる．しかし，これまでUNICAD®/SURFACEには，有効なボカシ面コマンドが用意されていなかったため，ボカシ面創成に多くの操作工数を要していた．また，筆者が知る限り他社の汎用CADシステムでも，有効なボカシ面機能を持つものは少ない．

その主な理由は，ボカシ面の場合，辺の個数が任意である（4辺や3辺とは限らない），創成すべき形状があいまいである，という特徴をもつために図形処理上の扱いがむずかしいということがあげられる．

このように，現在の形状モデリングにとって未解決な課題の一つであるボカシ面創成に関し，今回，周辺面のみを入力として，任意個の辺を持つボカシ面を自動的に創成する『N辺ボカシ面創成機能』を開発しUNICAD/SURFACE上で実現した．本機能の実現により，ボカシ面創成工数を大幅に削減できた．

Abstract　It is indispensable for modeling real-life complicated shape (e. g. an inner panel of a car body) to effectively generate a surface at a corner, where several fillet surfaces meet. Since UNICAD/SURFACE has not provided the effective commands for generating corner surfaces, it has been extremely time consuming to generate them by trial and error. As far as the author knows, there are few general purpose CAD systems which have this function. The reason seems that the number of boundaries is not fixed, and that the shape to be generated is ambiguous. Thus this has been considered to be one of the major remaining problems in geometric modelling.

We have developed a capability for generating smooth corner surfaces with given arbitrary number of boundaries, and has implemented it into UNICAD/SURFACE. As a result, the man-hour required has drastically decreased.

This paper describes the problems, requirements, and details of the algorithm, and finally evaluates the results.

1. はじめに

形状モデルには，曲線モデル，曲面モデル，および立体モデルがある．CAD/CAMシステムでは，それぞれ目的に応じた形状モデルを採用しており，日本ユニバックの汎用CADシステムUNICADでは，以下のようになっている[1][2]．

- UNICAD/SOLID……立体モデル
- UNICAD/SURFACE……曲面モデル
- UNICAD/DESIGN……曲線モデル

曲面モデルでは，複数の単一曲面を張り合わせて複雑な形状を表現する．UNICAD/SURFACEは，曲線・点等を入力として単一の曲面を創成するコマンドを約30個用意

し，単一曲面に対する豊富な形状創成機能を提供している．しかし，複数の曲面が関係する曲面創成，とくにボカシ面の創成については，有効な専用コマンドが用意されていない．このため，既存のコマンドを組み合わせ試行錯誤的に対応せざるを得ず，非常に操作工数がかかるという問題をかかえていた．

自動車の内板部品のような複雑な形状をモデリングすると，稜線に沿った複数のフィレット面が集まる頂点位置で，必ずボカシ面が必要になる．しかし，ボカシ面は，辺の個数が任意であり（4辺や3辺とは限らない），創成すべき形状があいまいであるという特徴をもつために，図形処理上の扱いがむずかしい．筆者が知るかぎり他社の汎用CADシステムでは，立体モデルであるMODIF[3]以外には優れたボカシ面創成機能を持つものは少ない．

このように，ボカシ面の創成は，現在の形状モデリング技術の一つの未解決な課題と考えられる[4][5]．また，ボカシ面は人目に触れるような意匠面として用いられることは少なく，ハイライトの通り，曲率連続性，隣接面との連続性などの厳密な面品質が要求されないことが多い．このため，ボカシ面の創成に必要以上に工数をかけたくないという，ユーザからの強い要求がある．

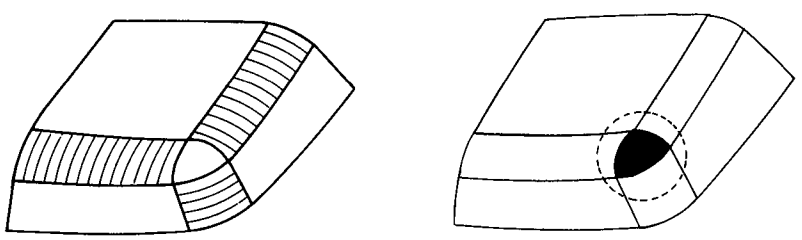
今回，任意個の辺数のボカシ面を自動的に創成する『N辺ボカシ面創成機能』を開発し，UNICAD/SURFACE上で実現した．本機能は，周辺面の指示のみを入力として，周辺面の傾向を反映し，かつすべての周辺面に滑らかに接続するボカシ面を創成する．この機能の実現により，ボカシ面創成工数が大幅に削減できたので報告する．

2. ボカシ面創成機能の問題点と要求

2.1 ボカシ面とは

フィレット面創成には，相貫線に沿う滑らかに連続する曲面の創成（稜線への面定義，図1(a))*と，複数の相貫線が交わる頂点位置での曲面の創成(頂点への面定義，図1(b))という二つの課題がある．

頂点への面定義で創成される曲面を，『ボカシ面』と呼ぶ．複数の相貫線が交わる頂点付近では，相貫線に沿って創成したフィレット面（図2-(a),(b),(c)）の端側が



(a) 稜線への面定義　　(b) 頂点への面定義

図1 フィレット面の創成

Fig. 1 Fillet surface along an edge and surface at a corner

* 以下では，とくに断らない限り，稜線への面定義で創成される曲面のことをフィレット面と呼ぶことにする．

重なる（図2-(d)）．そこで，フィレット面をトリム* し穴をあけた後（図2-(e)），頂点位置へ曲面を創成しなければならない．その曲面は，周辺の曲面の形状が延長されて溶け込んだボカシ形状を持つため，ボカシ面と呼ぶ．

2.2 ボカシ面の創成法の現状の問題点

現状の UNICAD 曲面コマンドには外部仕様上の制約として次の2点があり，ボカシ面の創成には大きな操作工数が必要である．

1) 曲面の辺の個数は3ないし4個に限られており，5辺以上の曲面は創成できない．

2) 多くのコマンドでは，創成曲面の形状を代表する曲線が，入力データとして要求される．

したがって，ボカシ面は通常次のいずれかの方法で作成される．

1) 複数の4辺の曲面で作成する方法（図3-(a)）

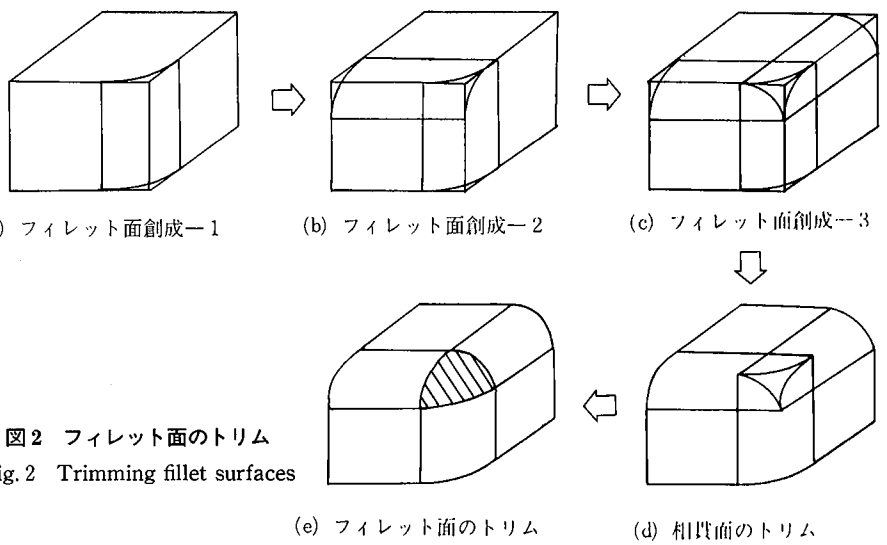


図2 フィレット面のトリム
Fig. 2 Trimming fillet surfaces

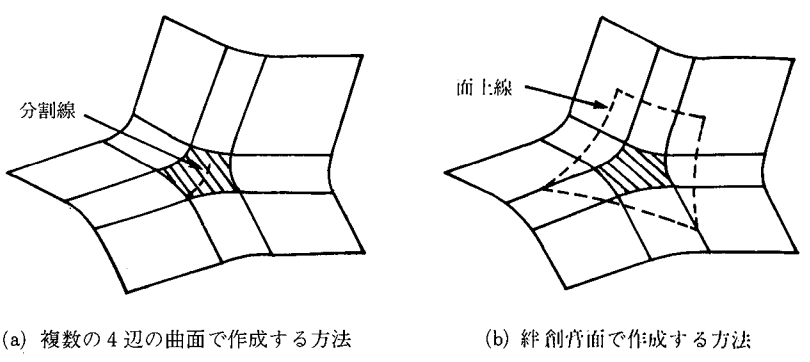


図3 現状でのボカシ面の創成法（斜線部は穴）
Fig. 3 Conventional methods for generating corner surface

* 面上線で曲面を切り取り，曲面の有効部分を定義することを「トリムする」と呼ぶ．

2) 絆創膏面で作成する方法（図3-(b)）

第1の方法は，ボカシ面を4辺の曲面に分割する曲線を作成し，4本の境界線を拘束条件とする曲面を創成する．このとき，分割線としてボカシ面の形状を代表する曲線を作成するのがむずかしい．適切な境界線が与えられないため，周辺形状を反映したボカシ面ではなく，単に穴を埋める曲面が創成されることが多い．

第2の方法では，穴の領域内に曲面を創成するのではなく，穴を含む広い領域に4辺の曲面を創成する．これが，穴に絆創膏を張って穴を隠すのに似ているため，この曲面を絆創膏面と呼んでいる．この方法では，第1の方法のように空間上の曲線ではなく，周辺面の面上線を4本与えればよい．面上線のためボカシ面の形状を代表する曲線は比較的与えやすいが，操作工数が多くかかることが問題である．表1は，図4の絆創膏面を創成したときの操作工数の例である．

2.3 ボカシ面コマンドへの要求

ボカシ面コマンドへの要求項目を整理すると，以下の4点が挙げられる．

1) 操作性が良いこと……N個の周辺面を入力するだけで，それらの曲面に接続するボカシ面を創成できることが必要である（図5）．

2) 処理の安定性が高いこと……少ない入力で確実にボカシ面が創成できることは，形状モデリング上効果が大きい．

3) ボカシ面の形状（“ふくらみ”）は，周辺面の形状の傾向からシステムが自動的に決定できること．

4) 1個のボカシ面を表す曲面およびパッチ数は，極力少ないこと．

表1 絆創膏面創成工数の例

Table 1 Operation time in adhesive tape method

操作内容	操作工数(分)
4個の面上点を結ぶ曲線を作る	3
曲線を各々の曲面に投影する	20
投影線を結合して4個の曲線を作る	2
4個の曲線から曲面を作る	2
合　計	27

(測定は UNIVAC 1100/61 II)

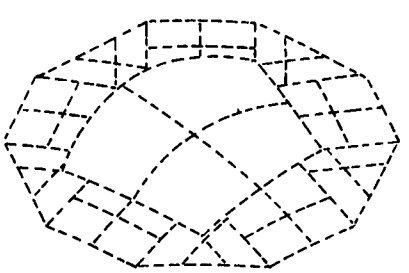


図4 絆創膏面の創成例

Fig. 4 An example of corner surface by adhesive tape method

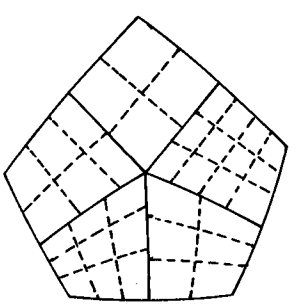


図5 N面表現（$N=5$）で創成されるボカシ面

Fig. 5 A corner surface consisting of N surfaces

3. ボカシ面コマンドの概要

本章では，UNICAD/SURFACE で実現したボカシ面創成機能について述べる．

3.1 ボカシ面コマンドの外部仕様

1) 機　　能……$N(3 \leqq N \leqq 10)$ 個の曲面* の枠線** で囲まれた領域に，周辺面に接続するボカシ面を創成する．

2) 入　　力

① 周辺面：$N(3 \leqq N \leqq 10)$ 個の曲面の枠線．N 個の枠線によって，閉じたループを構成していなければならない．

② ボカシ面の表現：N 面もしくは 1 面/ 2 面を指定する．

・N 面は $3 \leqq N \leqq 10$ のときに指定でき，ボカシ面は N 個の曲面で表現される（図 5）．

・1 面/2 面は，$N=4, 5, 6$ のときに指定でき，1 個（$N=4$ のとき）または 2 個（$N=5, 6$ のとき）の曲面で表現される（図 6）．これは，実用上 N は 3 から 6 が多く，かつそれらの場合に曲面の個数を減らしたいために，考えられた表現である．

③ 面分割点……$N=5, 6$ で 2 面表現を指定したときにのみ，指示する必要がある．創成されるボカシ面は，この分割点の位置で 2 個の曲面に分割される（図 6-(b),(c)）．

・$N=5$ のときは，対になる枠線の中点（P_1）と端点（P_2）を指定する．

・$N=6$ のときは，対になる枠線の端点（P_3）と端点（P_4）を指定する．

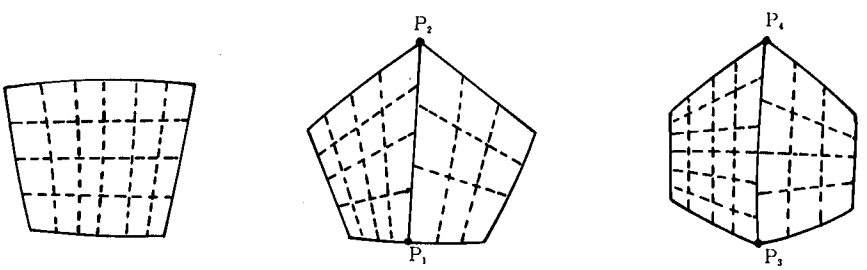
3) 出　　力……N 個，1 個もしくは 2 個の双 3 次曲面．

3.2 ボカシ面コマンドの内部仕様の概要

ボカシ面の創成処理は，4 個のモジュールから成る（図 7）．

1) 形状規定曲線の創成……各々の周辺面の枠線の中点とボカシ面の中央点とを結び，ボカシ面の形状を代表する N 個の曲線（形状規定曲線）を求める（図 8）．これらの曲線は，次の条件を満たしている．

① 周辺面の枠線の中点では，形状規定曲線の端点での接線ベクトルが周辺面の



(a) $N=4$　　(b) $N=5$　　(c) $N=6$

図 6　1 面/2 面表現で創成されるボカシ面

Fig. 6　Corner surfaces represented by one surface or two surfaces

* ここでの曲面とは，自由曲面または自由曲面を面上線でトリムした曲面(部分面)である．

** 枠線とは，曲面の有効部分を規定する面上線のことである．

接平面上にある．

② ボカシ面の中央点では，N 個の形状規定曲線の端点での接線ベクトルが同一平面上にある．

この形状規定曲線は，ボカシ面の形状を決める上で極めて重要である．

2) N パッチ表現のボカシ面（N パッチ近似面）の創成……ボカシ面を近似する N 個の1パッチ曲面を求める（図9）．これらの曲面は，形状規定曲線，周辺面の枠線，およびスロープ・ベクトル* をもとにして求められる．なお，求まった隣り合う N 個の1パッチ曲面は接平面連続になっている．

3) N 面表現のボカシ面の創成……N パッチ近似面から，ボカシ面を表す N 個の曲面を求める（図5）．このとき，1曲面は $m\times n$ パッチ（ただし，m，n は N 個の曲面ごとに異なる）で構成される．

4) 1面/2面表現のボカシ面の創成……N パッチ近似面と分割線（$N=5$ または $N=6$ のとき）から，ボカシ面を表す1個または2個の $m\times n$ パッチのボカシ面を求める（図6）．

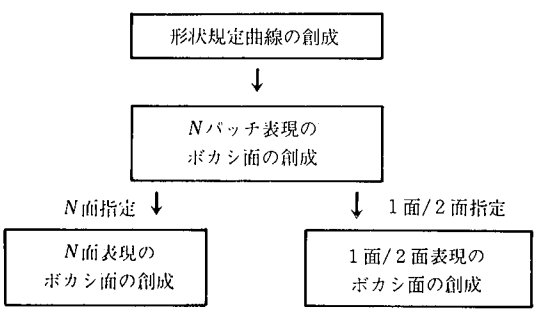


図7 ボカシ面創成処理のモジュール構成

Fig. 7 Program module

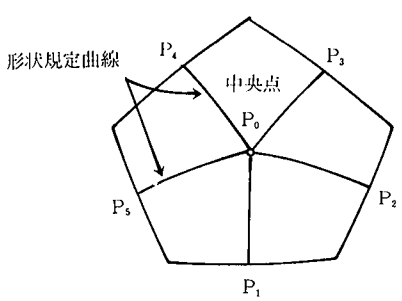


図8 形状規定曲線と中央点

Fig. 8 Constraint curves and center point

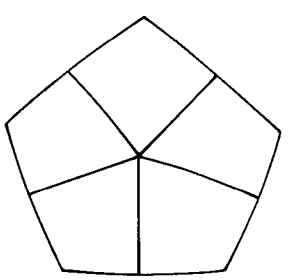


図9 N パッチ近似面の創成

Fig. 9 Approximation by N patches

* スロープ・ベクトルとは，曲面パッチの境界線に対して直交方向の曲面の接線ベクトルのことである．

4. アルゴリズムの詳細

本章では，ボカシ面創成処理の各モジュールのアルゴリズムについて述べる．

われわれが採用した形状規定曲線創成アルゴリズムは，MODIF を参考にしている．つまり，Bezier 曲線の制御点を順々に求め，それらを移動することで，ボカシ面の中央点および形状規定曲線を決める方法である[3]．ただし，ボカシ面の形状を決める上で重要な意味を持つ，形状規定曲線の接線ベクトル長の決め方にわれわれの独自性がある．

また，MODIF は，1セグメント曲線，1パッチ曲面で形状を表現しているため，計算された N パッチ近似面は周辺面といたるところで接平面連続である．一方，UNICAD は，n セグメント曲線，$m\times n$ パッチ曲面で表現しているため，N パッチ近似面の形状をもとにして，周辺面との境界線の一致および周辺面との接平面連続性を必要な精度内で保証する曲面の創成が必要である．

これらを十分な精度で保証することを，N 面表現のボカシ面創成および1面/2面表現のボカシ面創成では考慮しなければならない．

4.1 形状規定曲線の創成

形状規定曲線は，1セグメントの3次 Bezier 曲線として求める．ここで，入力された N 個の周辺面の枠線を境界線と呼び，境界線の中点を $P_i(i=1,\cdots,n)$，N 個の形状規定曲線の端点が一致する点を中央点 P_0 とする（図8）．

1) 境界線の中点 P_i における，単位接線ベクトル t_{i3} を求める（図10）．両隣の境界線の単位接線ベクトル t_{i1} と t_{i2} の平均ベクトル（$t_{i3}=(t_{i1}+t_{i2})/2$）を求め，次にこれを P_i における周辺面の接平面に投影する．

2) 形状規定曲線が2次 Bezier 曲線であると考えて，P_i における接線ベクトルの大きさを決める．

① P_1，…，P_n の平均の点を，基準点 P_b とする．

② 点 $P_iP_bt_{i3}$ で決まる直角三角形（図12）を考え，2次 Bezier 曲線の制御点を $P_iQ_iP_{i0}$ とする．ここで，$P_iP_b /\!/ Q_iP_{i0}$ を満たすように P_{i0} を定める．

このとき，Q_i を直線 P_iP_{iv} 上で動かすことにより P_i での接線ベクトルの大きさを制御できる．

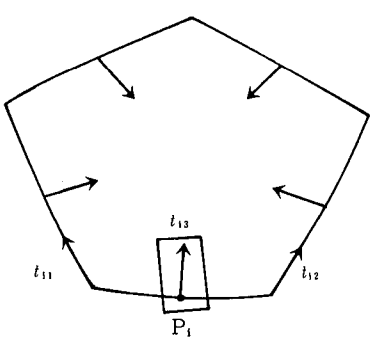


図10　P_i でのスロープ・ベクトル
Fig. 10　Slope vector at P_i

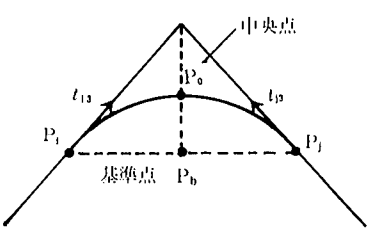


図11　中央点と基準点
Fig. 11　Center point and base point

形状規定曲線が円弧状に延び，ボカシ面が球状になるのが自然なボカシ方であるから，図12で$x=y$となるようにQ_iを決める．

よって，接線ベクトルの大きさLは

$$L=2x=2d/(1+\cos\theta)$$

となる．

3) 中央点P_0を求める．

2次Bezier曲線の制御点Q_iの平均の点を，中央点P_0とする．

P_1, …, P_nが同一平面α上にあり，接線ベクトルt_{13}, …, t_{n3}がP_bを通りαに垂直な軸に対称な場合（半球形状のボカシ面）は，ボカシ面の中央点P_0が対称軸上にのる（図11）．

4) 3次Bezier曲線の制御点を求める．

P_iQ_iの2：1内分点，Q_iP_0の1：2内分点を，制御点P_{i1}，P_{i2}とする（図13）．

5) 中央点P_0を通りN個の制御点P_{i2}で決まる平均的な平面を考え，P_{i2}をその平面に投影する（図14）．これは，N個の形状規定曲線の端点の接線ベクトルを，中央点P_0において同一平面上にのせるために行う．

こうして求まった制御点P_i，P_{i1}，P_{i2}，P_0から決まる曲線を形状規定曲線とする．

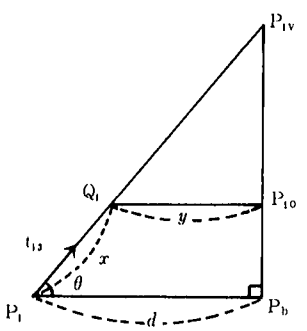


図12　2次Bezier曲線の制御点$P_iQ_iP_{i0}$

Fig. 12　Control points of quadratic Bezier curve

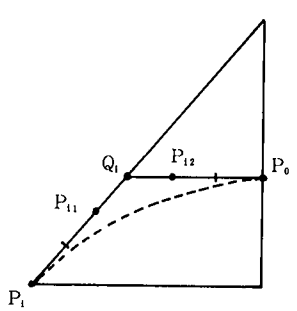


図13　3次Bezier曲線の制御点$P_iP_{i1}P_{i2}P_0$

Fig. 13　Control points of cubic Bezier curve

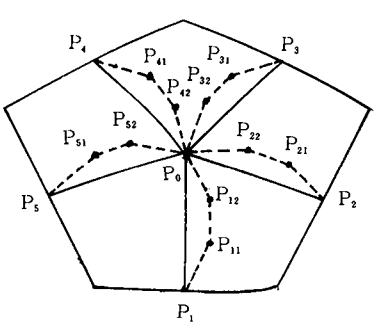


図14　創成される形状規定曲線

Fig. 14　Derivation of constraint curves

4.2 N パッチ表現のボカシ面の創成

N パッチ近似面は，木村，千代倉が提案している Gregory パッチ[3] を用いる．ここで Gregory パッチを用いる理由は，隣り合うパッチの境界線が一致し，かつ境界線の端点で 3 本のパッチ境界線が同一平面上にあれば，接平面連続な二つのパッチが生成可能なためである．

1) 4 本のパッチ境界線を求める……各々の周辺面の境界線を P_i で，2 本の曲線に分割する．分割した曲線を，両端点での単位接線ベクトルを拘束して 1 セグメント Bezier 曲線で近似する．求まった曲線および形状規定曲線をパッチ境界線とする．

2) 4 本のパッチ境界線から，Gregory パッチの制御点を生成し[3]，N パッチ近似面とする．

4.3 N 面表現のボカシ面の創成

N 個の Gregory パッチで近似されたボカシ面に対して各々のパッチを変形し，周辺面と境界線が一致し接平面連続となるような $m \times n$ の双 3 次曲面を創成する．

以下の手続きを，N 個のパッチに対して繰り返す．

1) パッチ境界線 f_1 で周辺面 S_1 に接続する曲面 SRF_1 を創成する（図 15）．

① パッチ境界線 f_1 を，周辺面 S_1 の枠線を近似するような n セグメント曲線に変形する．

② パッチ境界線 f_2 を，曲線 f_1 に対して構成点操作を行う*．

③ パッチ・データからスロープ曲線 g_1**，g_2 を求め，g_1，g_2 を境界線 f_1 と同じ構成点数およびセグメント間スケール比を持つ n セグメント曲線に変形する．

④ 境界線 f_1 の各構成点で，スロープ曲線の接線ベクトルを周辺面の接平面に投影する．

⑤ 境界線 f_1，f_2，スロープ曲線 g_1，g_2 から曲面 SRF_1 を創成する．

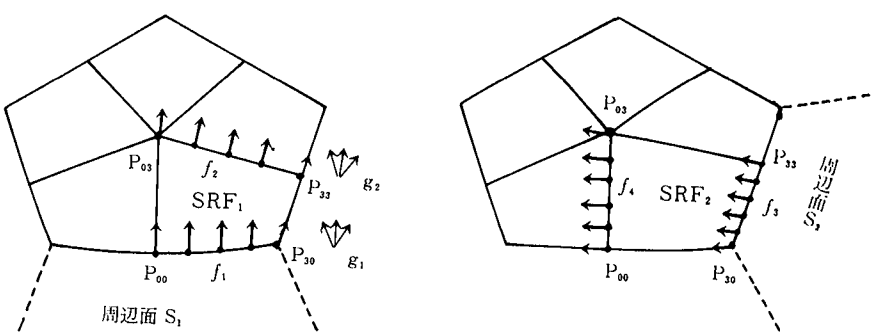


図 15　1 パッチ曲面から $m \times n$ パッチ面への変形

Fig. 15　Deformation from 1 patch surface to $m \times n$ patch surface

---

*　与えられた N 個の曲線に対して，構成点数およびセグメント間スケール比が等しい N 個の近似曲線を求めることを『曲線の構成点操作を行う』と呼ぶ．

**　UNICAD は双 3 次曲面のデータを，パッチ境界線の曲線と，スロープ・ベクトルの曲線化であるスロープ曲線として持っている．

2) 同様にして，パッチ境界線 f_3 で周辺面 S_2 に接続する曲面 SRF_2 を創成する(図15)．

3) Brown[6] のブレンド式①で曲面 SRF_1 と SRF_2 をブレンドしてボカシ面とする．

$$S=\alpha(u,v)S_1+\beta(u,v)S_2 \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots ①$$

$$\alpha(u,v)=u(1-u)/[u(1-u)+v(1-v)]$$

$$\beta(u,v)=v(1-v)/[u(1-u)+v(1-v)]$$

4.4 1面/2面表現のボカシ面の創成

多くの UNICAD 曲面コマンドと同様に，与えられたデータから創成されるべき曲面の面上線（フレーム曲線）を推定し，それらから曲面を創成する．このフレーム曲線の推定に N パッチ近似面が使われる．

1) Gregory パッチで表現された N パッチ近似面を，Coons パッチで近似する．これは，UNICAD では内部表現として Coons パッチを用いているためである．

2) 分割線を求める（2面表現のときのみ必要．図16)．

① 分割点 P_1, P_2 を結ぶ1セグメント曲線 f_d を求める．f_d の端点の接線ベクトルは，P_1，P_2 で N パッチ近似面の接平面上にある．

② N パッチ近似面の中央点での面法線ベクトルを投影方向として，f_d を N パッチ近似面に投影し，分割線 F_d を求める．

以下の 3)から 7)の処理を，N(周辺面の個数)$=4$ のときは1回，$N=5$ または $N=6$ のときは2回繰り返し，それぞれ1個，2個の曲面を創成する．

創成する曲面の4本の境界線を図17-(a)のように，A_{i1}，A_{i2}，B_{i1}，B_{i2} (i=1 あるいは i=1,2) とする．

3) 境界線に対し構成点操作を行う（図17-(b))．

A_{i1} と A_{i2}，B_{i1} と B_{i2} に対しそれぞれ構成点操作を行い，境界線 a_1，a_n，b_1，b_m とする．

4) 境界線から中間のフレーム曲線を求める（図17-(c))．

① 境界線 a_1，a_n の対応する構成点 P_1，P_2 を通り，b_1 と b_m を a_1 側の弧長比でブレンドした曲線 c_i(i=2, n−1) を求める．

② c_i の始終点以外の構成点を N パッチ近似面に投影し，スプラインで補間した曲線を a_i とする．

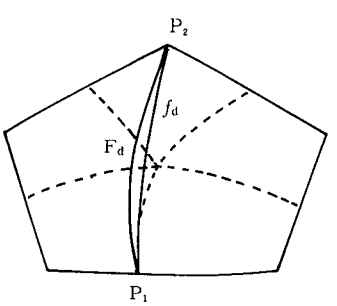


図16 分割線の作成

Fig. 16 Dividing curve

同様に，$b_i(i=2, m-1)$ を求める．

5) フレーム曲線に対し構成点操作を行う（図 17-(d)）．
$a_i(i=2, n-1)$ に対し，$b_j(j=2, m-1)$ との交点を求め，求めた交点，元の曲線の両端の接線ベクトル，および a_1 のセグメント間スケール比から曲線を求め，改めて a_i とする．同様に，$b_i(i=2, m-1)$ を求め直す．

6) フレーム曲線の端点の接線ベクトルを，周辺面の接平面に投影する（図 17-(e)）．

7) フレーム曲線から曲面データを求める．

5. 評価と今後の課題

5.1 評　　価

1) 面品質……図 18, 19, 20, 21 は，創成された 5 辺，6 辺のボカシ面の曲面形状を確認するために，SCULPTOR で求めた工具経路の例である．これらは，凹，凸，およびフラットの混在している典型的な周辺面形状に対して，ボカシ面を創成したものである．図の工具径路は，本機能で創成されたボカシ面が，周辺面の傾向を反映し，なだらかに形状が変化し，かつすべての周辺面と滑らかに接続していることを示している．

2) 処理時間……上例での処理時間を表 2 に示す．N 面表現の場合は，非常に速いといえる．2 面表現は，N 面表現に比べ 5 倍程遅いが，他の UNICAD 曲線コマンドと同程度である．

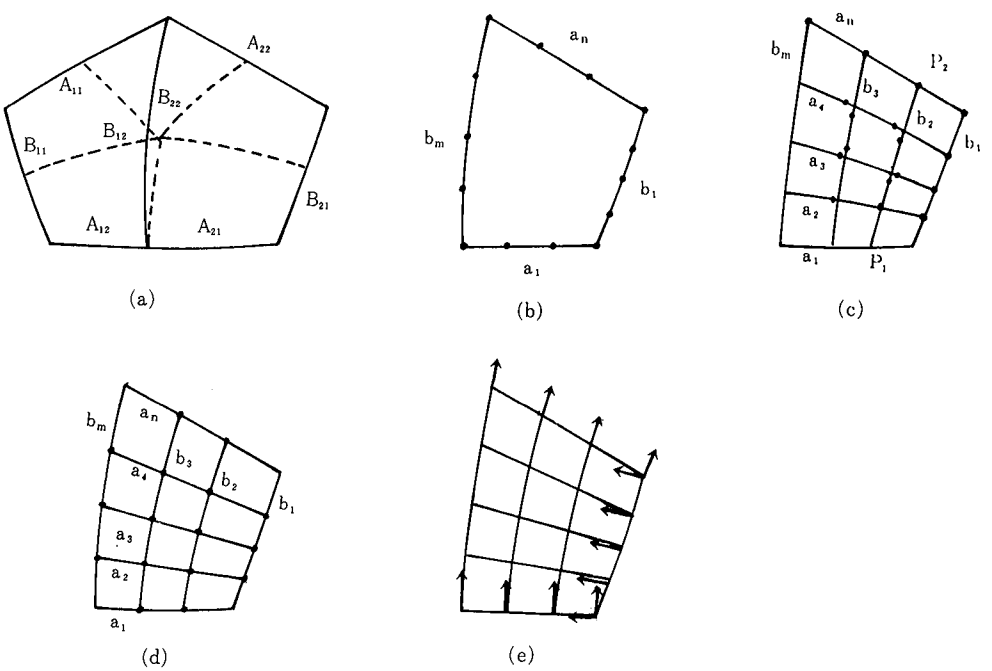


図 17　1 面/2 面表現のボカシ面の創成

Fig. 17　Generation of corner surfaces represented by one surface or two surfaces

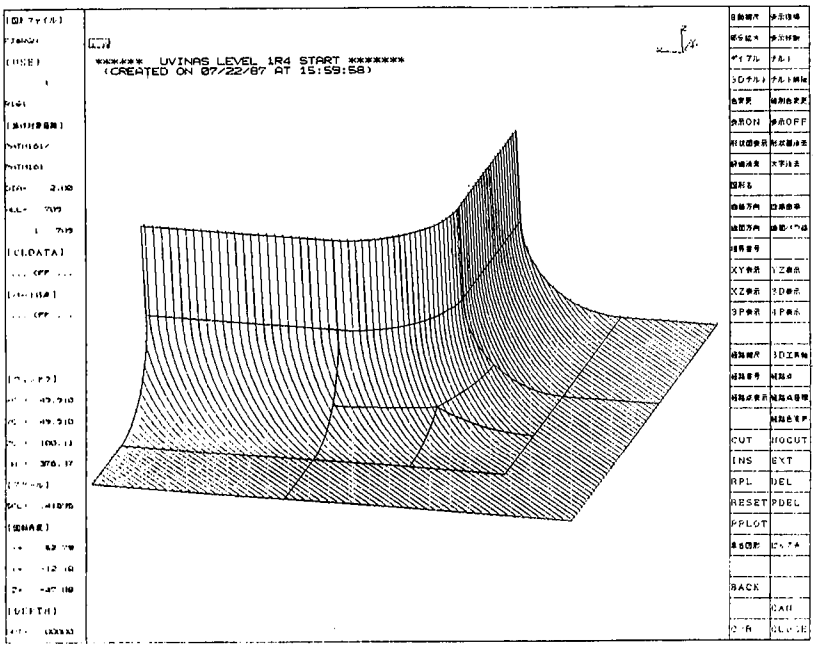


図 18　5 辺の N 面表現によるボカシ面の工具径路の例

Fig. 18　Corner surface (5 boundaries) represented by 5 surfaces

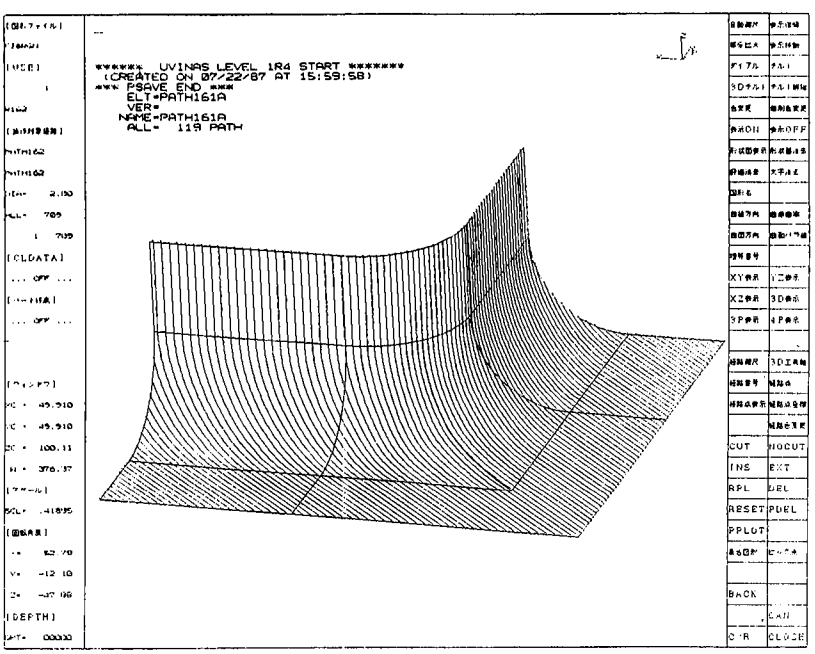


図 19　5 辺の 2 面表現によるボカシ面の工具径路の例

Fig. 19　Corner surface (5 boundaries) represented by 2 surfaces

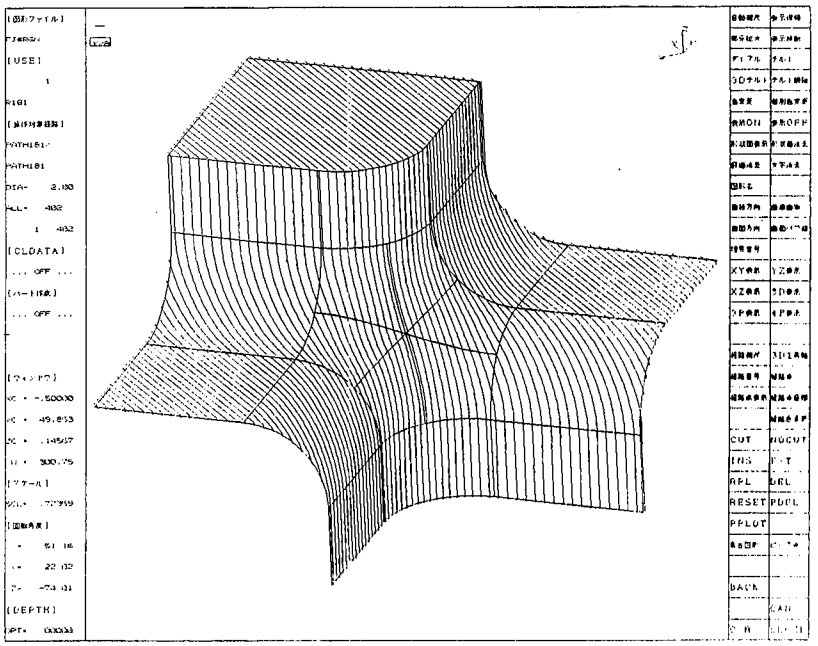


図 20　6 辺の N 面表現によるボカシ面の工具径路の例

Fig. 20　Corner surface (6 boundaries) represented by 6 surfaces

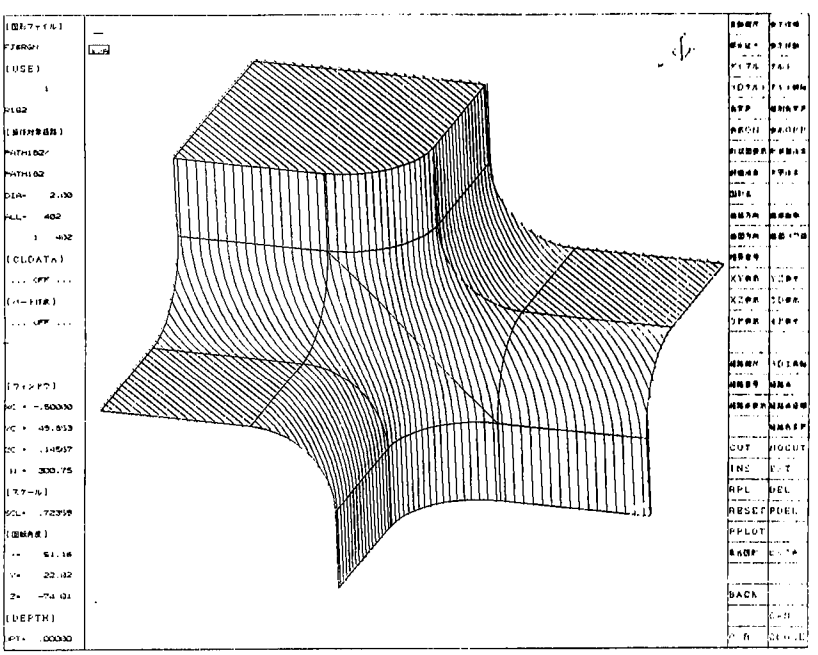


図 21　6 辺の 2 面表現によるボカシ面の工具径路の例

Fig. 21　Corner surface (6 boundaries) represented by 2 surfaces

表2　ボカシ面創成の処理時間の例

Table 2　Response time of the new command

	表　現	演算時間（秒）	入出力（秒）	応答時間（秒）
5辺	N面	17.2	3.6	23
	2面	100.1	1.8	113
6辺	N面	22.8	2.4	27
	2面	107.4	3.3	122

（測定は UNIVAC 1100/61 II）

表3　パッチ・面の個数の例

Table 3　Number of patches and surfaces

	表　現	面の個数	総パッチ個数
5辺	N面	5	71
	2面	2	312
6辺	N面	6	96
	2面	2	384

3)　操作性……周辺面の指示のみでボカシ面が創成でき，操作は非常に簡単である．表2の応答時間と周辺面の入力指示時間とを加えた時間が操作工数である．モデルは違うものの，同等な形状を対象としたときの従来法での操作工数(2.2節　表1）と比べ，大幅な工数削減が実現できた．

4)　N 面表現と1面/2面表現の比較……表3に，上例におけるボカシ面のパッチと曲面の個数を示す．1面/2面表現のときは，面の個数が少ないがパッチの個数が多く，必ずしも良い結果ではない．

5.2 今後の課題

1面/2面表現で創成されたボカシ面は，面の個数が少ない点は良いが，パッチ数が多いことと，処理時間が長いことが問題である．第1の問題は，周辺面の枠線に対し構成点操作を行うときに，曲線の構成点の個数が増えることが原因である．第2の問題点は，点および曲線の曲面への投影計算が多用されるためである．今後は，これらの点を改善してゆきたい．

6. おわりに

ボカシ面創成では，ボカシ面の形状を代表する曲線つまり形状規定曲線の創成がむずかしい．本稿で報告した形状規定曲線の創成法は，簡単な幾何演算に基づいているが，自然な形状のボカシ面が表現可能である．本機能を利用することによって，UNICAD/SURFACE の形状創成機能がより充実するものと確信する．

最後に本稿作成にあたり御指導をいただいた，日本ユニバック CAD/CAM システム二部大高課長に感謝の意を表する．また，本機能の開発は，戸塚玲子氏との共同開発であることを付記する．

参考文献
［1］ UNICAD 解説書・設計製図編，日本ユニバック，資料コード 483845424，1987.
［2］ UNICAD/SOLID 概説書，日本ユニバック，資料コード 198701200，1987.
［3］ 千代倉弘明，ソリッドモデリング，工業調査会，1985.
［4］ 前川佳徳，“金型製作の課題と CAD/CAM 利用の問題点”，PIXEL, No. 59, 1987, pp. 100～105.
［5］ 木村文彦，“形状モデリングと CAD/CAM”，精密工学会誌，Vol. 53，No. 3，1987，pp. 5～8.
［6］ R. E. Barnhill, J. H. Brown, and I. M. Klucewicz, “A New Twist for Computer Aided Geometric Design”, Computer Graphics and Image Processing, 8, 1978, pp. 78～91.

執筆者紹介 藤　井　　省(Satoru Fujii)
昭和 26 生．昭和 52 年，早稲田大学大学院理工学研究科生産工学専攻修士課程修了．同年，日本ユニバック(株)入社．現在，CAD/CAM システムの開発に従事．現在，CAD/CAM システム二部に所属．

論文

汎用 CAD/CAM システム UNICAD®/SURFACE における3次元編集設計

3-D Composite Design Functions in CAD/CAM System UNICAD®/SURFACE

新 田 光 義

要 約 3次元編集設計は，本来3次元であるモデルをそのままの形で入力し，各種の操作を行って図面等を出力する設計法を言う．とくに，既存の形状部品を組み合わせて形状を合成するところから編集設計と名付けた．

これは，図面に基づき3次元形状を頭の中で2次元化して扱う従来の伝統的設計法（前者と対比させ2次元設計とよぶ）よりも，より人間に近い進んだ方向と言える．

本稿では，日本ユニバックの汎用 CAD/CAM システム UNICAD/SURFACE で採用した3次元編集設計用コマンドについて概説する．

はじめに，旧来の2次元設計の欠点について考察し，その後で3次元編集設計でそれがどのように克服されるかを述べる．

Abstract The purpose of this paper is to explain the 3D composite design function, which processes 3D geometries directly and builds up the required one by simply assembling part blocks. It has been implemented in the CAD/CAM system, UNICAD/SURFACE.

At the first we will examine the weakness of traditional 2D design system. And afterwards we will describe the concept of 3D composite design system, and show how to use it.

1. はじめに

従来より実用化されている設計方法は，2次元設計とも言うべきもので，設計者が本来3次元形状である製品を頭の中に思い浮かべて，ある手順に従って2次元図面を描きプランを表現する．この場合，大きく分けて以下の3種の問題点がある．

1) 設計者個人の能力に大きく左右される．

2) 図面の解読を常に設計者が行わなければならず，2次元形状データが各フェーズごとに独立して存在してしまう．

3) 設計変更に弱い．

1)では，設計本来の物を作るという行為は，当然，設計者個人の能力そのものに依存する．しかし，頭の中に思い浮かべた形状を2次元の図面に書くという手順は，設計全般技術の蓄積であり，複雑な人間の判断が途中に入るとはいえ，企業ごと，製品ごとにほぼ一定しているはずである．

したがって，このような手順は標準化し，CAD システムに受け持たせることができる．しかし，通常の製図用 CAD システムは，2次元 CAD システムであり，本来の3次元形状を直接操作できない．このため，2次元入力ができる所までは設計者が行わなければならない．

2)では，設計情報の受け渡しが中心となるため，上流から下流へ一貫してデータが

流れることがない．すなわち，2 次元 CAD システムで作成するデータそのものは図面として出図されると，それ以後次の工程で有効に利用されることがない．

3)では，設計者が頭の中の 3 次元形状モデルを変更する場合，もともと 3 次元モデルがないため，設計者は 2 次元モデルの対応する部分を選び出し，変更する必要がある．

ここまでの関係を図 1 に示す．アイデアを最初に頭の中に思い浮かべてから，最終結果の図面が出てくるまでの全プロセスの内で上流の部分の役割を CAD システムが受け持てば持つほど，機能の大きい，人間にとってフレンドリーなシステムである．

したがって，従来より実用化されている 2 次元設計は，まだまだ人間から遠いシス

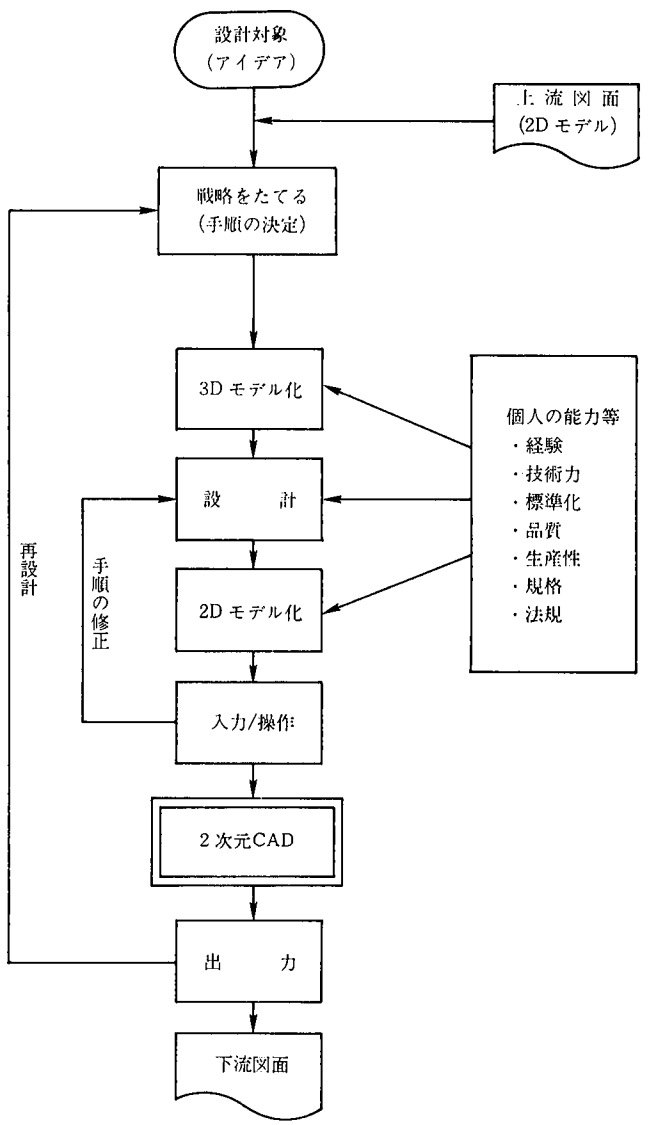


図 1　2 次元設計

Fig. 1　2D design system

テムと言える．これに対して，3次元編集設計では設計者が頭に思い浮かべた3次元形状そのままを3次元CADにより作成できる．このCADシステムに蓄積された3次元形状は，上流から下流まで一貫したデータとして操作可能であり，2次元設計のように常に図面に立ち戻って考えずに済み，図面→人間→形状→人間→図面—…→の無駄な繰り返しを減らすことができる．変更に対しては，そのまま3次元形状モデルを利用できる．

また，2次元設計同様，一定の手順に従って3次元CADにより作成された図面を設計段階のいつでも好きな時に出力ができ，図面の統一的な管理操作が可能である（図2参照）．これらを実現するのが，UNICAD/SURFACEの3次元編集設計用コマンド群である．

本稿では，3次元編集設計の概念，および汎用CAD/CAMシステムUNICAD/SURFACEで実現した機能を紹介する．

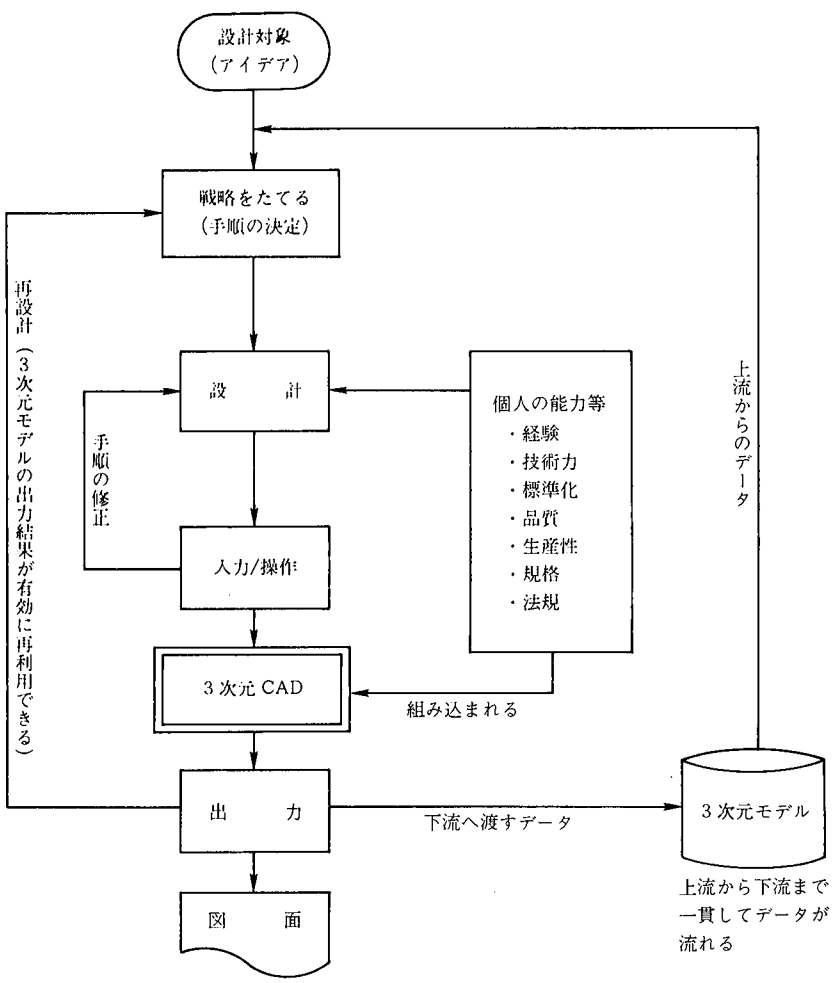


図2　3次元編集設計

Fig. 2　3D composite design system

2. 3 次元形状設計

設計者は，企業の設計活動の内で一定の役割を受け持っている．職務としては，部品の設計や，部品を組み合わせることによる設計が主となる．したがって，部品のようなあるまとまった〝かたまり〟を認識し，〝かたまり〟として操作できる必要がある．また，部品を組み合わせて作った集合部品も部品として操作できなくてはならない．なお，集合部品では，構成部品の仕様変更や取り替え等の構成部品に関する操作が必要である．このため，構成部品も部品としての認識および操作が必要である．すなわち，部品はどこに使用されていようと，その認識および操作ができ，部品が変更された場合にはそれが使用されているすべての場所で一斉に変更されなければならない．この部品に当たる〝かたまり〟をブロックと呼ぶ．ブロックは他のブロックに複数配置できる．ここで言う配置とは，位置関係を示すだけでデータを取り込むことではない．

したがって，元のブロックを修正すると配置されているすべてが変更される（図 3）．このブロックと配置の概念により，3 次元形状そのものの設計が可能となる（図 4）．

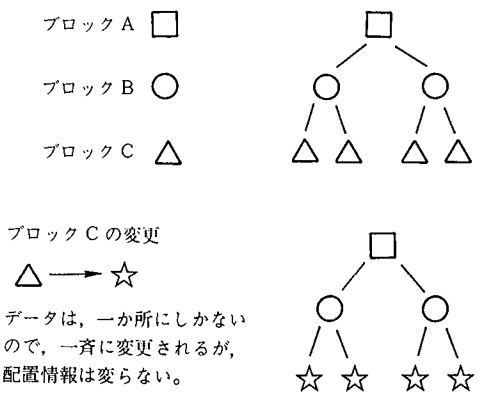


図 3　ブロックの変更

Fig. 3　Replacement of block

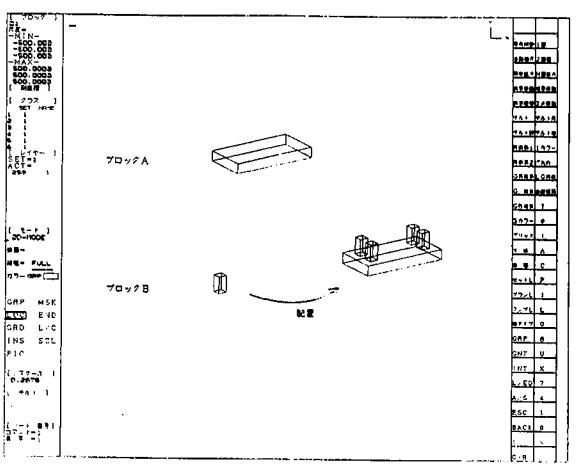


図 4　ブロックと配置

Fig. 4　Block and its uses

3. 3次元編集設計機能

3.1 ピクチャ作成

3.1.1 ピクチャ

図面を書くためには，正面図や右側面図などのように3次元形状をある面へ投影する必要がある．計算機にとって3次元形状を任意の視点から表示することは容易である．したがって，わざわざ3次元形状を投影して2次元形状を新たに創成しなくても，投影面さえ決めておけば，いつでも元の3次元形状を使って投影図を表示できる．この指定した投影図で表示される形状をピクチャと呼ぶ．ピクチャは，3次元形状および投影面（座標系および正投影面）によって定義される．

本システムでは，3次元形状をそのまま使用しているため，3次元形状を変更すると，そのままピクチャに反映されることに注意されたい．さらに付け加えれば，同一座標系のピクチャ達は互いに決まった位置関係で並ぶことが多い(図5)．そこで同一座標系のピクチャを同時に定義すれば，互いにピクチャ同志の位置関係を保ったまま一括操作することも可能である．

3.1.2 ピクチャ線種とピクチャ固有図形

同一形状を複数のピクチャで表示する場合，ピクチャごとに同一図形でも表示法を変える必要がある(図6参照のこと．図5と比較されたい)．たとえば，以下のような場合である．

1) 陰線処理等のように，表示されるピクチャによって線種が変わり，一本の線でも途中で何種類もの線種に分割される．
2) 概略図と詳細図等のように，同一図形が片方ではまったく表示されない．
3) 各々のピクチャに寸法や注記やハッチング等を追加することがある．

これらの処理に対応するため，ピクチャには以下の機能を持たせている．

1) ピクチャごとに，個々の図形に対し線種および表示しないという指定ができる

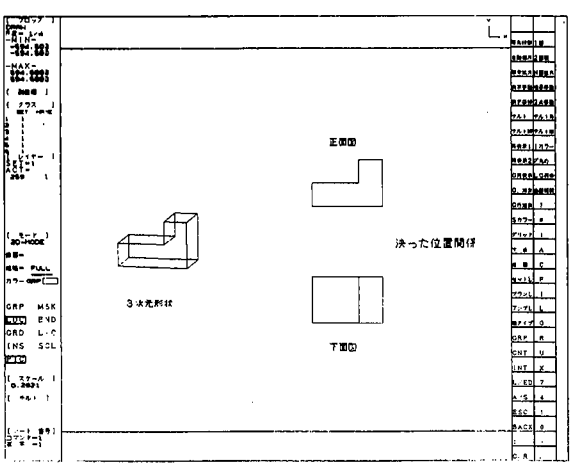


図5 ピクチャ関連図

Fig. 5 Relation of pictures

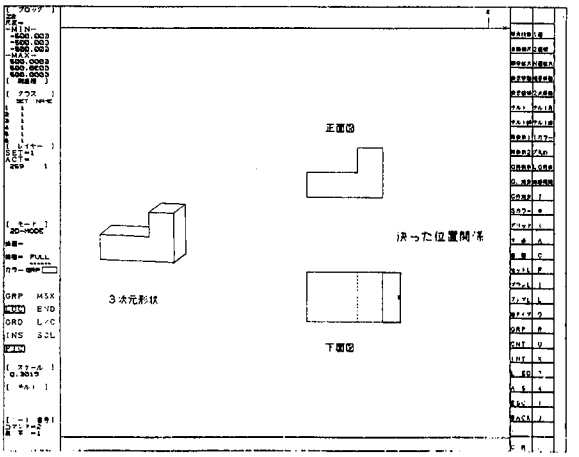


図6　ピクチャ
Fig. 6　Picture

(これを NO-DRAW の線種として扱う)．このピクチャごとの線種をピクチャ線種と呼ぶ．

2)　途中で何種類もの線種に分割する(部分線種と呼ぶ)．これは元の 3 次元形状を分割せずに行うこともできるし，分割することもできる．

3)　指示したピクチャだけにしか表示されない 2 次元図形（寸法を含む）を創成できる．この 2 次元図形は，元の 3 次元形状の存在するブロックに入れられるため 3 次元形状と一緒に扱う．

この 2 次元図形をピクチャ固有図形と呼ぶ．たとえば，自動寸法コマンドによってピクチャに寸法を創成した場合，寸法はこのピクチャにしか表示されないが，ストレッチ・コマンドにより 3 次元形状を変更すると，寸法も連動して変更される．

3.1.3　ドローイングと配置操作

紙に相当するブロックをドローイングと呼ぶ．このドローイングには，表題欄や注記等を記入するほか縮尺を指定してピクチャを配置する．配置操作は，図面（ドローイング）上のレイアウトを考える作業である．

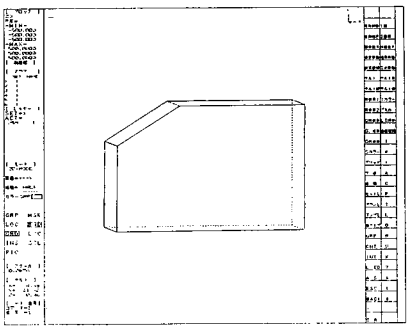
配置操作時にはピクチャの全図形が見えている必要はなく，当たりを調べるための外形線や位置関係を示すマークだけが表示されていた方が操作性が良い．

この目的のために UNICAD では代表図形と呼ばれ，図面とは無関係な図形をピクチャに定義する．この図形は，ピクチャ定義時または専用コマンドでいつでも定義できる．この図形は同一ピクチャの通常の図形と一緒に表示されることはなく，図面にも反映されない．代表図形で表示するか，通常図形で表示するかは，コマンドで指示する．

さて，ドローイングに配置したピクチャに対して，3.1.2 項に示したピクチャ固有図形やピクチャ線種を操作し図面を完成する．ここまでの流れを図 7 に示す．

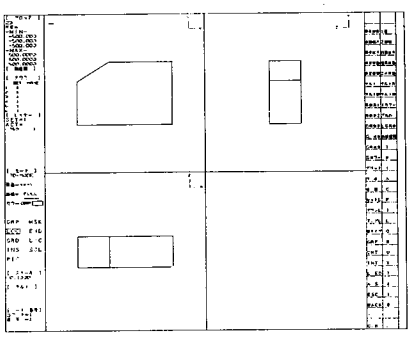
〈UNICAD コマンド〉

①モデル作成



・OPEN BLOCK 等一般コマンド

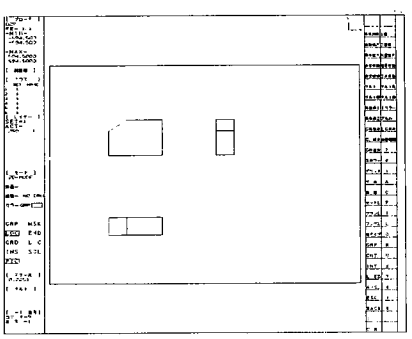
②ピクチャの定義



・CREATE PICTURE
(ピクチャの定義)

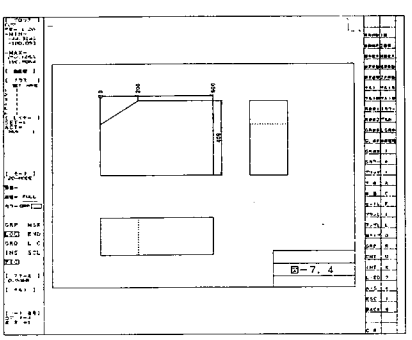
(図8 ピクチャ表参照)

③ドローイングへの配置



・CREATE DRAW
(ドローイングの創成)
・MODIFY DRAW
(ドローイングでの作業開始)
・USE PICTURE
(ピクチャの配置)
・CHG USE PICTURE
(ピクチャの再配置)

④図面編集



・MODIFY PICTURE
(ピクチャの変更開始)
・MODIFY ELEMENT
(部品の変更)
・CHG LINE TYPE-2
(線種の変更)
・一般寸法コマンド等

図7 設計の流れ

Fig. 7 Design flow

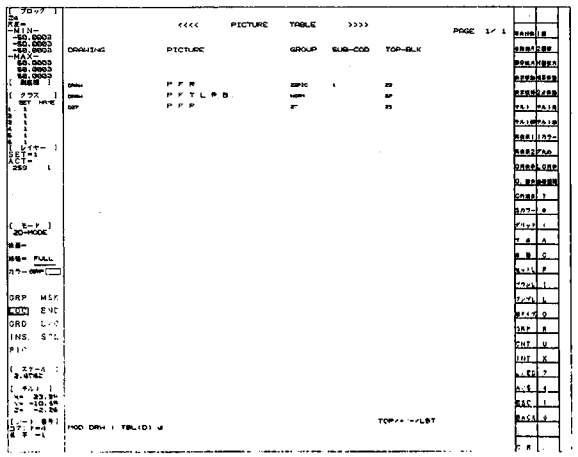


図 8　ピクチャ表
Fig. 8　Picture table

3.2 変 更 操 作

モデルの 3 次元形状に対して設計変更を行うとき，当然，図面に対しても反映する必要がある．このために UNICAD では，3 次元形状と 2 次元図面情報とが，一括して操作し変更できるようになっている．すなわち，前にも述べたように 3 次元形状と 2 次元図面情報(モデルと図面)の一体化が実現されている．このため，〝モデルの変更〟は，〝図面の変更〟と同じと考えてよい．

すでに，多くのモデルを CAD システム上に持っているシステムでは，変更はもちろん新規設計といえどもまったく新しく設計し直すことは少く，ほとんどの場合は，他の図面から流用することが可能である．したがって，以下では新規設計と設計変更等を区別せずに変更操作として述べる．

3.2.1 図形編集-1（切り張り細工）

ドローイングやピクチャ全体を複写することでも対処可能な場合もあるが，多くの場合，ピクチャの一部を変更しなければならない．たとえば，金型設計などでは，製品形状面以外は同一形状であることが多い．

この場合，ピクチャの複写後に元の製品面と新しい製品面とを入れ替えればよい．すなわち，不必要な部分を消し新規にその部分を作成し直すか，流用できる部分を複写するか，またはできるだけ似ている部分を複写した後それを修正する．これは，あたかも図面上で消しゴムで消して書き直したり，別図面の一部をコピーしてきて，のりで張る作業と似ている．UNICAD では，このために領域内を消すコマンド (CUT) や他から一部を複写してくるコマンド（DOCKING, COPYPICTURE INFO, REORGANIZE STRUCTURE 等）が用意されている．

3.2.2 図形編集-2（一括変更・創成）

形状変更で最も多いのは，ある一部の〝かたまり〟を移動・回転したり伸縮したりする場合である．この場合，対応する寸法も移動したり，値を変えたり，表示の仕方を変えたりする必要がある．ただ，既存の寸法だけでは不十分になったり，移動した

形状や寸法が他のものと重なったりすることがあるため，移動後に寸法を一括して作り直すことも起こり得る．UNICADでは，形状および寸法の一括変更のためにSTRETCH, ROTATE and TRANSコマンド，寸法の一括変更・創成のためにCALIBRATE MEASUREやMEASURE AUTOコマンド等が用意されている（図9）．

3.2.3 自動設計（変更操作の手順化）

ここでは，設計手順がきちんと手順化されている場合について述べる．UNICADには，UDL（UNICAD Design Language）と呼ばれる図形処理用プログラミング言語がある．UDLで書かれたプログラムでは，以下のことをUNICAD実行中にCRT画面から行える．

1) 算術演算・論理演算・図形演算
2) データの入出力
3) UNICADコマンドの実行

このUDLを使うと，設計手順を記述できるだけでなく，形状または寸法のためのパラメータを入力することによって目的とする形状や図面を作成できる．

3.2.1項と3.2.2項の機能は，既存の図面を変更するためのものであるが，本稿の機能はあらかじめ創成される形状の設計手順を視点を変えて記述しておき，パラメータにより目的とする形状や図面を作り出すためのものである．そして設計変更が起こったときは，既存の形状を直接修正するのではなく，UDLプログラムを実行し目的とする形状や図面を新しく作り出すわけである．この機能は，たとえば半径で形状の決まるフック・ピンや，長さによってバネの大きさなどが決まるスプリング・セット等に有効である．

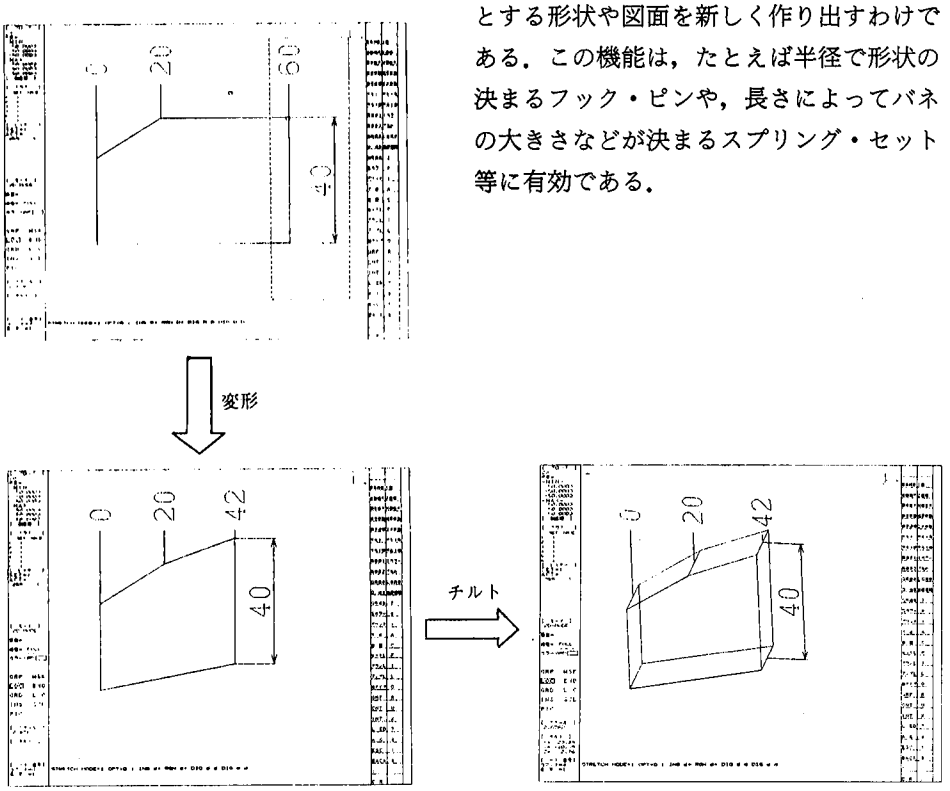
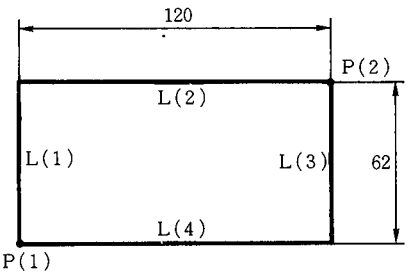


図9 ストレッチ・コマンドの使用例

Fig. 9 Example of STRETCH command usage

図 10 に，UDL プログラムの例を示す．

2 点の座標値を入力して，それらをコーナとする長方形を作り，水平方向と垂直方向の寸法線を入れる．(右図参照)




```
 PROG RECT ……マクロ・プログラム'RECT'の開始
*CREATE RECTANGLE ……*で始まる行はコメント行
 POINT P(2) ……点の宣言
 LINE  L(4) ……直線の宣言
*
 ##2D MODE ……2Dモードにする．
 ##1  VIEW (1) ……+X+Yの1面表示にする．
*
 'DIGITIZE TWO CORNER POINTS' DIG P2 ……2点P(1),P(2)をデジタイズ入力する．
*
 ENTER DIST ……長方形と寸法線の間隔を入力する．
*
 L(1)=P(1),PNT(P(1).X,P(2).Y,0)
 L(2)=PNT(P(1).X,P(2).Y,0),P(2)
 L(3)=P(2),PNT(P(2).X,P(1).Y,0)
 L(4)=PNT(P(2).X,P(1).Y,0),P(1)
                                   ……長方形を構成する四つの直線
                                       L(1)～L(4)を定義する．
 L.NUM=4
 OUTPUT L                          ……L(1)～L(4)を出力する．
*
 #MES HOR A=0:;
      L(2)/*S,L(2)/*E,PNT((P(1).X+P(2).X)/P(2).Y+DIST,0)
 #MES VER A=0:;                    }寸法線を出力する．
      L(3)/*S,L(3)/*E,PNT(P(2).X+DIST,(P(1).Y+P(2).Y)/2,0)
*
 STOP ……実行終了
 END  ……マクロ・プログラムの終了
```

図 10　UDL によるマクロ・プログラムの例

Fig. 10　Example of UDL program

3.3 図面作成（並行設計）

図面の作成は，一度形状ができあがってしまえば，形状の設計と違って一定の手順で行われるので手順を知っている者なら誰でもできる．したがって，三面図を一人で作成するよりも3人でそれぞれ一面図づつを担当した方が，トータルの作業量は同じでも，でき上がるまでの時間が速くなる．この目的のために開発されたのが，並行設計機能である．並行設計では，ジョブを元となるマスタジョブといくつかの作業用並行設計ジョブに分けて考える．マスタジョブから，更新用と参照用とに区別したブロックを並行設計用ジョブに移し編集設計作業を行う．作業が終了したら，マスタジョブへデータを返す．このようにして，並行して作成作業が行えるように配慮している．

この関係を図11に示す．

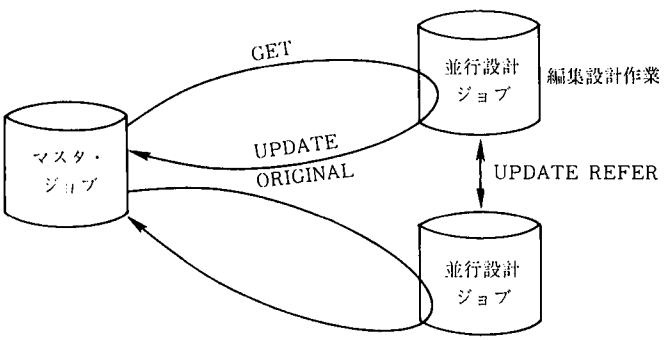


図11　並行設計

Fig. 11　Concurrent design

4. おわりに

ハードウェアの発達と低価格化と共に，CADシステムはソフトウェアも含めて発展しつづけている．それは，単なるレスポンスの改善や作業工程の一部をシステム化するだけにとどまらない．たとえば，従来は人間が頭の中で解析してきたことをCADシステム内でシミュレートしたり手助けしたり，あるいは独立して稼働していた既存のシステムを統合化したり多様である．なかでも顕著なのは，AIの導入やCAE化である．

本稿で紹介した3次元編集設計は，人間の思考に近づくという方向にそった，より役立つシステムと言える．2次元設計から3次元設計への移行は，従来，3次元モデルから2次元への変換を無意識に行っていた設計者にとっては当初は心理的な抵抗があるかも知れない．しかしながら，人間が図面から3次元を認識するのは訓練と経験の積み重ねである．3次元グラフィック・ディスプレイによる表示も画面単位で言えば2次元表示である．

ディスプレイを見ながらの入力操作も，基本的には2次元操作の組み合わせに過ぎず，2次元設計から3次元設計への移行は実際にCADシステムに触れてみれば，それほど異和感はないと考えられる．

本稿では，UNICAD/SURFACEにおける3次元編集設計機能の概要を紹介した．なお，主なUNICAD編集設計用コマンドを付表として添付した．筆者はこのような機

付表　主なUNICAD編集設計用コマンド一覧表

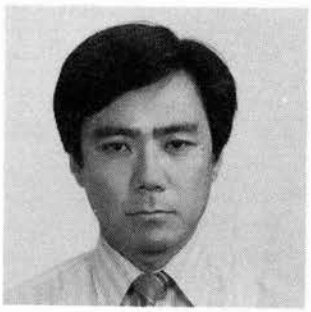
List of UNICAD commands associated with 3-D composite design

分類	コマンド	機　　能
ドローイング	CREATE DRAWING	ドローイングの定義 -図面サイズ・縦横・縮尺・ドローイング名等の指定 -配置予定のピクチャとあらかじめ関係付けておく.
	MODIFY DRAWING	ドローイング上で操作可能状態とする. -ドローイング名指示
	CLOSE DRAWING	ドローイング上での操作終了
ピクチャ	CREATE PICTURE	ピクチャの定義 -線種・代表図形・配置サイズ等の指定
	MODIFY PICTURE	ピクチャ上で操作可能状態とする.
	CLOSE PICTURE	ピクチャ上での操作終了 -自動的にドローイングでの操作が可能となる.
配置操作	USE PICTURE	ピクチャのドローイング上での配置 -縮尺・座標系・配置位置等の指示
	DEUSE PICTURE	ピクチャの配置の停止
	CHANGE USE PICTURE	ピクチャの配置関係の変更 -図形どうしの位置関係で定義可能
代表図形の操作	SELECT OUTLINE	代表図形の再定義
	DISPLAY PICTURE	代表図形を消して通常図形を表示
	ERASE PICTURE	通常図形を消して代表図形を表示
ピクチャ線種	CHANGE LINE TYPE-2	ピクチャ線種の変更 -CLASS・領域・重なり線・図形・グループ・ブロック・部分線種等の指示がある.
	HIDDEN LINE with PLANE	隠線処理
図形編集-1	CUT BLOCK	領域内の図形の削除
	DOCKING	下位ブロックの取り込み
	COPY PICTURE INFORMATION	ピクチャ固有図形とピクチャ固有線種の複写
	REORGANIZE STRUCTURE	下位構造の取り替え
図形編集-2	STRETCH	領域内の図形の伸縮または移動. このとき寸法値の再計算を行う.
	ROTATE AND TRANS	領域内の図形の回転. このとき寸法の再計算を行う.
	CALIBRATE MEASURE	ピクチャ編集後の寸法の再計算
	MEASURE AUTO	領域内の図形に対する並列寸法・累進寸法・直径/半径寸法の一括創成
自動設計	CREATE AUTO DESIGN INFORMATION	自動設計情報（UDLプログラム実行中の入力情報）の採取
	MODIFY AUTO DESIGN INFORMATION	自動設計情報の変更
	AUTO DESIGN	自動設計情報を入力し，UDLプログラムを実行
並行設計	LOGIN PD JOBUNIT INITIAL	並行設計ジョブの宣言
	GET PD BLOCK with PICTURE	マスタジョブよりドローイングとピクチャの並行設計ジョブへの取り込み
	UPDATE ORIGINAL BLOCK	並行設計ジョブより更新ブロックをマスタジョブへ取り込み，マスタジョブ内のブロックと置き換える.
	UPDATE REFERRED BLOCK	参照用ブロックの再取り込み
	DISPLAY PD INFORMATION	並行設計作業情報および取り込まれたブロックの状況の表示

能を実現している実用システムを寡聞にして知らない．このシステムは，すでにいくつかのユーザで使用されており，今後はユーザでの利用経験を踏まえて，3次元編集設計の普及に取り組んでゆきたいと思っている．

参考文献 ［1］ UNICAD解説書　設計・製図編，日本ユニバック　483845427-0.
［2］ UNICAD操作解説書　設計・製図編，日本ユニバック　483845424-5.
［3］ UNICAD解説書　UDL編，日本ユニバック　483845428-3.

執筆者紹介 新　田　光　義（Mitsuyoshi Nitta）
昭和24年生．50年学習院大学大学院自然科学研究科数学専攻修士課程修了．同年㈱日本ユニバック総合研究所入社．52年日本ユニバック㈱入社．55年よりUNICADの開発に従事．



論文 枠組壁工法住宅における構造の自動設計システム

An Automated Structural Design System for the Wood-frame Construction House

渡 辺 寛

要 約 本稿は，枠組壁工法住宅のCADシステムについて記述している．このシステムは，枠組壁構造の自動設計を行うことによって，構造データベースを生成する．その入力データは，既存の3次元設計支援システムによって生成された3次元家モデルである．そして，このデータベースを利用し積算システムや製図システムと連動させて，詳細見積書や構造図を出力する．

構造設計の自動化対象は次の通りである．

1) 入力の自動化（既存システムの出力データを利用し，新たな入力を不要とする）
2) 構造体の認識の自動化
3) 構造部材の選択の自動化

完全に自動的な構造設計システムを開発することは容易ではない．というのは枠組壁工法では，構造基準の標準化に問題があるからである．しかし，設計者が会話型操作によって多少補うことによって構造データベースを完成できる．

Abstract The paper describes the outline of a CAD system for a wood-frame construction house.

This system generates the structural database by designing the wood-frame structures automatically. Its input, the three-dimensional house design data is provided by the design support system. The system outputs the detail estimates and structural drawings by co-operating with the cost calculating system, the figuring system, and the drawing system.

The system realizes automation in:

1) input of data (with no interactive operations)
2) recognition of the structural model
3) selection of the housing materials.

It is not easy to develop a completely automated structural design system unless the structural standard is established in the wood-frame constructions. However, the usable structural database is expected to be completed in a relatively short time by the interactive operations of designers.

1. はじめに

枠組壁工法住宅は，工法の公開（昭和49年）以来，その建設戸数は伸長の一途を辿っている．住宅生産者は，処理業務の増大に伴い施工面においては，合理化を目的に小屋組のトラス化や枠組のパネル化等を工法の標準化と併行して行ってきた．また，設計面においては，顕在化する熟練技術者不足の中で，〝生産性と品質の向上〟，〝処理時間の短縮〟，〝省力・合理化〟等を目的に業務のシステム化を進めてきた．筆者は，日本ユニバックのSEサービスの一貫として住宅産業関連のいくつかの企業のCADシステムの開発に係わってきた．本稿では，その中から枠組壁工法住宅メーカの最大

手である A 社におけるシステム事例を紹介する．

本システムは，先に開発された 3 次元設計関連業務支援システム（設計サブシステム，図 1）をもとに，とくに「機械的な作業が 8 ～ 9 割」を占める枠組壁工法住宅の構造設計（以下，構造設計と呼ぶ）に焦点を当て，昭和 59 年に開発に着手したものである（図 1）．開発の目的は，〝構造設計の生産性・精度の向上〟および，後続の木拾いや原価積算等の〝合理化〟にあった．

構造設計は，構造的なキー・プランとして基本計画時に描かれる〝耐力壁線区画〟と，指定された各部位に対する〝架構方式〟とを主要設計条件として行われる．構造部位（床・壁・小屋組）の認識に始まり，架構部分やその構造特性を導き，構造部材（以下，部材と呼ぶ）を選択し，つぎに継手・仕口・納まりの決定へと進められる．

本システムでは，以上の認識にもとづき，次の事項について自動化を推進することとした．

1) 入力の自動化（設計サブシステムの設計情報を入力として用い，構造処理専用の入力は行わない．）
2) 構造体の認識の自動化
3) 構造部材の選択・決定の自動化
4) 構造部材の創成・取合の自動化（実長計算を含む）

本稿では，枠組壁工法の構造(体)の自動設計について，その手法を紹介するとともに実用化に向けての可能性を考察する．

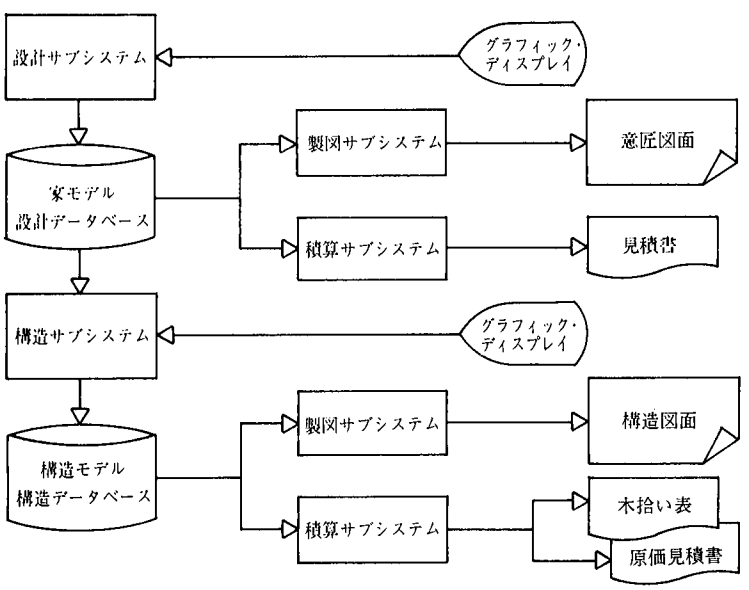


図 1 システム構成図

Fig. 1 System overview

2. 構造設計と自動化の考え方

構造設計作業は，標準化された構造基準と仕様にもとづいて行われる．整理・蓄積された技術者の構造知識によって進められる〝構造体の認識〟と，標準化された部材

リスト，スパン表や詳細図集を参照して行われる〝部材の架構〟に大別（図２）される．

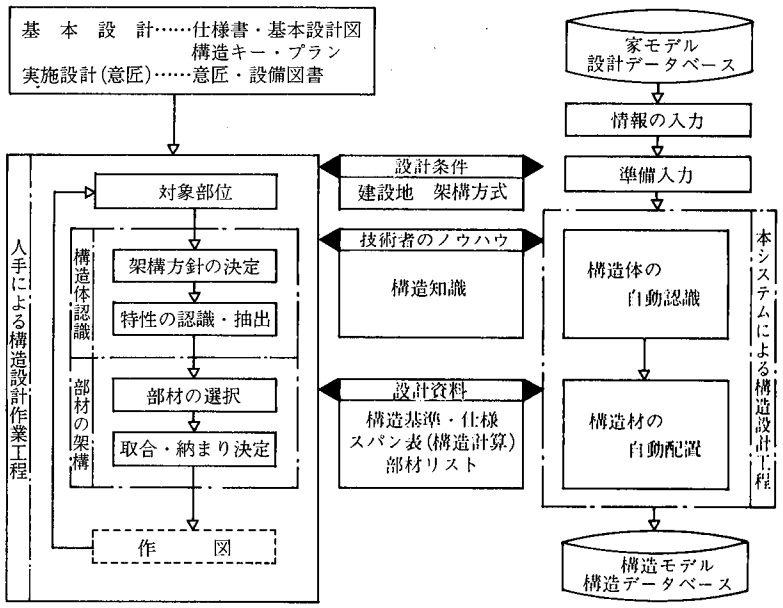


図 2　構造設計の自動化の概念

Fig. 2　Automated structural design concept

自動構造設計システムは，構造設計を行うために必要な〝情報の入力〟および，部材の架構に必要な情報（部材の創成場所，選択・決定与件等）を導き出す〝構造体の認識〟が自動化されたシステムを指す．また，このシステムにおいては，入力情報の精度と品質，構造知識の豊富さによって自動化の水準が決定される．

本システムでは，先に開発したCADシステムの設計情報の中から，構造設計のための入力として

1)　耐力壁線区画（設計者の設計意図の伝達・反映）

2)　設計条件（建設地・各部の架構方式等の外部条件）

を利用した．

また，構造知識は，熟練技術者の持つノウハウを〝構造記述言語〟によって，順次，蓄積することとした．

3.　構造記述言語

枠組壁工法の構造体は，層的な構造を持つ(図３)．各部位層は，架構部材の種類単位に，領域・線・点の属性として表現される．

技術者の持つ構造設計知識は，〝属性定義〟という形式を用い表現し蓄積する．また，これによって，従来のシステムの仕様化過程（設計者と仕様作成者による諸作業）における理解の相違や，記述内容の曖昧さや矛盾を払拭するとともに，設計者の意図を明確に記述し忠実に再現できる．

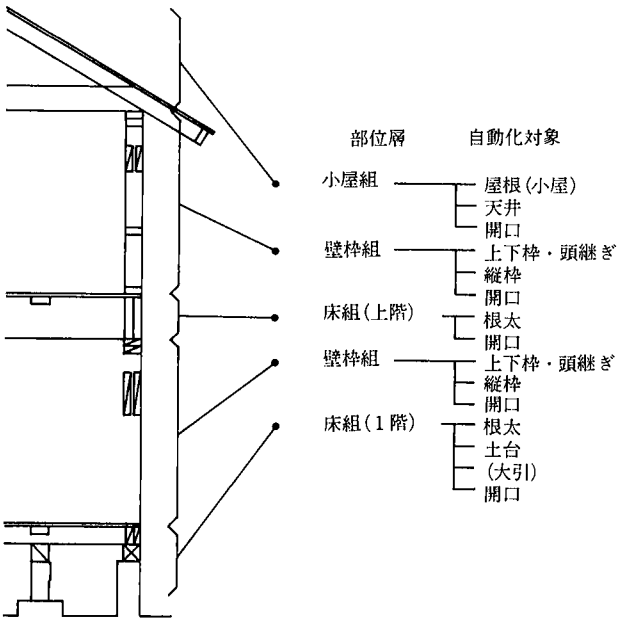


図 3　枠組壁工法の構造体

Fig. 3　Structure of a wood-frame construction

さて，これらのことを目的として，50種余りの語彙（〝使用語〟と呼ぶ）を用い，論理式で構造設計知識を表現することを試みた．一般形を次に示す．ここで，* は 0〜n 個の項の繰り返しを示す．

構成要素〔属性〕＝ 使用語（構成要素〔属性〕〔，構成要素〔属性〕〕*）

構成要素は，〝構成領域〟・〝構成線〟・〝構成点〟の総称であり，ある属性を持つ構成要素を明示する．

属性は，ある特定の性質を示し，その性質を持った構成領域・構成線・構成点を使用語によって関係付ける．

使用語は述語名であり，その作用は n 項関係を満たす属性によって新たな属性を定義（創成）する機能を持つ（図 4）．

構成：(A＝構成(A0, A1))	機能：A0 とA1 から構成される Aを定義する．たとえば A が構成線の場合，A0 と A1 は構成点
共有：(A＝共有(A0, A1))	機能：A0 が A1 と共有関係にあるとき，共有する部分を A とする．
連結：(A＝連結(A0))	機能：互いに平行関係にあり端点が一致する複数の構成線 A0(つまり，同一直線上にある線分群) を，最小の座標値を持つものから順に繰り返し連結し一つの線分としたものを A として定義する．

図 4　使用語の例

Fig. 4　Structure definition language

構造的なキー・プランである耐力壁区画から創られる基本情報（基本構造領域）は，この言語によって次のように記述される．ここで，$^+$ は 1〜n 個の項の繰り返しを示す．

構成領域〔基本構造領域〕＝構成（設計情報＊耐力壁線$^+$）
＝分割（構成領域〔基本構造領域〕，設計情報＊支持壁線）

4. システムの概要

本システムは，家モデル（入力設計情報）を基に小屋組・壁枠組・床組の3部位について，構造体の自動認識と構造材の自動配置を行い構造モデルを構築する．

また，製図と積算の両システムとの連動によって，成果物としての図書を出力する（図1）．

4.1 構造体の自動認識

1) 準備入力……処理対象の建物ごとに異なる建築条件，つまり建設地域の指定や小屋組・床組の架構方式（トラス架構・梁木架構や在来束立床組等）などの選択は，準備入力として処理に先立って行われる（図5）．

1. 計画建物タイプ	01	一般住宅
2. 材質・種類	01	S2-S—P—F
3. 基礎天端高さ	01	一般地域標準
4. 吊天井仕様	01	吊天井仕様　有
5. まぐさ高さ	01	1階＊20，2階＊20
6. 1階床高さ	01	土台＋大引＋204
7. 基準値	01	タイプ：フリー＊18
8. TBLバージョン名	1	自動創成＊全
9. 葺き降し継手	2	葺き降し継手＊梁木仕様
10. 使用構造材	8	首都圏
11. 使用構造材材種	2	STANDARD
12. 構造仕様	8	首都圏

図 5　準備入力項目

Fig. 4　Data preparation items

2) 構造体の自動認識……入力設計情報（家モデル）は図6に示すデータ構造であり，3次元情報として外形・屋根・屋内・屋外・基礎に分類（大分類）され，さらに各階別に分類（各部）されデータベースとして格納されている．

構造体の自動認識は，家モデルから抽出された情報にもとづいて，蓄積された構造知識によって進められる（図7）．

屋根を構成する屋根部位面より，軒先・けらば・棟・稜等の屋根構成線を導き出し，切妻や寄棟等の屋根型の自動認識を行う．そして，根太・土台・大引・束等の部材を自動創成するための最下階の床組における構成領域・構成線・構成点を創り出す．

たとえば，壁枠組における盲壁や開口壁の識別は次のように行われる．

これらは，家モデルより得られる部屋壁面情報と，内・外建具情報から自動認識される．部屋壁面は，部屋壁部位として壁線に置換される．建具情報は，開口

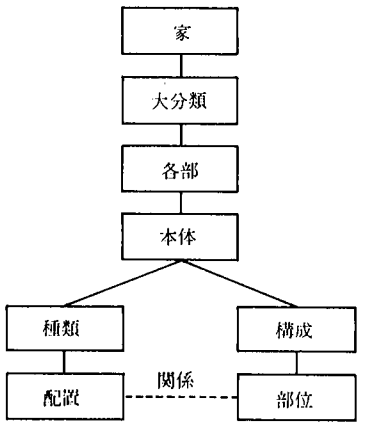


図6　3次元家モデルのデータ構造

Fig. 6　Data stucture of the 3 dimensional housing model

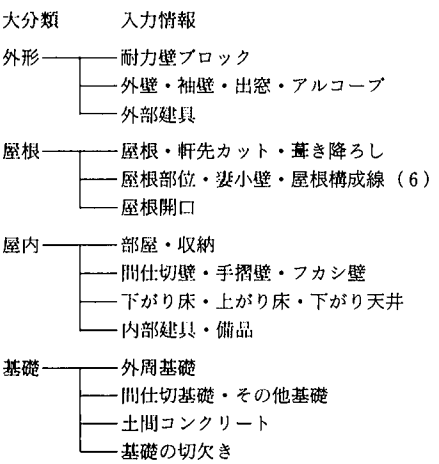


図 7　家モデルからの入力情報

Fig. 7　Input data from the housing model

情報（躯体開口 ROW，ROH）を持ち，部屋壁面と〝穴〟の関係を持っており，穴部位として認識され，開口壁線に置換される．その壁線を開口壁線で分割することにより，盲壁と開口壁を自動認識する．同時に，壁線や開口壁線の交点を壁交点として自動認識する．

構造体の荷重の伝達に従って，小屋組・壁枠組・床組の各部位に，また上階から下階へと各階別に自動認識された結果，構成要素としての属性の集合体である〝構造情報〟を生成する．

根太・梁木等を対象とする構成領域，梁・まぐさ・棟木等を対象とする構成線，竪枠・金物等を対象とする構成点等がそれである（図8）．

3)　構造情報……構造情報は，各階別の各部位（基本・小屋・壁・床）に分類され，構成要素としての属性から形成される．

構造情報は，構造体の自動認識後に抽象的に構築された構造モデルである（図

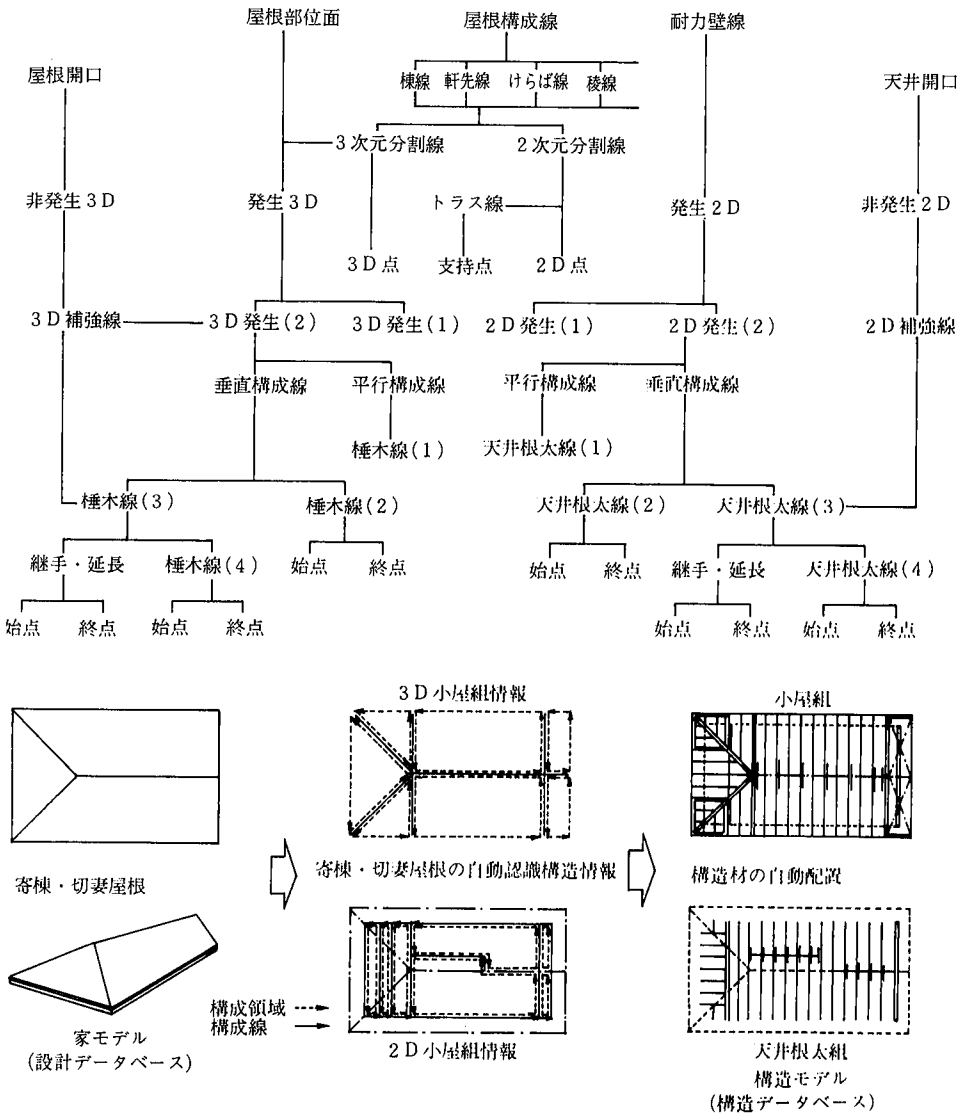


図 8　構造体の自動認識（小屋組の例）

Fig. 8　Automated recognition of a wood-frame structure

9）．小屋組・壁枠組は3次元，床組は2次元とし，抽象的（たとえば，構成領域の持つ架構方向により，根太・大引・束等を同一構成領域を用いて創成する）情報とした．また，構成要素としての属性に応じた配置情報・部材の決定要素や取合情報等の固有属性を同時に生成する．

一般的な壁交差部について，構造情報の事例を示す．壁交差部（交点）は，壁枠組における属性〝壁交点〟として認識される．構成点〔壁交点〕は，次の固有属性を持つ（図10）．

固有属性の〝接続タイプ〟と〝壁区分〟は，その交点を構成する竪枠の組み合わせを決定する要素であり，壁区分の壁優劣情報は，頭継ぎ材や上・下枠材の取

合（端部における部材の勝負）を決定する要素である．また，〝位置〟は決定された部材の配置場所を指定する情報である．

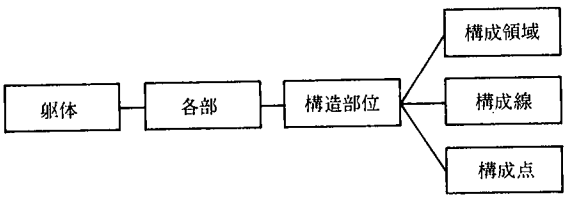


図 9　構造情報のデータ構造

Fig. 9　Data structure of a wood-frame structure information

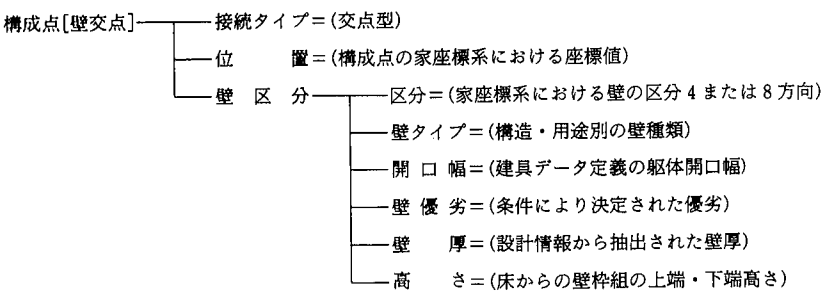


図 10　構造情報例（構成点〔壁交点〕）

Fig. 10　An example of the wood-frame structure information

4.2　構造材の自動配置

必要構造材は，発生部材を指示する発生部材情報によって，制御・自動配置される．また，創成された部材は実体としての構造構成要素となり，構造モデルとしての構造データベースを創り上げる．

1)　発生部材情報……発生部材情報は，構造体の架構方式ごとに発生の可能性のある構造材種類をあらかじめ定義した情報である．

発生部材情報は，図 11 の情報より構成される．

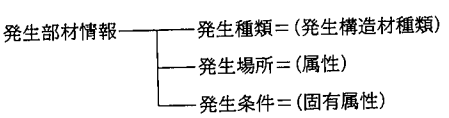


図 11　発生部材情報データ型

Fig. 11　Data types of the generated parts information

本システムでは構造基準・仕様の改訂・更新や新たな架構方式に容易に対応できるように，外部的なデータとして扱うこととした．

2)　データとしての部材……自動配置される部材は，それぞれの部材の持つ固有の性質や，成果物としての表現形態等が定義される．

部材は，図12で示される情報により構成される．

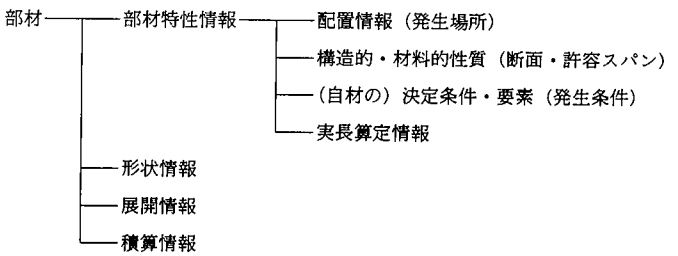


図12　部材データ型

Fig. 12　Data types of parts

部材は，後続のシステムから多目的に利用される情報である．そこで，データ主導型を目指す本システムではデータの独立性を高めるため，部材のシステムからの分離を図った．

3)　部材の自動配置……発生部材情報として指示された構造材種類ごとに生成された構造情報によって，合致する属性（発生場所）を検索し，抽出した固有属性（発生条件）を部材の決定要素・配置条件として導く．また，発生条件に適合した一つの部材が構造材種類の中から自動決定される．

決定された部材は，構造情報にもとづいて，部材の持つ配置情報に従って，図13のような約20種類の配置コマンド（たとえば，根太は〝構成領域に対して，複数の線として発生〟する．また，竪枠は〝構成点に対して，単一の点として発生〟する）のもとで，取合・操作および自動配置され，構造データベースに格納される．

自動配置の事例として図14に壁交点の例を示す．

構成点〔壁交点〕は，構造情報として，上述（図10）の情報を持つ．壁区分情報の壁厚より，204型式の一般壁であるか，206型式のバルーン・フレームであるかが識別される．また，壁タイプと開口幅より，対象交点における壁の接合状態を得る．壁の接合状態が壁区分の組み合わせとして，接続タイプ別に定義された〝部材取合テーブル〟により，壁の接合状態を指示する分類コードを抽出する．その分類コードと壁区分別に開口幅が定義された〝部材決定テーブル〟により，一つの壁交点部材が自動決定される．

決定された部材は，家モデル座標系におけるX－Yの各軸と対応した壁区分情報と，部材の持つ壁区分情報とが一致するように，部材の取合操作（回転・対称）を行う．家モデルから得られた位置を配置場所として自動配置される（図14）．また，壁区分の持つ高さ情報をもとに，同時に部材の実長が求められる．

4)　構造モデル（構造データベース）……創成された構造部材は，構造モデルの構成要素として図15の構造を持つデータベースに格納される．

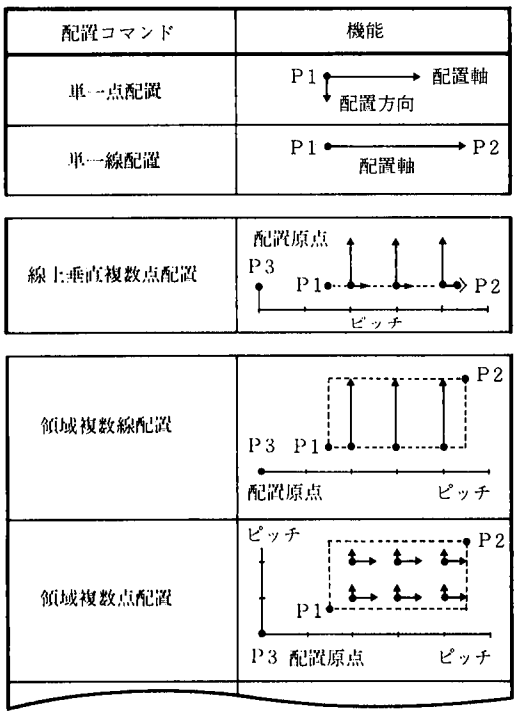


図13　自動配置コマンド例

Fig. 13　Automated layout commands

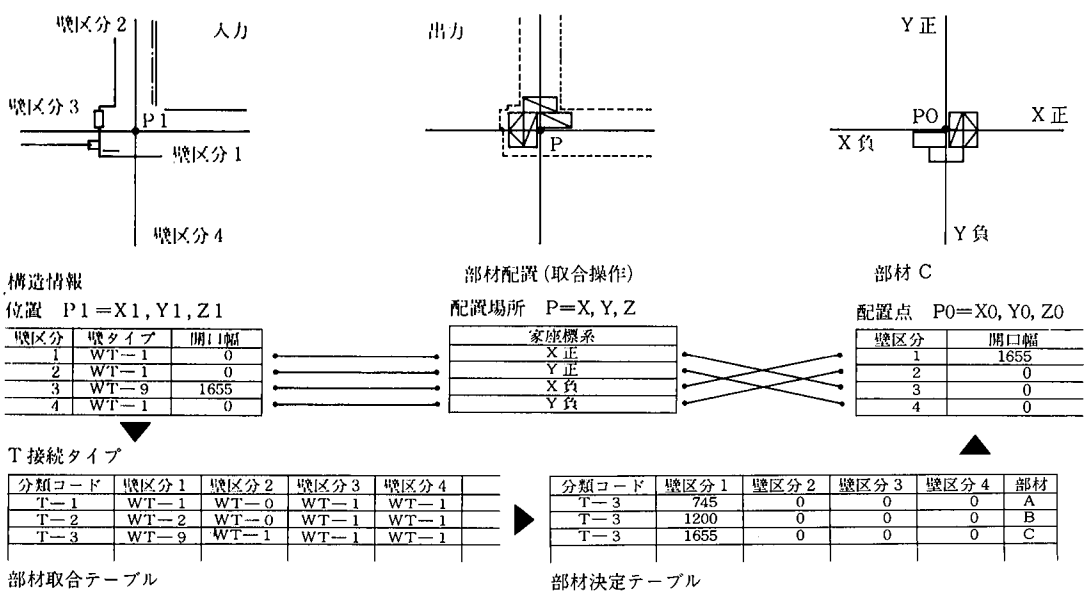


図14　自動配置例（〔壁交点〕）

Fig. 14　An example of the automated layout

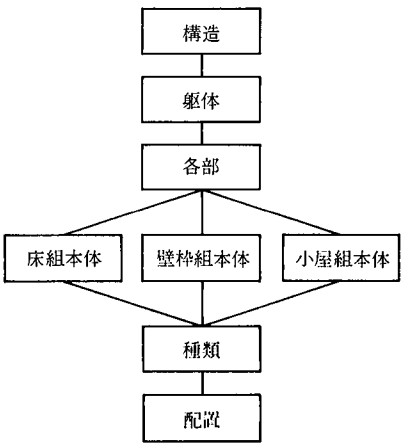


図15　構造モデルのデータ構造
Fig. 15　Data structure of the structural model

各階別（各部）と部位別（本体）に分類され，構造材種類ごとに部材は格納される（表1）．従来の単一目的（作図のみのため）のデータベースとは異なり，多目的（作図・木拾い・原価積算等）な利用が期待できるデータベースを実現した．また，後続システムのために，その必要情報を抽出・加工する共通的なプログラムを追加することにより，既存システム（製図・積算システム）の利用を可能とした（図1）．

5. システムの特徴

1)　システムの形態……本システムは，大型汎用コンピュータを用いた自動構造設計システムである．

2)　処理対象範囲……2階建ての専用住宅と共同住宅をその対象範囲としている．

3)　自動発生部材の種類……本システムでは基礎を除き，図3に示す各部位の木部および，アンカーボルト，換気孔等を自動発生部材の対象としている．木部においては，木拾いの対象となる補足材（根絡みや合板受材等）も主要材と同様の発生対象部材としている．

4)　部材の断面決定……本システムでは，構造部材種類ごとに，必要な断面決定要素を生成し，その要素に合致した部材を選択・決定することによって行われる．

たとえば単材の梁木の場合，その位置における屋根勾配・架構スパンを，その決定要素としている．また，補強材としての壁開口部のまぐさ材や，床組・天井根太組の梁の場合には，自材に影響する負担荷重を荷重負担長さに置換して，その決定要素としている（図16）．これらの決定要素と合致する部材は必要断面を満足する．

5)　経済性・施工性の追求……枠組壁工法で使用される木材のうちで，断面形式で204，206，210等，定尺長さで8F～22Fのものが輸入される定尺材である．本システムでは，定尺材の有効活用を目的として，部材の経済性を図っている．たと

表1　自動発生部材例

Table 1　Automatically generated parts

①壁枠組

発　生　部　材	自動設計	会話操作	単一創成・配置	グループ創成・配置
頭継ぎ	○	○	単一線(長さ2点)	—
吊天頭継ぎ	○	○	単一線(長さ2点)	—
上　枠	○	○	単一線(長さ2点)	—
下　枠	○	○	単一線(長さ2点)	—
一般竪枠	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
バルーン・フレーム竪枠	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
手摺壁竪枠	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
出窓竪枠	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
浴室廻り竪枠	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
開口上部竪枠	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
開口下部竪枠	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
バルーン・フレーム合板受	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
筋違い		○	単一線(長さ2点)	
備品・設備下地		○	単一点＋単一線	
梁・まぐさ受金物	○	○	単一点(補1点)	

②床　組

必　要　部　材	自動設計	会話操作	単一創成・配置	グループ創成・配置
床根太	○	○	単一線(長さ2点)	領域複数線(対角2点 自由入力＋1点)
端根太側根太	○	○	単一線(長さ2点)	—
側根太	○	○	単一線(長さ2点)	—
端根太転び止め	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
転び止め	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
添え木	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
梁	○	○	単一点(補1点)	—
	○	○	単一線(補1点)	
金物　JH	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
BH	○	○	単一点(補1点)	—
吊木受		○	単一線(長さ2点)	領域複数線(対角2点 自由入力＋1点)

③小屋組

発生部材	自動設計	会話操作	単一創成・配置	グループ創成・配置
トラス　T	○	○	単一線(方向2点)	線上複数点(長さ2点＋1点)
GT_1	○	○	単一線(方向2点)	—
GT_2	○	○	単一線(方向2点)	—
GT_3	○	○	単一線(方向2点)	—
GET	○	○	単一線(方向2点)	—
G	○	○	単一線(方向2点)	—
TH	○	○	単一線(方向2点)	線上複数点(長さ2点＋1点)
[illegible]	○	○	単一線(方向2点)	
棟木板	○	○	[illegible]	
棟　木	○	○	単一線(長さ2点)	領域複数線(対角2点 自由入力＋1点)
鼻隠し棰木	○	○	単一線(長さ2点)	—
棰木継	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
隅棰木	○	○	単一線(長さ2点)	—
寄棟軒先補強棰木	○	○	単一点(補1点)	—
母　屋		○	単一線(長さ2点)	—
棰木転び止め	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
棰木受	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
雲筋違い	○	○	[illegible]	
振れ止め	○	○	単一線(長さ2点)	
金　物　JH	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
BH	○	○	単一点(補1点)	—
TW-30	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
TS	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)
S-65	○	○	単一点(補1点)	線上複数点(長さ2点＋1点)

えば，土台材（定尺長さ3m/4m）の継手位置や，根太（定尺長さ8F～16F）の継手位置を経済的な適正化処理を行い自動決定している．

また，現場でのフレーミング工程において，施工手順の矛盾があってはならない対象については施工性を考慮している．たとえば，壁枠組における枠組の建て起こし順は，それぞれの壁交点における壁優劣をもとに，枠組交差部の勝負を自動決定している．

6. 実用化と今後の課題

6.1 実用化に向けて

本システムは，現在，処理業務フロー等のシミュレーションを含む試行を行っている．なお，使用しているハードウェアの構成を図17に示す．

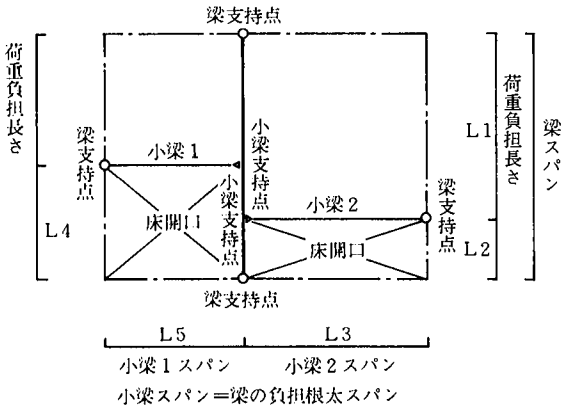


図16　断面決定要素例（小梁を持った床組）

Fig. 16　Example of elements to determine a section

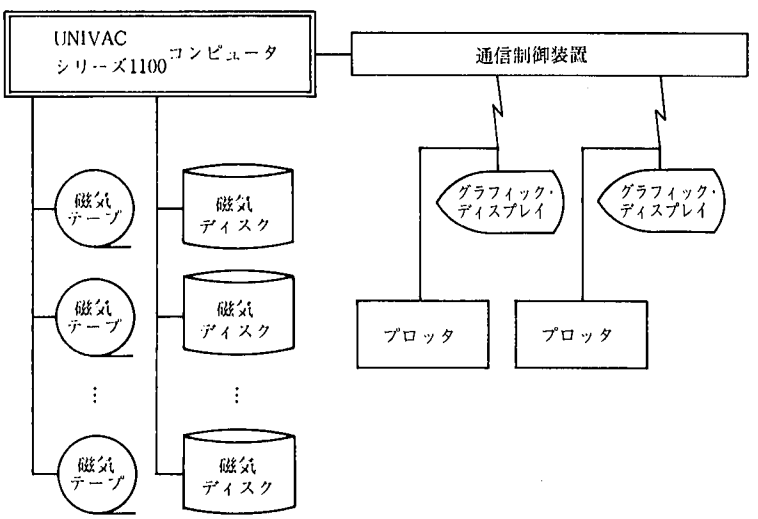


図17　ハードウェアの構成

Fig. 17　Hardware configuration

1)　システムの処理時間……処理時間の例として，規模40坪程度の専用住宅の場合の処理時間（出力例）を示す．自動構造設計処理では処理時間約18分であり，発生部材数の増加に伴い処理時間が延長する傾向となっている．

成果物としての階別の基礎・土台・床組・天井根太組・小屋の各伏図および，竪枠割付図を自動生成する製図処理では，処理時間約30分であり，スケール1/100，1/50と大差なく処理される．

2)　処理業務の流れ……現在の集中処理形態における業務は，自動設計の後，本システムの会話操作機能（図1）を用いて，不足材の追加，創成材の変更・修正等の補完を行い構造モデルを完成する流れとなっている（図18）．

3)　出力例（構造図）……本システムの出力例を図19に示す．

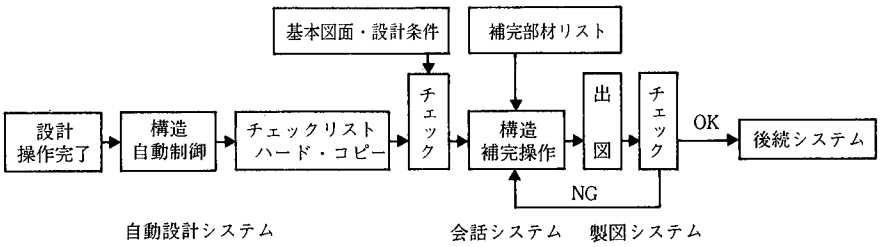


図18　処理業務の流れ
Fig. 18　Process flow

6.2 今後の課題

1)　本システムの構造知識は技術者の持つノウハウである．現在，まだ蓄積されていない構造知識は，未標準化部分，たとえば設備配管を逃げるための床組根太架構方向の変更や，壁枠組の筋違いの方向決定等の知識である．なお，これらについては，会話操作による補完作業に委ねている．未標準化部分の一般化および構造知識の充実を図る一方で，現状での自動化範囲の見極めを行う必要があると考えられる．

2)　また，構造知識を，構造基準と仕様の変更に追従させなければならない．自動認識の根幹である構造知識は，システムの耐久性と保持性の面から知識データベースとして，システムに組み込む必要がある．

3)　現在の製図システムによって自動生成される構造図には，必要な詳細寸法や注記等の不足部分がある．そこで，今後は完成度を高めるために，寸法と注記の自動生成や，重なりチェックの自動化等の製図システムの改良を行う予定である．

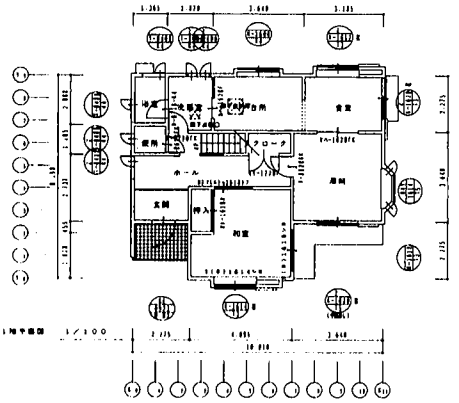
7. おわりに

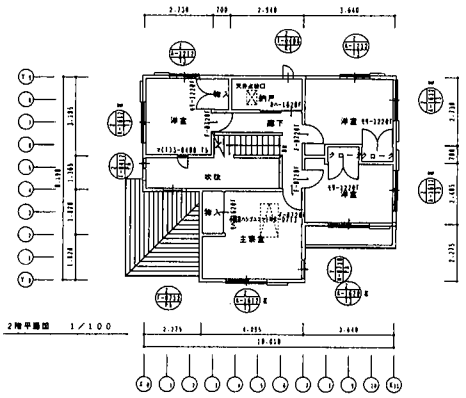
本システムの試行の結果，処理時間の短縮と熟練技術者の負担の軽減が確認され，本システムの実用化への目処がついたところである．

今後は，導入に向けて処理業務の流れの確立とシステムの完成度の向上に務めるとともに，設計関連業務への人工知能利用の可能性を研究・追求してゆきたいと考えている．

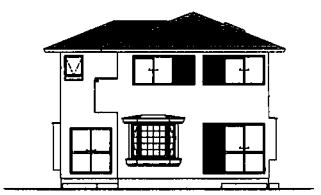
また，部材や施工について標準化が進んでいる枠組壁工法であるが，工法としての制限緩和に伴ない在来工法化が進行している．論理式として仕様記述を行う中で構造基準と仕様上の不明瞭な点（床組束立方式における部材の納まりや取合，小屋組の在来工法化等）が抽出され解決されたが，それらは今後の構造基準と仕様の改訂の際の基礎資料として利用される予定である．

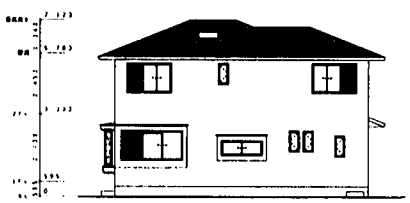
最後に本稿は，日本建築学会・電子計算機利用委員会による第9回電子計算機利用シンポジウム1987にて発表された内容に多少手を加えたものであり，御協力頂いた関係者各位に深く感謝致します．

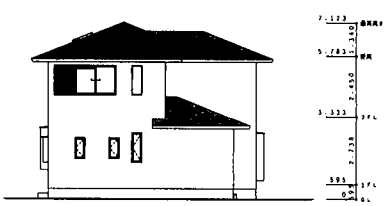




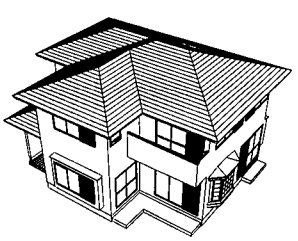
平面図





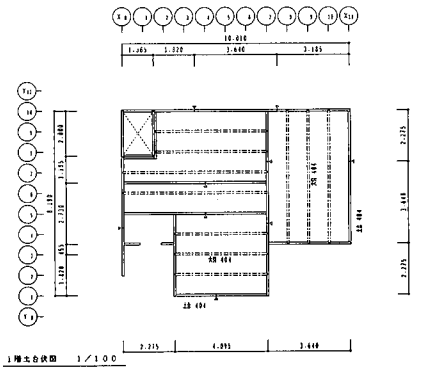


立面図

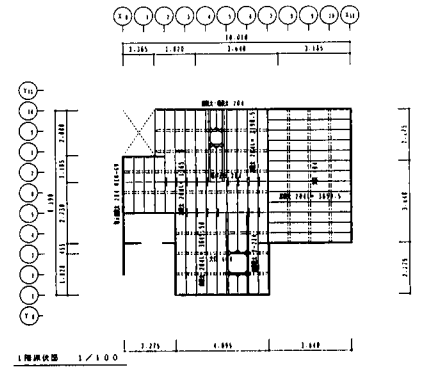


外観透視図

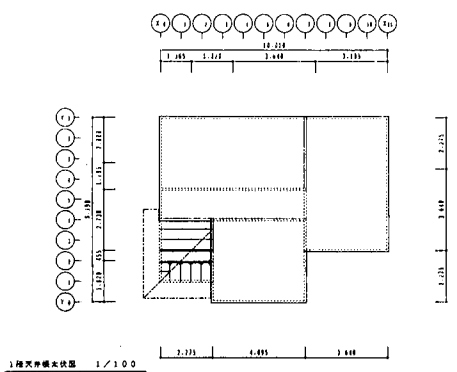
図19　構造自動設計システムの出力例（その1）



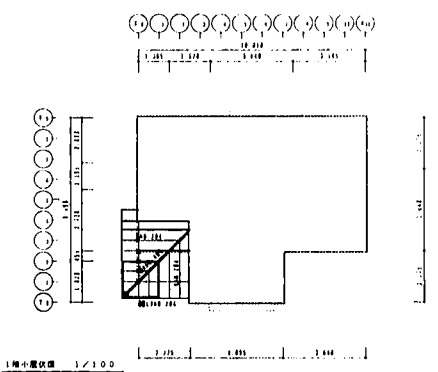
土台伏図



床伏図

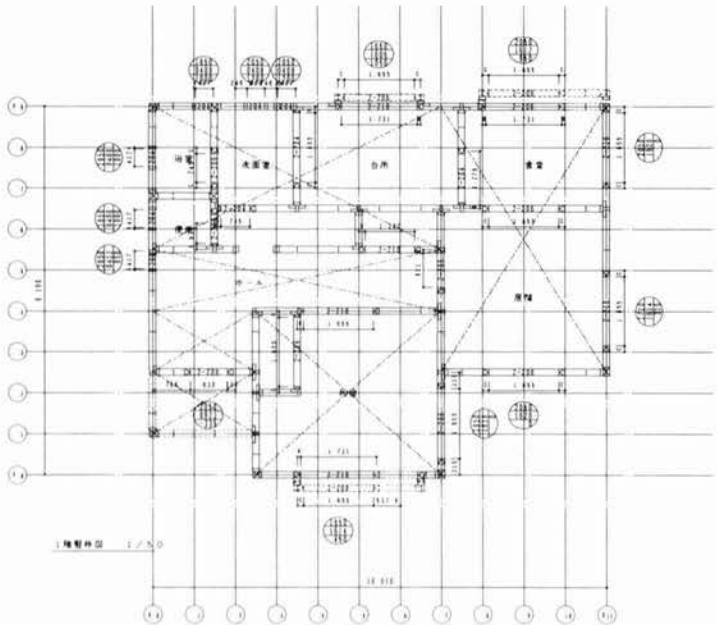


天井根太伏図



小屋伏図

図19　構造自動設計システムの出力例（その2）



堅枠図

図 19　構造自動設計システムの出力例（その 3）

Fig. 19　Sample of output

参考文献　［1］　柳生孝昭，CAD のための Data Metamodel，情報処理学会「グラフィックスと CAD」シンポジウム論文集，1983 年．
［2］　篠田博水，3 次元家モデルに基づく住宅一貫システム，日本ユニバック（株）技報，1985 年 2 月，第 8 号．
［3］　木村一嘉，図形情報システム―その設計方法と事例―，マグロウヒル好学社，1978．
［4］　長尾　真，言語工学（人工知能シリーズ 2），昭晃堂，1984．
［5］　大須賀節雄，CAD/CAM と人工知能，PIXEL，1983 年，5/6 月号～1987 年 2 月号．

執筆者紹介　渡　辺　　寛（Hiroshi Watanabe）
昭和 22 年生．44 年早大理工学部物理学科卒業，同年日本ユニバック（株）入社．各種グラフィック・ソフトウェアおよび CAD システムの開発を経て，現在，ハウジング CAD システム開発に従事．

論文

ソフトウェア設計法の味見

A Sampler of Software Design Methods

峰 尾 欽 二

要 約 ソフトウェア設計法の有用性を示し普及を計るために，小さな課題に対して二つの設計法を味見した．一つは，表示的意味記述の枠組みにより仕様からCobolコードを作成する方法であり，もう一つはCSPの仕様からModula-2コードを作成する方法である．味見の結果，これらの設計法が産業界においても十分実用に耐えるという感触を得た．

Abstract To demonstrate the applicability and usefulness of software design methods and to promote them, we "tasted" them in applying to a small exercise. One of the design methods we tasted is one that generates Cobol code from a specification in the frame of denotational semantics. The other is the method to generate Modula-2 code from a specification in CSP. As a result of the tasting, we got the feeling that above two methods are applicable and also effective for the applications in the business fields.

1. はじめに

企業でのソフトウェア開発に活用するために，ソフトウェア工学の味見を行ってきた．その第1報は「仕様記述の味見」[1]である．これはその第2報である．

ソフトウェア工学を開発の現場で活用するためには，ある種の条件を整える必要がある．たとえば，

1） 現場担当者の教育訓練，

2） 集団作業を行うための，技法に関連するいろいろな標準の設定，

3） 現行技法との共存方法と移行手順，

4） 技法の推進調整のための機関の設立，

などである．また，ソフトウェア工学の活用を推進するためには，具体的な課題に対して試用し，その成果を明示する必要がある．このような理由から，ソフトウェア工学全般にわたっての方法を試用し，その味見を数人の仲間とともに行っているわけである．本稿は設計方法についてである．

設計に関して，設計対象・設計方法・設計支援環境などが議論できる．設計対象についての議論は，ある範囲の対象を深く考察し，それによって設計そのものをある程度自動化しようというものである．ここでは，対象についての仕様から自然にその実現がえられることを目指す．このような設計法は対象主導型といっていい．LCP[2]やJSP[3]は課題対象を一定の考え方で表現し，それから一定の手続きによってその実現を計る点でこの例である．

設計方法についての議論は，対象を特定しないでソフトウェア作成上に現れる固有の問題のみを扱う．課題に対する計算方法は別途に検討するものとして，たとえば課題や算法の分割・階層化を議論する，抽象データ型とその段階的詳細化などがこの例である．

分割統治や抽象の概念を中心に，抽象データ型とその段階的詳細化(HISP[4], IOTA[5], HDM[6], VDM[7])，後件から述語変換系によって最弱前件を求めプログラムを導出する Dijkstra[8]，Gries[9]の方法，その並行プロセス版である相互通信型逐次プロセス(CSP[10]) などがこの例である．

設計支援環境は，設計法を支援する環境のことで，その設計法の成否に大きく影響するものであるがここでは触れない．

対象主導型の設計法は，それを修得すればその種の課題に対して直ちに活用できる利点をもっている．解法主導型の設計法は，一般的な方法であるだけに，対象に対する優れた見識を必要とし，多くの職業プログラマにとって直ちに活用できる方法とはいえない．そこで小さな課題を設定し，勉強の中から浮上した二つの方法を試用してみることにした．設計法1は，プログラム言語の仕様記述を見習って，表示的意味論[11]の枠組で仕様を作成し，それから自然にプログラムが作成できるというものである．これは対象主導型設計法になっている．設計法2は，CSP を取上げ JSD[12]流に考えたものである．これは解法主導型の方法に対象主導型の考えを埋め込んだ，いわば混合型のものといえる．

2. 例　　題

例題として書店の売掛管理を選んだ．つぎのような問題である．

『ある書店では，特定の顧客に対して書籍の掛売を行っている．販売係は，書籍とともに納品伝票を顧客に渡し，すぐにその写し（売上伝票）を会計係に送る．

会計係では，月末になるとその月の売上伝票を集計し売掛残高報告書を作成すると同時に，各客先に請求書を送ることになっている．

顧客は，現金や銀行振込などによって請求額を支払うが，一度に全額を支払うとは限らない．入金額は財務係から随時入金伝票として会計係に回送されている．会計係の仕事をプログラムしたい．

売上伝票：顧客名；〔書籍名；単価；冊数；金額〕
　　（上記の繰り返し）
入金伝票：顧客名；入金金額
売掛残高報告書：前月末総未済残高；当月総売上高；当月総入金高
　　；当月末総未済残高
請求書：顧客名；前月末請求額；当月買上額；当月支払額；当月請求額』

3. 設 計 法 1

課題は，それぞれ顧客名で整列されている売上伝票ファイル，入金伝票ファイル，顧客ファイルを入力して，請求書ファイル，新顧客ファイル，報告書ファイルを出力するものとする．なお，売上伝票の書籍名，単価，冊数，金額は単純化するために無視し，顧客名と合計金額のみとした．

3.1 表示的意味論による仕様記述

プログラム言語の表示的意味論では，まず構文の領域と意味の領域を帰納的な方程式系で定義する．つぎに意味関数を定義する．意味関数とは，構文要素に意味領域の

要素を割り当てる関数で，この関数値が構文要素の意味となる．すなわち，Syn が構文領域を，Sem が意味領域であるとすると，意味関数は

μ : Syn → Sem

である．構文要素 $s \in$ Syn の意味を $\mu[s]$ で表示する．

課題の場合，入力ファイルは下記の構文で示したような言語とみてよい．その構文領域は，客名などのように基礎となる領域，売上レコードなどのように領域構成子で作られる領域，および入力ファイルなどのように構文の再帰的な方程式系により決まる領域からできている．意味領域は，出力ファイル（請求書ファイル，新顧客ファイル，報告書ファイル）の状態と計算のための作業領域の状態からできている．入力構文要素ごとの状態の変化を，次のような意味関数で帰納的に定義して課題の意味を与える．

μ : Syn → [状態→状態]

例題の仕様を書くと次のようになる．

1）構　文　領　域

客名 ∈ 客名①= 文字列
金額 ∈ 金額 = 整数
残高 ∈ 残高 = 整数
売上レコード ∈ 売上レコード = 客名 × 金額
入金レコード ∈ 入金レコード = 客名 × 金額
顧客レコード ∈ 顧客レコード = 客名 × 残高
入力ファイル ∈ 入力ファイル
客データ ∈ 客データ
取引データ ∈ 取引データ
取引 ∈ 取引

2）構　　文

入力ファイル ::= 客データ　入力ファイル
客データ ::= 顧客レコード　取引データ
取引データ ::= 取引　取引データ
取引 ::= 売上レコード　入金レコード

3）意　味　領　域

{状態}

$\sigma \in \Sigma$ = [{invf, newcf, rep}+{grdsum, cussum}
→請求書*②+新顧客+報告書+総計+客計]
ただし σ(invf) ∈ 請求書*
σ(newc) ∈ 新顧客*
σ(rep) ∈ 報告書
σ(grdsum) ∈ 総計
σ(cussum) ∈ 客計

{出力レコード}

請求書 = 客名 × 前残 × 買上 × 支払 × 請求

新顧客 ＝ 客名 × 残高
報告書 ＝ 前残 × 総売 × 総入 × 新残

{合計領域}

総計 ＝ 前残 × 総売 × 総入
客計 ＝ 客名 × 残 × 売 × 入

{項目}

客名 ∈ 客名 ＝ 文字列
前残 ∈ 前残 ＝ 整数
買上 ∈ 買上 ＝ 整数
支払 ∈ 支払 ＝ 整数
請求 ∈ 請求 ＝ 整数
残高 ∈ 残高 ＝ 整数
総売 ∈ 総売 ＝ 整数
総入 ∈ 総入 ＝ 整数
新残 ∈ 新残 ＝ 整数
残 ∈ 残 ＝ 整数
売 ∈ 売 ＝ 整数
入 ∈ 入 ＝ 整数

4） 意 味 関 数

billing：入力ファイル→[Σ→Σ]

billing【入力ファイル】③(σ)＝auxbilling【入力ファイル】(initialstate)；④makereport

ここで

initialstate＝{invf→〈〉, newcf→〈〉, rep→〈〉, grdsum→〈0, 0, 0〉}⑤

auxbilling：入力ファイル→[Σ→Σ]

auxbilling【客データ　入力ファイル】(σ)＝cdataf【客データ】(σ)；
auxbilling【入力ファイル】

makereport：Σ→Σ

makereport(σ)＝σ[rep/reportediting(σ(grdsum))]⑥

ここで

reportediting：総計→報告書

reportediting(s)＝〈s・前残，s・総売，s・総入，
(s・前残＋s・総売－s・総入)〉

cdataf：客データ→[Σ→Σ]

cdataf【客データ】(σ)＝auxcdataf【客データ】(σ)；grdsumupdate；invoicing
；newc

ここで

auxcdataf：客データ→[Σ→Σ]

auxcdataf【顧客レコード　取引】(σ)＝custf【顧客レコード】(σ)；tloop【取
引】

grdsumupdate：Σ → Σ

grdsumupdate(σ)=σ[grdsum/〈σ(grdsum)・前残＋σ(cussum)・残,
σ(grdsum)・総売＋σ(cussum)・売,
σ(grdsum)・総入＋σ(cussum)・入〉]

invoicing：$\Sigma \rightarrow \Sigma$

invoicing(σ)=σ[invf/σ(invf)^〈σ(cussum)・客名，σ(cussum)・残，
σ(cussum)・売，σ(cussum)・入，
(σ(cussum)・残＋σ(cussum)・売－
σ(cussum)・入)〉]

newc：$\Sigma \rightarrow \Sigma$

newc(σ)=σ[newc/σ(newc)^〈σ(cussum)・客名，(σ(cussum)・残＋
σ(cussum)・売－σ(cussum)・入)〉]

custf：顧客レコード→[$\Sigma \rightarrow \Sigma$]

custf【顧客レコード】(σ)=σ[cussum/〈顧客レコード・客名，顧客レコード・前残，
0,0〉]

tloop：取引データ→[$\Sigma \rightarrow \Sigma$]

tloop【取引　取引データ】(σ)=tranf【取引】(σ)；tloop【取引データ】

tranf：取引→[$\Sigma \rightarrow \Sigma$]

tranf【取引】(σ)=

(isSales⑦(取引)→

σ[cussum/〈σ(cussum)・客名，σ(cussum)・残，
(σ(cussum)・売＋売上レコード・金額)，σ(cussum)・入〉]

isReceipt (取引)→

σ[cussum/〈σ(cussum)・客名，σ(cussum)・残，
σ(cussum)・売，(σ(cussum)・入＋入金レコード・金額)〉])

なお，上の表記について若干補足する．

① 下線付きの名前は領域名を，下線のない名前は領域変数を表す．

② D^* は D 上の列を表す．すなわち，

$D^*=\{\langle d_1, d_2, \cdots, d_n\rangle \mid 0<n,\ d_i \in D\}+\{\langle\rangle\}$

$d \in D$ かつ $d^*=\langle d_1, d_2, \cdots, d_n\rangle \in D^*$ のとき $d^* \hat{} d=\langle d_1, d_2, \cdots, d_n, d\rangle$

③ 意味関数 f を構文単位 s に適用することをとくに f【s】と書く．

④ 関数 f, g に対し f；g は関数合成 g∘f と同じである．

⑤ 数え上げによる写像の定義

⑥ $\sigma[s/v]$ は状態 σ の変化を示し，s の値だけを v に変える．すなわち，$\sigma(s)=v$ となる．

⑦ 入力レコードが入金レコードか，売上レコードかを知ることができるブール関数があると仮定した．

3.2 実　　現

COBOL での実現例を図1で示す．それぞれの意味関数を手続きとし，帰納的に定義した関数は perform 文による反復構造に変換する．ただし，論理入力ファイルからレコードを得る入力手続きを用意した．

```
 identification division.
 program-id. billing.
*
 environment division.
 configuration section.
 source-computer. univac-1100.
 object-computer. univac-1100.
 input-output section.
 file-control.
      select sales assign to disc.
      select receipts assign to disc.
      select customers assign to disc.
      select invoices assign to disc.
      select balreport assign to disc.
      select newcustomers assign to disc.
 data division.
 file section.
 fd   sales label record standard.
 01   srec pic x(18).
*
 fd   receipts label record standard.
 01   rcprec pic x(18).
*
 fd   customers label record standard.
 01   crec pic x(18).
*
 fd   invoices label record standard.
 01   invrec.
      02 cnames pic x(10).
      02 bal pic 9(8).
      02 sal pic 9(8).
      02 receipt pic 9(8).
      02 newbal pic 9(8).
*
 fd   balreport label record standard.
 01   breprec.
      02 bal pic 9(12).
      02 sal pic 9(12).
      02 receipt pic 9(12).
      02 newbal pic 9(12).
*
 fd   newcustomers label record standard.
 01   ncusrec.
      02 cnames pic x(10).
      02 newbal pic 9(8).
*
 working-storage section.
 01   grdsum.
      02 bal pic 9(12).
      02 sal pic 9(12).
      02 receipt pic 9(12).
 01   cussum.
      02 cnames pic x(10).
      02 bal pic 9(8).
      02 sal pic 9(8).
      02 receipt pic 9(8).
*
 01   inputfile.
      02 inputeof pic x value space.
         88 is-inputeof value 'e'.
      02 inputrec.
         03 rectype pic x.
            88 is-sales value 's'.
            88 is-receipts value 'r'.
            88 is-customer value 'c'.
         03 inputitems.
            04 cnames pic x(10).
            04 amounts pic 9(8).
            04 balances pic 9(8).
      02 mincode pic x(10).
 01   inputrecordarea.
      02 salesrec.
```

```
          03 cnames pic x(10).
          03 amounts pic 9(8).
       02 receiptrec.
          03 cnames pic x(10).
          03 amounts pic 9(8).
       02 cusrec.
          03 cnames pic x(10).
          03 balances pic 9(8).
 procedure division.
 billing.
      perform openinf.
      perform openoutf.
      perform initialstate.
      perform readinputfile.
      perform auxbilling.
      perform makereport.
      perform closeinf.
      perform closeoutf.
      stop run.
 initialstate.
      move 0 to bal of grdsum,
                sal of grdsum,
                receipt of grdsum.
 auxbilling.
      perform cdataf until is-inputeof.
 makereport.
      move corr grdsum to breprec.
      compute newbal of breprec = bal of grdsum
                                  + sal of grdsum
                                  - receipt of grdsum.
      write breprec.
 cdataf.
      perform auxcdataf.
      perform grdsumupdate.
      perform invoicing.
      perform newc.
 auxcdataf.
      perform custf.
      perform readinputfile.
      perform tloop.
 grdsumupdate.
      add bal of cussum to bal of grdsum.
      add sal of cussum to sal of grdsum.
      add receipt of cussum to receipt of grdsum.
 invoicing.
      move corr cussum to invrec.
      compute newbal of invrec = bal of cussum
                                 + sal of cussum
                                 - receipt of cussum.
      write invrec.
 newc.
      move cnames of cussum to cnames of ncusrec.
      compute newbal of ncusrec = bal of cussum
                                  + sal of cussum
                                  - receipt of cussum.
      write ncusrec.
 custf.

      move cnames of inputitems to cnames of cussum.
      move balances of inputitems to bal of cussum.
      move 0 to sal of cussum,
                receipt of cussum.
 tloop.
      perform tranf until not (is-sales or is-receipts)
                           or is-inputeof.
      perform readinputfile.
 tranf.
      if       is-sales
               add amounts of inputrec to sal of cussum
      else if is-receipts
               add amounts of inputrec to receipt of cussum.
```

```
openinf.
    open input sales, receipts, customers.
    perform readsales.
    perform readreceipts.
    perform readcustomers.
closeinf.
    close sales,
          receipts,
          customers.
openoutf.
    open output invoices,
                balreport,
                newcustomers.
closeoutf.
    close invoices,
          balreport,
          newcustomers.
readinputfile.
    if not (cnames of cusrec = high-value and
            cnames of salesrec = high-value and
            cnames of receiptrec = high-value)
        perform getminrec
    else

        move 'e' to inputeof.
getminrec.
    move cnames of cusrec to mincode.
    if mincode > cnames of salesrec
        move cnames of salesrec to mincode.
    if mincode > cnames of receiptrec
        move cnames of receiptrec to mincode.
    if      mincode = cnames of cusrec
              move 'c' to rectype
              move corr cusrec to inputitems
              perform readcustomers
    else if mincode = cnames of salesrec
              move 's' to rectype
              move corr salesrec to inputitems
              perform readsales
    else if mincode = cnames of receiptrec
              move 'r' to rectype
              move corr receiptrec to inputitems
              perform readreceipts.
readcustomers.
    read customers into cusrec at end
     move high-value to cnames of cusrec.
readsales.
    read sales into salesrec at end
     move high-value to cnames of salesrec.
readreceipts.
    read receipts into receiptrec at end
     move high-value to cnames of receiptrec.
```

図1 COBOLによる実現

Fig. 1 Implementation in COBOL

4. 設 計 法 2

4.1 CSPの概要

CSPの中心概念はプロセスと事象である．課題対象をその一部であれ全体であれプロセスと考える．たとえば，プロセスとして自動販売機を考える場合は，事象は，「硬貨を入れる」，「ボタンを押す」，「缶ビールが出る」などとなる．形式的には，プロセスと事象は無定義術語で，以下に記すような演算とそれが導く性質をもつものと理解

する．プロセス P の事象全体を P の事象集合 (alphabet) といい，αP と書く．プロセスが生まれた最初の状態からある時点までに生起した事象の系列を考え，これを足跡 (trace) という．プロセス P の足跡の全体を足跡集合 (traces) といい，τP と書く．

ここでプロセス P をその事象集合 αP との足跡集合 τP との組；

$P=(\alpha P, \tau P)$

として定義する．ここで，$\tau P \subseteq \alpha P^*$は，次の性質を満たすものとする（$\alpha P^*$ は αP の有限列の全体）．

① $\langle\rangle \in \tau P$　(ただし，$\langle\rangle$ は αP^* の空列)

② $s\hat{}t \in \tau P$ ならば $s \in \tau P$

(ただし，$\hat{}$ は αP^* の結合演算)

①は，P の生誕したままの状態，つまり何の事象も起こっていない時を示し，②はある時点までの足跡が $s\hat{}t$ であれば，それ以前のある時点で，それまでの足跡がちょうど s になっていることを示している．

プロセスの全体 P の上でいくつかの演算を定義する．

1) $\mathrm{STOP}=(A, \{\langle\rangle\})$

被演算子をもたない定数演算である．プロセス STOP は，事象集合として A をもっているが，何の事象も生起しない（死に詰まり）ものである．

2) $(a \to P)=(\alpha P \cup \{a\}, \{\langle\rangle\} \cup \{\langle a\rangle\hat{}t \mid t \in \tau P\})$

プロセス $(a \to P)$ は，まず事象 a が生起して，それ以降はちょうどプロセス P と同様に挙動するプロセスである（a それから P）．

3) $(a \to P \mid b \to Q)=(\alpha P \cup \alpha Q \cup \{a,b\}, \{\langle\rangle\} \cup \{\langle a\rangle\hat{}s \mid s \in \tau P\} \cup \{\langle b\rangle\hat{}t \mid t \in \tau Q\})$

(ただし，$a \neq b$)

$a \neq b$ として，a が生起すればその後は P と同様，b が生起すればその後は Q と同様に挙動するプロセスである（a それから P，または b それから Q）．

4) $(x : B \to P(x))=((\bigcup_{x \in B} \alpha P(x)) \cup B, \{\langle\rangle\} \cup \{\langle x\rangle\hat{}s \mid x \in B \wedge s \in \tau P(x)\})$

5) $P/s=(\alpha P, \{t \mid s\hat{}t \in \tau P\})$

プロセス P の足跡がちょうど s であったとき，それ以降の P の挙動をするプロセスである（s のあとの P）．

6) $P \| Q=(\alpha P \cup \alpha Q, \{s \mid s \in (\alpha P \cup \alpha Q)^* \wedge (s \uparrow \alpha P) \in \tau P \wedge (s \uparrow \alpha Q) \in \tau Q\})$

プロセス P とプロセス Q が並行に動いているようなプロセスのことである（P は Q と並行）．

$s \uparrow \alpha P$ は足跡 s を事象集合 αP に制限したもの（つまり，足跡 s を構成する事象のうち αP に属するものだけを抽出したもの）を表している．

7) $(\mu X : A.F(X))=(A, \bigcup_{n \geq 0} \tau(F^n(\mathrm{STOP}))$

事象集合 A をもつプロセスで，プロセス変数 X に対する方程式 $X=F(X)$ の一意解となるもの．なお，F は以上に挙げた演算（μ を含む）によって構成された X 上の関係である．

4.2 CSP による仕様記述

本節では，図2の書店の請求書発行システム INVSYSTEM の仕様を CSP の枠組で書いた例を説明する．

詳細に至るまで確定した CSP のための特定の記述言語は，今のところ，まだないようである．このため，足りないところは適宜，補っている．

```
〔1〕 const noc = 20   (*  The maximum number of customers  *)
     Id = {1, ..., noc}   (*  The set of customer Id numbers  *)
〔2〕 INVSYSTEM = Customers ‖ DataBase ‖ Invoice ‖ Report
〔3〕 Customers = ‖ (i ∈ Id) i : C   (*  C simulates a customer  *)
〔3.1〕 C = (openacc → opensv → Cl)
〔3.2〕 Cl = (buybook ? u → price ! u → Cl
           | makepay ? v → pay ! v → Cl
           | closeacc → closesv → C)
〔4〕 DataBase = ‖ (i ∈ Id) i : SV   (*SV  is the state vector of a customer  *)
〔4.1〕 SV = (opensv → V(0, 0, 0) | mkinv → notmkinv → SV)
〔4.2〕 V (balance, sale, receipt)
          = (price ? u → V (balance, sale+u, receipt)
            | pay ? v → V (balance, sale, receipt+v)
            | mkinv → sv ! ⟨balance, sale, receipt⟩
                    → V (balance+sale−receipt, 0, 0)
            | closesv → leftl. mkinv ! ⟨balance, sale, receipt⟩ → SV)
〔5〕 Invoice = (start → Inv (0) (0, 0, 0) )
〔5.1〕 i ∈ Id ◇ Inv (i) (tb, ts, tr)
          = (i. mkinv →
              (i. notmkinv → Inv (i+1) (tb, ts, tr)
               | i. sv ? x → left. mkinv ! ⟨i, x⟩
                           →Inv (i+1) (tb+π1 (x), ts+π2 (x), tr+π3 (x) ) ) )
〔5. 2〕 i = noc+1 ◇ Inv (i) (tb, ts, tr)
          = (left. mkrep ! ⟨tb, ts, tr⟩ → Invoice)
〔6〕 Report = (left. mkinv ? x → right ! ε1 (x)
             | left. mkrep ? x → right ! ε2 (x)
             | (i ∈ Id) i. leftl. mkinv ? x → right ! ε1 (x) )
```

なお，ここで，π_1, π_2, π_3 は次の射影

$\pi_i(\langle e_1, e_2, e_3\rangle) = e_i (i = 1, 2, 3)$

のことで，ε_1 は invoice の編集を，ε_2 は report の編集をする関数を意味する．編集の詳細については省略する．

さて，以下では，上の仕様を順を追って説明する．なお，次の［ ］付きの番号は上の仕様での番号に対応している．

［1］ noc は定数で，このシステムで扱える最大顧客数を示す．Id は顧客の ID 番号の集合を表す．

［2］ INVSYSTEM は四つのサブシステムからなり，それらが同時並行して動く．

そのうち，Invoice と Report は，それぞれ一つのプロセスからなっているが，Customers と DataBase の二つは，そうではない．Customers では，noc 個の C プロセスが，DataBase では，やはり noc 個の SV プロセスが，それぞれ同時並行して動く（［3］，［4］を参照）．

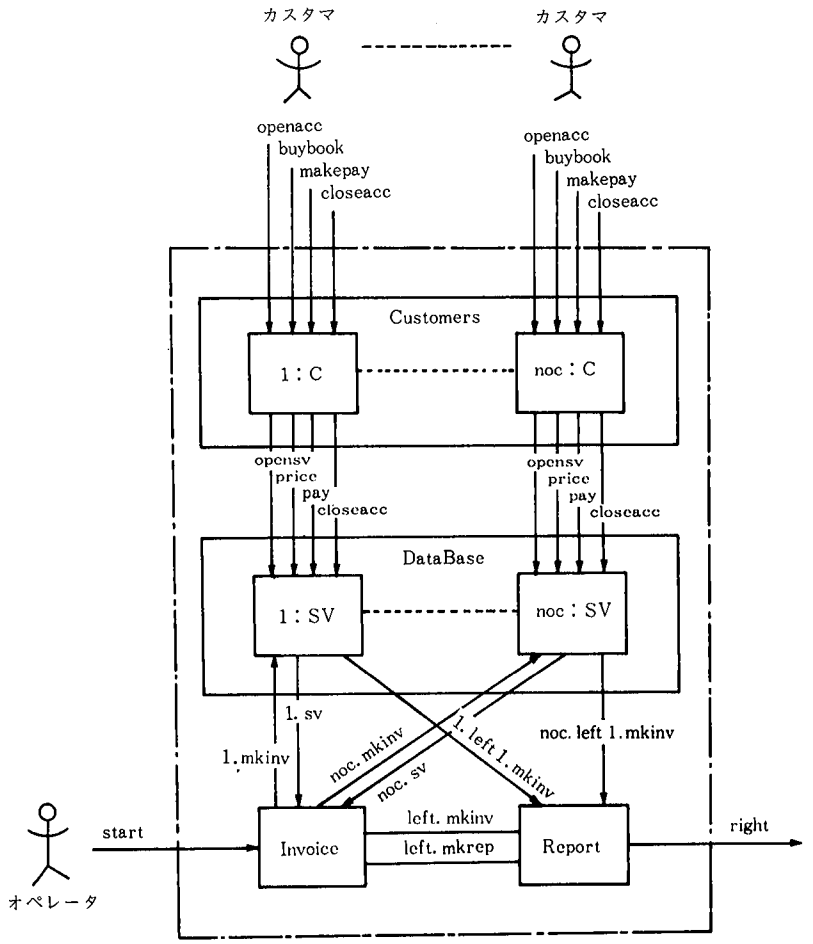


図2　INVSYSTEM

Fig. 2　Overview of INVSYSTEM

［3］　Customers は，noc 個のプロセス i：C(i ∈ Id) が同時並行して動くサブシステムである．

i：C はラベル付きプロセスで，ここでは Id の要素がラベルになっている．i：C の振る舞いは，どれも C と「同型」だが，ラベルが違えばそれらの事象集合には共通な要素はまったくないので，noc 個の i：C プロセスは，互いに独立に，相互には何も交信することなく動く．i：C が交信するのは，このシステムの外側にいる顧客 (i) と，システム内の DataBase の一要素である i：SV プロセスだけである．

C は，顧客の振る舞いをシミュレートするプロセスで，その動きを定義するのが [3.1] と [3.2] である．ここで，？と！は，それぞれチャネル入力と出力を表す．

C は顧客の口座開設指示 (openacc) によって動き始め，まず対応する SV に初期化指示 (opensv) を出す．それ以降の振る舞いは C1 に従う．

C1 の振る舞いは，次の 3 通りに分かれる．

① 本の購入：

buybook チャネルから，本の値段(u)を受け取り，それを price チャネルを通して SV に知らせ，C1 の初期状態に戻る．

② 支払い：

makepay チャネルから支払い金額(v)を受け取り，それを pay チャネルを通して SV に知らせ，C1 の初期状態に戻る．

③ 口座の閉鎖：

顧客の口座閉鎖指示（closeacc）を受けると，SV に閉鎖指示（closesv）を出す．そのあと，C の初期状態（C1 の，ではない）に戻って，顧客登録指示（openacc）を待つ．

［4］ DataBase は，noc 個のプロセス i：SV(i ∈ Id) が同時並行して動くサブシステムである．

SV は，顧客の状態ベクトルの役割を担うプロセスで，顧客の前月末残高(balance)，当月売上累計(sale)，当月入金累計(receipt)について，その最新の状態を保持する．請求書作成時には，それらの値を請求書作成プロセス Invoice に教える．

SV の振る舞いは次の 2 通りである．

① 初期化：

C から初期化指示（opensv）を受けると，後は V(0, 0, 0) として振る舞う．

② 初期化以前の請求書作成指示への返答：

初期化以前に Invoice プロセスからこの指示（mkinv）を受けると，SV は，作成不要指示（notmkinv）を Invoice に返し，SV の初期状態に戻って，初期化指示（opensv）を待つ．

V(balance, sale, receipt)の振る舞いは，次の 3 通りである．

① 当月売上の累積：

C からチャネル price を通して送られてきた本代 u を累積して，以降は V(balance, sale＋u, receipt)として振る舞う．

② 当月入金の累積：

C からチャネル pay を通して送られてきた入金額 v を累積し，あとは V(balance, sale, receipt＋v)として振る舞う．

③ 状態ベクトルの送信：

Invoice から，請求書作成指示（mkinv）を受けると，チャネル sv を通してその時の状態ベクトルを Invoice に返してから，状態ベクトルの値を更新し，あとは V(balance＋sale－receipt, 0, 0)として振る舞う．

④ 終了化：

C から終了化指示(closesv)を受けると，Report に請求書作成指示を出して，SV の（V の，ではない）初期状態に戻る．

［5］ 請求書作成開始指示（start）によって動きだし，あとは Inv(0)(0, 0, 0) として振る舞う．

Inv(i)(tb, ts, tr) の振る舞いは，i の値によって次のように変わる．

① i ∈ Id の場合：

顧客 i の請求書を作成し，月報を作るために状態ベクトルの値の累計を取る．まず i：SV に請求書作成指示（i. mkinv）を出す．あとは i：SV からの返答如何によって，その振る舞いは次の 2 通りに分かれる．

ⓐ 作成不要指示（i. notmkinv）が返ってきたら，何もしないで次の処理 Inv(i+1)(tb, ts, tr) に移る．

ⓑ チャネル sv を通して状態ベクトル x が返ってきたら，チャネル left. mkinv を通して，Report に顧客 ID とその状態ベクトルを送り，状態ベクトルの値を累計し，次の処理 Inv(i+1)(…) に移る．

② i=noc+1 の場合：

この時の (tb, ts, tr) には，すべての顧客の状態ベクトルの値が集計されている．この集計結果をチャネル left. mkrep を通して Report に送ってから，Invoice の初期状態に戻って，請求書作成開始指示（start）を待つ．

［6］ Report は，Invoice と SV の指示にもとづいて，請求書および月報を編集する．

4.3 CSP 仕様の実現

本節では，CSP による仕様 INVSYSTEM を Modula-2 の並行プロセスで実現した例を説明する．はじめに CSP プロセスの実現法を検討し，次にその結果を課題である書店のシステムに適用する．

4.3.1 CSP プロセスの実現法

Modula-2 のプロセスは，最も原始的なコルーチン機構である．これを用いて CSP プロセスを構築すると相当に繁雑になる．ここでは，N.Wirth によって提示されているモニタ方式の並行プロセスを使用する．Wirth は，WAIT (signal) と SEND (signal) で動作する並行プロセスを Modula-2 で実現している[13]．

1） 事象とメッセージ通信……CSP のプロセスは，事象で同期をとって動作する．ある事象が起こると，それを待つすべてのプロセスが起動する．メッセージ通信は，通信チャネルとメッセージを複合したものである．通信チャネルも事象の一つであり，これで入力と出力の同期をとりメッセージを転送する．メッセージは，システムで通信する最大レコードで定義する．

事象には，特定プロセスに固有のものからすべてのプロセスに共通のものまでがある．書店システムの例では，start は Invoice プロセス固有の事象であり，openacc はすべての Customers プロセスに共通の事象である．

CSP のプロセスを構築するためには，事象をどう表すかが中心問題になる．この実現では，枚挙型の EVENT で事象を定義した．しかし，これだけでは共通な事象をもつプロセス中の特定プロセスを表せない．そこで後に述べる PROCESS 型のプロセス識別子と組にして，〈PROCESS, EVENT〉で事象を表すことにする．この結果，start のようなシステムに固有の事象でも，〈Inv, start〉としなければならなくなる．

2） CSP プロセスの生成……CSP プロセスを制御する型や手続きは，モジュール CSP に密閉されている．プロセスの定義に必要な EVENT や MESSAGE はインポートされ，定義結果はエクスポートされている．

各プロセスは，固有のプロセス記述子 (ProcessDescriptor) をもっており，型 PROCESS はプロセス記述子をポイントする．プロセスには，PROCESS 型の値が対応しており，これがプロセス識別子になる．プロセス記述子は，signal と event と msg のフィールドをもつレコードで構成する．signal はプロセスを停止させたり，再開させるのに使用する．

CSP プロセスは，手続き Process (P, p) で生成する．ここで，P はプロセス・テキストの手続き名，p は生成したプロセス識別子を受取る PROCESS 型の変数である．

CSP は，システムの外にある環境プロセスと並行に動作する．書店システムの環境プロセスは，書店や本を買う人達である．この実現では，主プログラムが環境プロセスとシステムの間のインタフェースをとる．主プログラムのプロセス識別子は，CSP モジュールの初期化部で生成しエクスポートする．

3） 事象の発生と待ち……a を事象，P をプロセスとするとき，

$(a \rightarrow P)$

は，事象 a があり，次に P の動作をするプロセスを表す．この記法は，プロセス自身が事象を発生することと，事象を待つことの二つの意味を含んでおり，そのどちらかは文脈で判断する．このため，次の二つの手続きを設ける．

Event (p, e)*：事象 〈p, e〉 を発生する．p は事象を伝えるプロセスの識別子である．

Prefix (p, e)：事象 〈p, e〉 を待つ．ここで，p は自分自身の識別子である．

4） 複数事象の選択的待ち……a, b を事象，P, Q をプロセスとするとき，

$(a \rightarrow P \mid b \rightarrow Q)$

は，事象 a か b があり，それにより次の動作が P か Q になるプロセスを表す．手続き Choice (p, x) は，これを近似的に実現する．p は自分自身の識別子で，プロセスは p の事象が発生するまで待つ．発生した事象は，変数 x に代入される．プロセスは，x の値により処理を分ける．p が待つ事象でなければ，プロセス側でそれを無視する．

5） メッセージ通信……c をチャネル事象，m をメッセージとするとき，c ? m はチャネル入力を，c ! m はチャネル出力を表す．チャネルの入出力は，事象 c で同期をとりメッセージ m を転送する．

チャネル入力は，手続き Input (p, c, x) で近似的に実現する．p は自分自身の識別子で，p の事象が発生するまで待つ．c は発生した事象を代入する変数，x はメッセージを代入する変数である．c がチャネル事象でなければ，プロセス側でそれを無視する．Input は，Choice の動作を含むので，その代用に使うこともできる．

チャネル出力は，手続き Output (p, c, m) で実現する．p は事象 c を伝えるプロセスの識別子で，m は転送するメッセージである．なお，Output は，Event の動作を含んでいる．

---

* 手続き Event は，まず事象を伝えるプロセスが待ち状態にあることを確認するため，手続き AWAIT を用いる．AWAIT は，一時的にコントロールを放棄し，後に自動的に再開させるもので，Wirth の並行プロセスを拡張した．なお，プロセスが待ちにないときの SEND (signal) は無効になる．

以下に，モジュールCSPのコードを示す．

```
(* 書店の売掛請求システム： CSPの設計をModula_2のプロセスで実現 *)

MODULE INVSYSTEM;
  FROM SYSTEM IMPORT TSIZE;
  FROM Storage IMPORT ALLOCATE;
  FROM Processes IMPORT
    SIGNAL, StartProcess, SEND, WAIT, Awaited, Init, AWAIT;
  FROM PC9801 IMPORT
    KeyPressed, ReadInt, Write, WriteString, WriteInt, WriteLn,
    StartScroll, SaveCursor, SetCursor, ClearLine, Invert, Bell;


  TYPE  EVENT = ( openacc, opensv, buybook, makepay, price, pay,
                  closeacc, closesv, start, sv, mkinv, mkrep,
                  notmkinv );

        MESSAGE = RECORD
                    i, b, s, r: INTEGER
                  END;

  MODULE CSP;
    IMPORT  EVENT, MESSAGE, TSIZE, ALLOCATE,
            SIGNAL, StartProcess, SEND, WAIT, Awaited, Init, AWAIT;

    EXPORT  PROCESS, Main,
            Process, Event, Prefix, Choice, Output, Input;


    TYPE  PROCESS = POINTER TO ProcessDescriptor;

          ProcessDescriptor = RECORD
                signal: SIGNAL;  event: EVENT;  msg: MESSAGE
            END;

    VAR  Main: PROCESS; (* 主プログラム・プロセス *)


    PROCEDURE Process (P: PROC; VAR p: PROCESS);
    BEGIN
      ALLOCATE (p, TSIZE(ProcessDescriptor));   (* CSPプロセス *)
      Init (p^.signal);
      StartProcess (P, 400)
    END Process;


    PROCEDURE Event (p: PROCESS; e: EVENT);
    BEGIN
      WITH p^ DO
        WHILE NOT Awaited (signal) DO AWAIT END;
        event := e;  SEND (signal)              (* e -> P *)
      END
    END Event;


    PROCEDURE Prefix (p: PROCESS; e: EVENT);
    BEGIN
      WITH p^ DO
        REPEAT WAIT (signal) UNTIL event = e    (* e -> P *)
      END
    END Prefix;


    PROCEDURE Choice (p: PROCESS; VAR x: EVENT);
    BEGIN
      WITH p^ DO
        WAIT (signal);  x := event              (* x:B -> P(x) *)
      END
    END Choice;


    PROCEDURE Output (p: PROCESS; c: EVENT; m: MESSAGE);
    BEGIN
      WITH p^ DO
        WHILE NOT Awaited (signal) DO AWAIT END;
        event := c;  msg := m;  SEND (signal)   (* c ! m *)
      END
    END Output;
```

```
PROCEDURE Input (p: PROCESS; VAR c: EVENT; VAR x: MESSAGE);
BEGIN
  WITH p^ DO
    WAIT (signal);  c := event;  x := msg   (* c ? x *)
  END
END Input;


BEGIN
  ALLOCATE (Main, TSIZE(ProcessDescriptor));
  Init (Main^.signal)
END CSP;
```

4.3.2 書店の売掛請求システムの実現

これまでの結果をCSPの設計仕様［3］(Customers)，［4］(DataBase)，［5］(Invoice)，［6］(Report)に適用し，各プロセスをModula-2で実現する．プロセスは，設計仕様にほぼ忠実に実現でき，CSPの設計記述能力を実証するものと言える．

プロセスは，テキストの上では引数をもたない手続きと同じ形をしている．ただし，プロセスは，ループしており手続きの出口に行くことはない．また，手続き呼び出しの代りに，Start Processによって手続きの入口に入る．手続きの変数は，プロセスの状態になる．

プロセス達は，主プログラムで最初に生成される．CustomersとDataBaseは同じ動作をするnoc個のプロセスがある．

1） CustomersとDataBaseプロセス……CustomersとDataBaseは，プロセス識別子C[i]とSV[i]をもつ．ここで，iは口座番号に相当し，大域変数accを用いて各プロセスに伝えられる．

Customersの動作は，設計仕様から明らかであろう．buybookチャネルとmakepayチャネルの入力待ちおよびcloseaccの事象待ちにInputを使い，closeaccの時はメッセージを無視している．

DataBaseは，状態変数Vで口座内容を保持している．priceチャネルとpayチャネルで口座内容を更新し，mkinv事象で口座内容を出力する．ただし，口座がopenされていなければ，notmkinv事象を起こす．

```
(* 口座番号iごとの顧客プロセス達とデータベース *)

CONST noc = 20;  (* number of Customers *)

VAR   acc: INTEGER;

      C, SV: ARRAY [1..noc] OF PROCESS;


PROCEDURE Customers;
  VAR  x: EVENT;  msg: MESSAGE;  i: INTEGER;
BEGIN
  i := acc;   (* 口座番号 *)
  LOOP
    Prefix (C[i], openacc);
    WriteString (' 開設 ');  WriteLn;  Event (SV[i], opensv);
    LOOP
      Input (C[i], x, msg);
      CASE x OF
        openacc : Bell (1)
      | buybook : Output (SV[i], price, msg)
      | makepay : Output (SV[i], pay, msg)
      | closeacc: EXIT
      END
    END;
    WriteString (' 閉鎖 ');  Event (SV[i], closesv)
  END
END Customers;
```

```
PROCEDURE DataBase;
  VAR  x: EVENT;  V, msg: MESSAGE;  i: INTEGER;
BEGIN
  i := acc;  V.i := acc;        (* 口座番号 *)
  LOOP
    Choice (SV[i], x);
    CASE x OF
      opensv: WITH V DO
                b := 0;  s := 0;  r := 0;
                LOOP
                  Input (SV[i], x, msg);
                  CASE x OF
                    price  : s := s + msg.s
                  | pay    : r := r + msg.r
                  | mkinv  : Output (Inv, sv, V);
                             b := b + s - r;
                             s := 0;
                             r := 0
                  | closesv: EXIT
                  END
                END;
                Output (Rep, mkinv, V)
              END
    | mkinv:  Event (Inv, notmkinv)
    END
  END
END DataBase;
```

2）InvoiceプロセスとReportプロセス……Invoiceのプロセス識別子は，Invで，start事象で起動する．Invoiceは，openしているDataBaseから口座内容を受取り請求書を発行する．また，当月の合計金額を集計し報告書を作る．請求書と報告書の編集は，Reportプロセスに任せる．

報告書を出力するとInvoiceの仕事は終わり，主プログラムにstart事象を伝える．この間，主プログラムは待ち状態になっている．

Reportのプロセス識別子はRepである．プロセスは，常にチャネル入力待ちしており，チャネル事象(mkinvかmkrep)により，請求書か報告書の編集をする．なお，チャネルは二つのプロセス間にユニークで片方向の通信をするため，i. left 1. mkinv, left. mkinv, left. mkrepで事象を区別しているが，繁雑になるので略した．また，rightは表示装置にした．

```
(* 請求書と報告書プロセス *)

VAR   Inv, Rep: PROCESS;

PROCEDURE Invoice;
  VAR  x: EVENT;  total, msg: MESSAGE;  k: INTEGER;
BEGIN
  LOOP
    Prefix (Inv, start);
    WITH total DO
      b := 0;  s := 0;  r := 0;
      FOR k := 1 TO noc DO
        Event (SV[k], mkinv);
        Input (Inv, x, msg);
        CASE x OF
          sv: Output (Rep, mkinv, msg);
              b := b + msg.b;
              s := s + msg.s;
              r := r + msg.r
        | notmkinv: (* none *)
        END
      END
    END;
    Output (Rep, mkrep, total);
    Event (Main, start)        (* 主プログラムを起動 *)
  END
END Invoice;
```

```
PROCEDURE Report;
  VAR  x: EVENT;  msg: MESSAGE;
BEGIN
  LOOP
    Input (Rep, x, msg);
    CASE x OF
      mkinv: WriteString (' 請求書：口座番号 ');  WriteInt (msg.i, 2)
    | mkrep: WriteString (' 報告書：合計金額   ')
    END;
    WriteString (' 前残 ');  WriteInt (msg.b, 6);
    WriteString (' 買上 ');  WriteInt (msg.s, 5);
    WriteString (' 入金 ');  WriteInt (msg.r, 5);
    WriteLn
  END
END Report;
```

3） 主プログラム……主プログラムは，プロセス識別子 Main をもち，これはモジュール CSP がエクスポートする．主プログラムは，CSP の各プロセスを生成し，キーボードとモニタを使って環境プロセスとシステム間のインタフェースをとる．詳細は次頁の動作例を参照のこと．

```
(* 主プログラム：プロセス達を生成し環境とのインタフェースをとる *)

  VAR   menu: CHAR;      msg : MESSAGE;

BEGIN
  FOR acc := 1 TO noc DO
    Process (Customers, C[acc]);
    Process (DataBase, SV[acc])
  END;
  Process (Invoice, Inv);
  Process (Report, Rep);
  Invert (TRUE) ;  WriteString (' acc ( open | buy | pay | close ) ');
  Invert (FALSE);  WriteString ('  ');
  Invert (TRUE) ;  WriteString (' invoice ');
  Invert (FALSE);
  StartScroll (3);
  LOOP
    SaveCursor;  Write ('>');
    REPEAT AWAIT UNTIL KeyPressed();
    ReadInt (acc, menu);  SetCursor;  ClearLine;
    IF (1 <= acc) & (acc <= noc) THEN
      WriteString (' 口座番号 ');  WriteInt (acc, 2);
      CASE menu OF
        'o':  Event (C[acc], openacc)
      | 'c':  Event (C[acc], closeacc)
      | 'b':  WriteString (' 購入 ');  ReadInt (msg.s, menu);  WriteLn;
              Output (C[acc], buybook, msg)
      | 'p':  WriteString (' 支払 ');  ReadInt (msg.r, menu);  WriteLn;
              Output (C[acc], makepay, msg)
      ELSE
       SetCursor;  ClearLine;  Bell (1)
      END
    ELSIF menu = 'i' THEN
      WriteLn;
      Event (Inv, start);
      Prefix (Main, start);      (* 請求書の終りを待つ *)
      WriteLn
    ELSE
      Bell (1)
    END
  END
END INVSYSTEM.
```

以下は動作例である．

```
acc ( open | buy | pay | close )      invoice

口座番号  1 開設
口座番号 10 開設
口座番号 20 開設
口座番号  1 購入 1000
口座番号  1 購入 2000
口座番号 10 購入 3000
口座番号 20 購入 4000

請求書：口座番号  1 前残      0 買上  3000 入金     0
請求書：口座番号 10 前残      0 買上  3000 入金     0
請求書：口座番号 20 前残      0 買上  4000 入金     0
報告書：合計金額    前残      0 買上 10000 入金     0

口座番号 10 支払 3000
口座番号 10 閉鎖  請求書：口座番号 10 前残   3000 買上     0 入金  3000

請求書：口座番号  1 前残   3000 買上     0 入金     0
請求書：口座番号 20 前残   4000 買上     0 入金     0
報告書：合計金額    前残   7000 買上     0 入金     0

>
```

5. おわりに

設計法1は，表示的意味論の枠組といっても単に関数型言語でプログラムを書いたにすぎない．continuationなど厄介なことは現れないので簡単であり，企業内で採用できる方法である．普及のためには，枠組をさらに明確にし，具体構文，抽象構文，文脈条件などと，記述項目を分離して書くといいようである．

設計法2はCSPの記法でad hocに仕様を書いたようにみえるが，これはJSDの考えをCSPで記述したものといえる．したがって，まずJSDの普及を図る必要がある．

設計法1，2ともに，結局，対象主導型の設計法である．仕様を作成すると実現方法が自然にえられるので，仕様から実現を導くプログラム生成系ができる可能性がある．仕様作成，設計，文書化，保守などのための支援系も考えることができ，これらのためにも，両者の記述言語を設定し，保守し発展させる必要を感じる．なお，本発表は，ソフトウェア工学味見会(板倉教，加藤潤三，峰尾欽二，森澤好臣，宗像清治，澤田晟司，染谷誠，田端福雄，山崎利治）の成果の一部である．

参考文献

[1] 加藤潤三，染谷誠，板倉教，田端福雄，森澤好臣，山崎利治：仕様記述の味見，情報処理学会ソフトウェア工学研究会，40-12, 1985. 2. 7.

[2] J.D.ワーニエ，B.M.フラナガン著，鈴木君子訳：ワーニエ方式によるプログラミング学習（上，下），日本能率協会，1973.

[3] M.A.ジャクソン著，鳥居宏次訳：構造的プログラム設計の原理，日本コンピュータ協会，1980.

[4] K. Futatsugi, K.Okada：A hierarchical structuring method for functional software systems, 6th ICSE, IEEE, 1982, pp. 393～402.

[5] R.Nakajima, T.Yuasa (eds)：The IOTA programming system, LNCS No.160, Springer-Verlag, 1983.

[6] B.A.Silverberg, et al.：The HDM Handbook 3 vols, SRI Internationl, 1983.

[7] C.B.Jones：Systematic software development using VDM, Prentice/Hall, 1986.

[8] E.W.ダイクストラ著，浦昭二，土居範久，原田賢一訳：プログラミング原論，サイエンス社，1985.
[9] D.Gries：The Science of programming, Springer-Verlag, 1981.
[10] C.A.R. Hoare：Communicating sequential processes , Prentice/Hall, 1985.
[11] 中島玲二：数理情報学入門，朝倉書店，1982.
[12] M.A.Jackson：System development, Prentice/Hall, 1983.
[13] N.Wirth：Programming in Modula-2, 3rd ed., Springer-Verlag, 1985.
[14] D.G.Kourie：The design and use of a Prolog trace generator for CSP, Software-Practice and Experience, 17, 1987, pp. 423～438.

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年日本ユニバック㈱入社，教育部のプログラミング教育担当を経て，現在，生産技術一部所属．

論文

JSD仕様の直接実行

Direct Execution of a JSD Specification

加 藤 潤 三

要 約 現在，筆者らは，ジャクソン・システム開発法の支援環境（JSE: Jackson System development Environment）を開発している．今回JSEの部分系である“JSD仕様の直接実行系”の基本的な機能を実現したので報告する．JSDを適用したシステム開発におけるラピッド・プロトタイピングが，この“JSD仕様の直接実行系”を利用することによって可能になった．

JSEでは，JSD仕様のテキスト表現と，その直接実行のためにJSL（Jackson System development Language）を導入した．本稿においては，JSLの主要部とその解釈系（interpreter）を中心に説明する．なお，JSDを知らない読者のためにJSDの簡単な説明も挿入しておいた．

Abstract This paper presents a direct execution program of a JSD (Jackson System Development) specification, which is developed as a part of the JSE (Jackson System development Environment). We introduced a language, named JSL (Jackson System development Language) which is a textual form of a JSD specification. We can make a prototyping approach in a JSD project using the JSL interpreter which executes a JSD specification directly.

The paper describes the main part of JSL and explains its interpreter. We insert a short explanation of JSD for a better understanding of the JSL interpreter.

1. はじめに

業務システムの開発における誤りの多くは，コード化よりも要求仕様や設計に起因する[1]．要求仕様にある誤りは，テスト段階のような後の段階で検出されることが多い．誤りの発見が遅くなるほど，それが引き起こす損害から立ち直ることがむずかしくなる．また，利用者もこの種の開発過程に不安を感じるであろう．誤りの早期発見には仕様の実行が有効である．それによって，利用者の要求の誤解や仕様の記述の誤りが見つけられるだけでなく，システムが何をするのかを利用者に示すことができる．仕様の誤りを未然に検出したり，その正しさを保存した実現が可能であれば，仕様の直接実行がソフトウェアの品質に与える影響は一層大きくなる．このためには，業務システムの開発を要求段階から保守段階まで，一貫して支援するシステム開発法の採用が求められる．さらに，その開発法を支援する道具があればより効果が上がるであろう．システム開発法とその支援道具は，利用者や開発者の“システムを正しく記述するための能力”を向上させるためにも重要である．

ジャクソン・システム開発法（JSD）は業務処理分野において実績を持つ実用的な方法である[2][3]．JSDは，要求の総体的な表現しかないプロジェクトの開始時からシステムの完成，さらには保守においても適用できる．また，支援道具の充実によって，JSDの実用性は増大するであろう．すでに，いくつかのJSD支援道具が利用可能であ

り[3]，たとえばAdaのシミュレーション・パッケージを使ったJSD仕様の実行の実験報告[4]も発表されている．

JSD仕様の直接実行は，仕様の検証という点で意味を持つ．仕様の検証にはシミュレーション・モデルを生成しそれを実行してみる方法[4]と，仕様を直接実行する方法が考えられる．JSDの仕様は原理的には実行可能なため，筆者らは実際に直接実行する処理系の開発を決定し，JSD仕様のテキスト表現かつ実行可能な仕様記述言語としてJSLを導入した．なお，JSEの直接実行系には，トレース機能などの“結果の理解を容易にするための機構”が用意されている．

以降，第2章でJSEの概要，第3章でJSLの主要部を説明し，第4章でJSDの簡単な紹介とその仕様の解釈を述べ，Jacksonの著書にある簡単な例[2]を使いJSLの実行結果を示す．

2. JSE

JSEはJSD仕様の図形エディタや，JSL解釈系，JSD変換系(JSD仕様を目的言語に変換する）からなる．JSEの概要を図1に示す．

開発者が図形表現を使い，仕様を見たり操作できれば便利である．図形表現は，テキスト表現よりも仕様の静的側面の理解を容易にする．図形エディタは構造図とJSLプロセス宣言との間の変換を双方向で行う[6]．これは，ちょうどデータ構造エディタ[7]と同様の機能を持つ．さらに，このエディタはJSLで記述された仕様のネットワーク図SSD (System Specification Diagram) も描く．JSDの図式表現で仕様を記述しさえすれば，JSEのエディタはそれをJSL直接実行系やJSL変換系の入力に機械的に変換するため，プロジェクトの使用言語を図式表現で統一できる．

JSL解釈系は，JSLで記述された仕様を実行する．具体的には後で説明する．なお，JSDの仕様は，システムの実際の開発資源や実行環境からは独立しているので，仕様

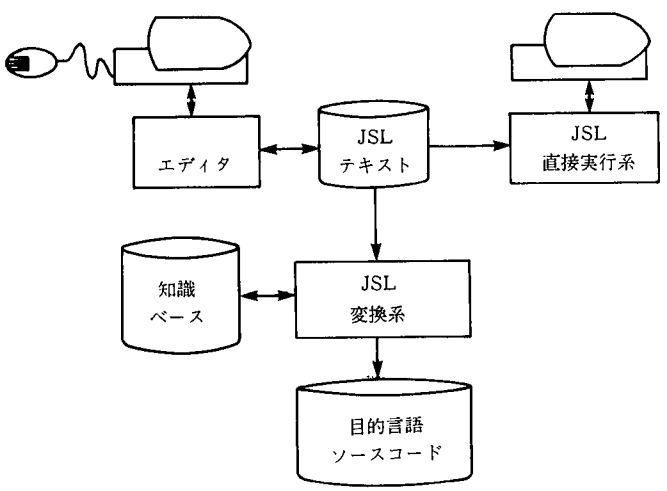


図1 JSEの構成

Fig. 1 Overview of JSE

の確認の後で実行環境に適合するように，実行可能なコードに変換しなければならない．JSDの実現段階では，仕様を実際の実行環境で実行可能なコードに変換する．このため，JSDは転換（inversion）や分離（dismemberment）のようないくつかの機械的変換法を用意している．

JSL変換系は，JSLで記述された仕様を知識ベースによって伝統的言語（たとえばCOBOL)のコードに変換する．知識ベースは，全体へ適用される規則と部分へ適用される規則からなる．JSLによる仕様記述にも実現のためのデータ型の宣言が含まれているが，データ型の宣言がないときは知識ベースの規則に従って，データ型が推論され省略時解釈値が与えられる．JSL変換系は，当面は一つのプロセッサ上での実現を支援する．したがって，複数の仮想プロセッサを含めた複数プロセッサ上での実現の支援は将来の設計に委ねている．

3. JSL

JSLは，JSD仕様をテキスト形式で表現するために導入したものである．このテキストは，処理系の特性に合わせてJSLパーサによって，さらに別の内部表現に変換される．JSLのプログラムはシステム宣言によって記述され，イベント，データストリーム，状態ベクトル，プロセスの各宣言部で構成される．なお，プロセス宣言部は，JSDの構造テキストで記述したものである．

本稿ではJSL構文を説明するのに拡張BNFを使う．拡張BNFは，BNFに零回も含めた繰り返しを{ }で，零回か1回を[ ]で表現する記法を追加したものである．

3.1 システム宣言

これはJSLの最上位の宣言である．

〔構文〕

```
〈SystemDeclaration〉
  : : = 'SYSTEM'
        〈SystemHeader〉
        〈EventDeclaration〉
        〈DataStreamDeclaration〉
        〈StateVectorDeclaration〉
        〈ProcessClassDecralation〉
        'SYSTEM' 'END'
```

3.2 システム見出し部

仕様の識別子をこの宣言で与える．

〔構文〕

```
〈SystemHeader〉: : = ' : ' 〈Idenfier〉
```

〔例〕

```
SYSTEM : simplebank
```

3.3 イベント宣言部

プロセスの振舞いは，イベントによって定義される．システムで発生するすべての

イベントをここで宣言する．この宣言は，通信文(message)の型とその宛先の宣言を意味している．ここで言う宛先とは，通信文に埋め込まれている“通信文を受け取るプロセス”の識別子のことである．通信文本体は，属性によって示される値の列である．通信文本体からデータを参照する時はこの属性名を使う．JSD仕様の行動(Action)は，ここでいうイベントである．

事務処理システムにおいては，イベントの発生がトランザクションの発生を意味する．トランザクションは，レコードによって表現される．以降，通信文の型(message type)をイベントの意味で使う．

〔構文〕

```
<EventDeclaration>
  : : = 'EVENT'
        {<EventDescription>}
        'EVENT' 'END'
<EventDescription>
  : : = <EventName>
        [<ProcessId>]
        '('{Attribute>}')'
```

〔例1〕

```
EVENT
  invest customer_id (date customer_id amount)
EVENT END
```

この例では，イベントの名前はinvestであり，このイベントの属性は，date, customer_id, amountである．このイベント（invest）を受け取るプロセスの識別子はcustomer_idの値である，ということを宣言している．

〔例2〕

```
EVENT
  enquiry (customer_id)
EVENT END
```

これは宛先プロセスの識別子の宣言がない例である．プロセスの識別子の省略が可能なのは，この通信文が通過するデータストリームが一つで，かつ結合されているプロセスも一つである時で，宛先が唯一つに決まっている時である．enquiryというイベントは，属性にcustomer_idを持ち，宛先が唯一つ決まっている通信文の型である．

3.4 データストリーム宣言部

データストリーム・クラスは，このデータストリーム宣言で定義される．データストリーム・クラスは長さの制限がない待ち行列であり，そこに入る通信文の型と，このデータストリームで結合される1対のプロセス・クラスによって規定される．データストリーム宣言で引用する通信文の型は，すべてイベント宣言部で定義されていなければならない．また，プロセス・クラス宣言部にあるすべてのread/write命令のデータストリーム名は，このデータストリーム宣言部で定義されていなければならない．

〔構文〕

```
〈DataStreamDeclaration〉
  : : = 'DATASTREAM'
        {〈DataStreamClass〉}
        'DATASTREAM' 'END'
〈DataStreamClass〉
  : : = 〈DataStreamName〉
        ['('{〈EventName〉}')']
        ':' 〈DataStreamConnection〉
〈DataStreamConnection〉
  : : = 〈SenderProcessClass〉
        〈MultiplicityMark〉
        〈DataStreamConnector〉
        〈MutiplicityMark〉
        〈ReceiverProcessClass〉
〈SenderProcessClass〉
  : : = '#' | 〈ProcessClassName〉
〈ReceiverProcessClass〉
  : : = '#' | 〈ProcessClassName〉
〈DataStreamConnector〉
  : : = '_' | 〈DataStreamName〉
〈MultiplicityMark〉
  : : = '〉' '|' '〉〉'
```

注：'#'は外部世界を意味する．'_'は宣言されているデータストリーム名を省略時解釈することを意味し，'〉'は1を意味し，'〉〉'は多を意味する．

〔例1〕

```
DATASTREAM
  c (invest pay_in withdraw terminate)
    :# 〉 _ 〉 customer_1
DATASTREAM END
```

この例ではcというデータストリーム・クラスはinvest, pay_in, withdraw, terminateという型の通信文を要素として持ち，外部世界からプロセス・クラスcustomer_1へ結合されている．

〔例2〕

```
DATASTREAM
  exception_report ( )
    : customer_1 〉 _ 〉 #
DATASTREAM END
```

イベント名を省略した例である．省略した場合，通信文の型は宛先がシステムの外で，属性は1行の文字列であると解釈する．これは出力書式が未決定で，出力が報告書である時に宣言する．データストリーム・クラス exception_report は customer_1 からシステムの外へ結合されており，その出力は1行の文字列である．

3.5 状態ベクトル宣言部

状態ベクトル・クラスはこの宣言部で定義される．状態ベクトル・クラスはJSDにおける状態ベクトル結合を定義する．状態ベクトル宣言部は，参照されるプロセスの局所変数や，アクセス方法の宣言を含む．状態ベクトル結合の名前は，プロセスの内部状態の複写を要求する通信文と，その回答の通信文が通過する経路を示す．アクセス方法では，状態ベクトルの選択条件と参照順序を指示する．JSLは，一つのプロセスに対し，複数の状態ベクトルの参照を許す．

〔構文〕

```
<StateVectorDeclaration>
  : = 'STATEVECTOR'
      {<StateVectorClass>}
      'STATEVECTOR' 'END'
<StateVectorClass>
  : : = <StateVectorName >
        [ ' ( ' { <LocalVariableName> } ' ) ' ]
        [ 'ORDER' ' = ' {<LocalVariableName> }]
        [ 'PREDICATE' ' = ' <BooleanExpression>]
        ' : ' <StateVectorConnection>
<StateVectorConnection>
  : : = <InspectedProcessClassName>
        <MultiplicityMark>
        <StateVectorConnector>
        <MultiplicityMark>
        <InspectProcessClassName>
<StateVectorConnector>
  : : = ' _ ' | <StateVectorName>
```

注：'_'マークは，宣言した状態ベクトル名を意味する．<MultiplicityMark>は，データストリーム宣言部と同じ構文である．

〔例1〕

```
STATEVECTOR
  cv (balance) : customer_1  >>_>  enquiry_fn
STATEVECTOR END
```

プロセス・クラスcustomer_1のプロセスは，balanceという名前の局所変数をもっており，プロセス・クラスenquiry_fnはcustomer_1クラスの内部状態を状態ベクトル結合cvを使い参照する．cvは多対1である．すなわち，enquiry_fnの具体例の一つに，多くのcustomer_1の具体例が結合されている．

〔例2〕

```
STATEVECTOR
  cvs (customer_id balance last_date)
  ORDER = balance customer_id
  PREDICATE = balance > 0
```

```
  : customer_1 >>_> fn_1
STATEVECTOR END
```

ここで状態ベクトルcvsは，プロセス・クラスcustomer_1の局所変数のうちcustomer_id, balance, last_dateを参照し，balanceの値が正のプロセスでbalance, customer_idの値の昇順にアクセスするという状態ベクトル結合で，プロセス・クラスcustomer_1の多くの具体例に対して，プロセス・クラスfn_1の一つの具体例が結合されていることを宣言している．

3.6 プロセス・クラス宣言部

システム内のプロセス・クラスの内部構造はここで宣言される．この部分の構文は，JSD仕様の構造テキストと同じである．本稿では，JSLの主な通信命令（communication statement）のみを説明する．

〔構文〕

```
〈ProcessClassDeclaration〉
  : : = 'PROCESS'
        {〈ProcessClass〉}
        'PROCESS' 'END'
〈ProcessClass〉
  : : = 〈ProcessClassName〉
        [〈ProcessClassType〉]
        [〈ConstantDeclaration〉]
        〈Structure〉
        〈Operations〉
        〈Conditions〉
        〈ProcessClassName〉 'END'
〈Operations〉
  : : = 'OPERATIONS'
        {〈OperationDeclaration〉}
〈OperationDeclaration〉
  : : = 〈Number〉 ' : ' 〈Statement〉
〈Statement〉
  : : = 〈AssignStatement〉
      | 〈CommunicationStatement〉
      | 〈ProcedureCall〉
〈CommunicationStatement〉
  : : = 〈ReadStatement〉
      | 〈WriteStatement〉
      | 〈GetStateVectorStatement〉
      | 〈WaitStatement〉
```

3.6.1 Read命令

Read命令はデータストリームから通信文を得て，〈VariableName〉によって指示された変数に代入する．変数が省略されているとき，プロセスは通信文の属性名の頭に'.'を付けることによって通信文中の属性の値をアクセスできる．

たとえば，.amount は通信文中の属性名 amount の値を示す．また，データストリームに通信文がない時は，この read 命令で通信文がデータストリームに入るまで，プロセスを停止し，通信文が入ったとき，停止した read 命令からプロセスの実行を再開する．

〔構文〕

```
<ReadStatement>
  : : = 'sread'
        '('
        <DataStreamName> {<DataStreamName>}
        [ ',' <VariableName> ]
        ')'
```

〔例 1〕　sread (c)

通信文をデータストリーム c から読む．

〔例 2〕　sread(a b)

通信文は，データストリーム a と b を「ラフマージ」(rough merge) したデータストリームから読まれる．2 個以上のデータストリームから読める順に通信文をマージすることを「ラフマージ」という．JSD ではラフマージを SSD で静的に定義する．JSL でも同様である．たとえば，JSL では，同じプロセス宣言の中で sread (a b) と sread (a) を使用できない．

〔例 3〕　sread(a , rec)

データストリーム a から通信文を読み，変数 rec に格納する．

3. 6. 2　Write 命令

write 命令はデータストリームに通信文を書く．

〔構文〕

```
<WriteStatement>
  : : = 'swrite'
        '('
        ["each"]
        <DataStreamName>
        [<EventName>]
        {<Expression>}
        ')'
```

〔例 1〕　swrite(c 'invest' 860228 10000 "jun kato")

この例では，通信文はデータストリーム c に書かれる．通信文の型は invest で，その本体は 860228，10000，"jun kato"という値の列である．

〔例 2〕　swrite(exception_report "over draw")

報告書を書く例である．ここでは exception_report というデータストリームに "over draw"という通信文を書く．通信文の送り手のプロセス識別子は，通信文に暗示的に含まれる．

〔例 3〕　swrite("each" timer 'alarm')

通信文の型alarmをデータストリーム・クラスtimerに属するすべてのデータストリームに書く．

3.6.3 GetStateVector命令

GetStateVector命令は，他のプロセスの状態を参照する．宛先のプロセスの内部状態の複写を要求するときにGetStateVector命令を使用する．

〔構文〕

```
〈GetStateVectorStatement〉
  : : = 'getsv'
        '('
        〈StateVectorName〉
        〈Destination〉
        〈VariableName〉
        ')'
〈Destination〉
  : : = "next" | 〈ProcessIdentifier〉
```

〔例1〕　getsv (cv "jun kato" svrec)

この例の実行結果は，"jun kato"を識別子にするcustomer_1プロセス・クラスのプロセスから状態ベクトル結合cvを経由して，その状態ベクトルの複写が変数svrecに代入される．この命令は，cv状態ベクトル宣言のアクセス方法でORDER宣言をもたない状態ベクトルに限り有効である．

〔例2〕

状態ベクトル結合cvsに属する状態ベクトルのすべてを逐次参照することができる．この結果をsvrecに格納するときには以下のように記述すればよい．

getsv (cvs "next" svrec)

すべてのプロセスを参照し終わったとき，svrecはnilになる．この命令は，cvs状態ベクトル宣言のアクセス方法で，ORDER宣言をもつ状態ベクトルに限り有効である．

3.6.4 Wait命令

プロセス間の同期をとるときにwait命令を使う．

〔構文〕

```
〈WaitStatement〉
  : : = 'wait'
        '('
        ["all"]
        〈DataStreamName〉
        [〈ProcessIdentifier〉]
        〈EventName〉
        ')'
```

〔例1〕　wait ("all" ta 'tgm')

この例の命令を実行すると，データストリーム・クラスtaに属するデータストリー

ムから tgm という型の通信文がなくなるまでプロセスの実行を停止する．wait 命令は write 命令と対で使う．たとえば，

swrite ("each" ..) ; wait ("all" ..)

のように記述する．

〔例 2〕　wait (ta "jun kato" 'tgm')

この命令は

swrite (ta 'tgm' 871031 "jun kato") ; wait (ta "jun kato" 'tgm')

という対で記述される．これを実行すると，"jun kato"を識別子するデータストリーム・クラス ta で結合されたプロセスが，このデータストリームから tgm という通信文を読むまで，プロセスの実行が停止される．

4. JSL の 実 行

JSL の実行は JSD 仕様の解釈でもあるので，はじめに JSD の特徴を述べ，それから JSD 仕様とその解釈を述べることにする．次に JSL 解釈系の概要を説明し，終りに簡単な例を用い，JSL で記述した仕様とその実行結果を示す．

4.1 JSD の 特 徴

JSD は，「手法」とそれを使う「手順」からなる．JSD の「手順」は，仕様作成段階とその実現段階に分かれる．仕様作成段階では，実際の実行環境や実現のための資源に関する決定をせず，実現段階で実際の実行環境や実現のための資源に関するすべてを決定する．

ソフトウェア・システムの開発において，一般に事実または事実に近い具体的な事柄の決定は安定しているが，事実から遠い（抽象的な事柄の）決定ほど不安定であると言える．さらに，やさしい決定を先にし，むずかしい決定ほど後にするという順序もソフトウェア・システム開発では一般的である．JSD の「手順」は，この考えを基本にしている．すなわち，JSD は事実から遠い決定を後にする（言い換えれば，事実に近い順に決定する）ことや，やさしい問題から先に片付け複雑でむずかしい問題を後回しにするための「手順」を示している．

仕様作成段階では，事実の記述からはじめ実世界のモデルを作る．つぎに，モデルを拡張し機能要求を仕様化する．実現段階では仕様を再構成（re-package）して，実現環境にあった最終コードにする．この再構成で使う「手法」は，変換とスケジューラの導入である．事実というよりも，開発者が考案する要素が大半を占めるスケジューリング・アルゴリズムは，実行環境に依存した決定を含むため，実現段階で追加することになる．

JSD の「手法」には，階層構造を持たない「並行処理」の概念と，「やさしいわかりやすいプログラム」を「要求される速さと効率を満たすプログラム」に変換する「プログラム変換」手法とが背景にある．はじめに仕様として，並列に動くプログラムを記述する．JSD によって「プログラム変換」を使い，効率のよいシステムに仕上げる．「手法」と「手順」とは，密接な関連があるので「手順」と抱き合わせで JSD の概略を述べる．

JSD は並列に稼働しているプロセスを明示的に記述する仕様作成段階と，その仕様

を実際の実行環境に適したプログラムに変換する実現段階に分かれる．並列に動くプロセスを記述する利点は，動的な側面を仕様に反映できることと，複雑な階層構造の導入が避けられることである．実現段階で，動的側面をスケジューラの導入で解決し，主プログラムとサブルーチンからなる階層構造にする．主プログラムがスケジューラである．また，実現段階で，仕様中のプロセスの局所変数をデータベース(ファイル)のレコードとしてプロセスから分離する．

原理的には，JSDの仕様は直接実行可能であり，JSDを適用したラピッド・プロトタイピングが採用できる．さらに，仕様を直接実行し確認した後に一連の変換を施して実際の実行環境に適したプログラムに改良できる．これは，操作的アプローチ(operational approach)[5]によるシステム開発も可能であることを意味する．JSDを操作的アプローチの一つに挙げるのもそのためである．

4.2 JSDの仕様とその解釈

JSDの仕様は，主として次の文書からなる．

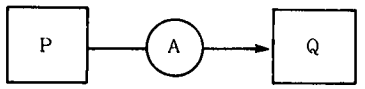
1) 逐次プロセスからなる分散通信網による記述．この通信網をJSDではSSD(System Specification Diagram)といい，表記法が定まっている．

2) それぞれの逐次プロセスは，図(またはテキスト)表現のジャクソン木(Jackson Tree)で定義される．

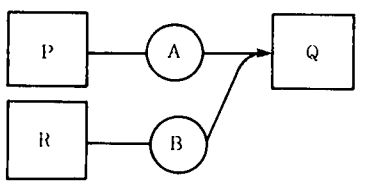
3) 行動の定義……ここで，行動は，外界における事象（イベント）を示す．モデル・プロセスは，行動の発生時間の順序をジャクソン木で定義した構造を骨格としている．プロセスは，事象によって起動され，次の事象の発生を待つ所で停止する．また，システムの外界と接しているプロセスだけでなく，システム内部にあるプロセスも同様に事象によって起動する．以降システム内で発生する事象も含めることにする．事象は，その存在の他に，それに付随する属性をもつ．この事象の属性によって，プロセスがもつデータが定義される．プロセス間通信の場合の事象の発生を通信文の発生と解釈できるので，事象ごとに対応する通信文の型があると考えてよい．

プロセス間の通信には，2通りの方式がある．①データストリーム結合による通信と，②状態ベクトル結合による通信である．いずれも非同期通信である．通信文が通過する通信路は，無限長の通信文のバッファであり，仕様においては静的に定義される．図2の例を用いプロセス間通信について述べる．

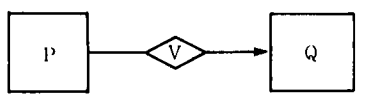
図2(a)と(b)は，データストリーム結合による通信である．これの解釈を図3を用い述べる．図3(a)は，図2(a)と対応し，プロセス間の通信路が一つである結合を示している．通信路Aは無制限に通信文をもつことができるので，プロセスPは通信文を送るために停止することはない．図2(b)はラフマージといい，複数のプロセスからの通信文が一つの通信路にまとめられ，一つのプロセスに送られる結合を示す．図3(b)はこれの解釈であり，通信路AとBのうちのどちらの通信文を優先的にプロセスQへ送るかは非決定的であり，先に到着した通信文を優先的にプロセスQへ送るという原則があるにすぎない．図2(c)は状態ベクトル結合による通信である．これは，相手のプロセスの内部変数の値を参照する通信である．プロセスが互いに状態ベクトル結合によって通信しあうときは，①参照するプロ



(a) データストリーム結合



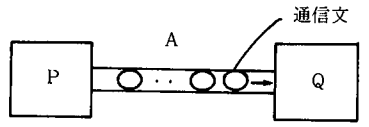
(b) ラフマージのある結合



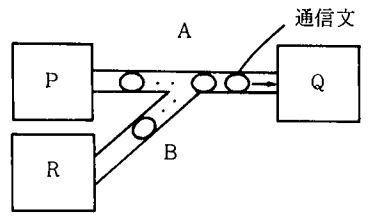
(c) 状態ベクトル結合

図2 システム仕様図 (SSD) による表現

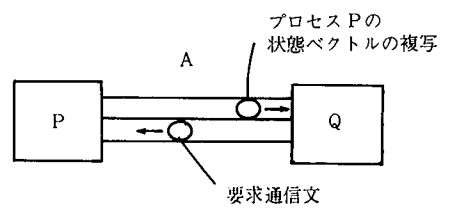
Fig. 2 Examples of SSD



(a) データストリーム結合



(b) ラフマージのある結合



(c) 状態ベクトル結合

図3 各種結合 (図2) の通信路による表現

Fig. 3 Representation by communication channels in cases of various connections (ref. Fig. 2)

セスが参照されるプロセスの状態ベクトルの複写を要求する通信文を送る，②要求通信文を受け取った宛先のプロセスは，その内部状態の複写を送り返す，③①のステップに戻る，という順に通信が行われる．このとき，宛先プロセスの内部状態は変化しない．図2(c)の場合はプロセスQが，プロセスPの内部変数の値を送ることを要求し，プロセスPがその内部変数の値を回答する通信である．なお，この場合はプロセスPの内部状態は変化していない．図3(c)はその一つの解釈を示している．

システム内に通信文があるとき，それを受け取るプロセスは必ず存在する．したがって，システム内に通信文がある限りJSD仕様は実行されている．これは，仕様においてはプロセスの生成や消滅を意識する必要はないことを意味する．

4.3 JSL直接実行系

JSL直接実行系は，JSLを解釈し実行する処理系である．この直接実行系では，2種類の通信方法（つまり一つはデータストリーム結合を経由するもの，他の一つは状態ベクトル結合を経由するもの）をもつオブジェクトとして扱う．しかも，これらのプロセスがあたかも並行に実行されているかのように，JSL直接実行系はシミュレートする．

4.3.1 JSL 直接実行系の構成

JSL 直接実行系は，JSL パーサと JSL 実行系からなる．その構成を図 4 に示す．

4.3.2 JSL パ ー サ

JSL パーサは，JSL テキストの構文解析と静的な意味解析を行い以下を生成する．

1) 事象テーブル……事象宣言を表現しており，通信文パーサによる通信文の解析，宛先プロセスの決定，実行系内部の通信文の作成に必要な情報をもつ．

2) プロセス間結合テーブル……データストリーム宣言と状態ベクトル宣言を表現しており，通信文パーサがどの通信路に通信文を送るかを決定するための情報をもつ．

3) 補助テーブル……状態ベクトルのアクセス法，データストリームのラフマージについての情報をもつ．

4) プロセス宣言テーブル……プロセス宣言の表現で，局所変数，構造部の解析木，命令や条件式の解析木をもつ．

4.3.3 JSL 実 行 系

図 4 で示すように，JSL 実行系は，通信文パーサ，プロセス・インタプリタ，スケジューラ，ユーザ・インタフェース，ユーティリティからなり，LISP 専用機ではそれぞれオブジェクト指向言語であるフレーバ (Flavors) のクラスで実現されている．通信文パーサとプロセス・インタプリタは，JSL パーサによって生成された諸テーブルを参照する．また，JSL 実行系ではプロセスとバッファはフレーバのクラスで表現し，通信をオブジェクト間のメッセージ通信で実行し，構成要素間のデータ交換をすべてフレーバ間のメッセージ通信に統一している．

以下に JSL 実行系の構成要素の概要を述べる．

1) 通信文パーサ……事象を JSL 実行系内部の通信文に変換し，これを宛先のプロセスのバッファに送る．3 種類の通信文があり，一つはデータストリーム結合を経由する通信文，残りの二つは状態ベクトル結合を経由する通信文である．さら

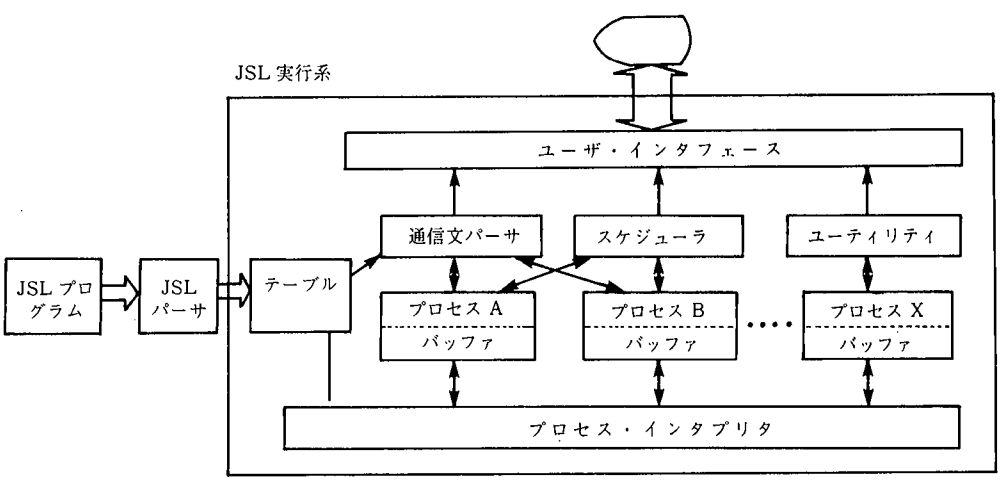


図 4　直接実行系の概要

Fig. 4　Overview of JSL interpreter

に，後者は，宛先のプロセスの状態によって，次の2通りの処理，①宛先のプロセスが存在すれば，そのプロセスに結合されているバッファの待ち行列に通信文を加える，②宛先のプロセスがまだ生成されていないときは，そのプロセスとプロセスに付随するバッファを生成する，に分かれる．なお，生成されたプロセスの内部状態は初期化される．逆に，プロセスから外部へ通信文を送るときは，実行系内部の通信文をユーザ・インタフェースの書式に変換して，ユーザ・インタフェースに送る機能をもつ．

2) プロセス・インタプリタ……プロセスの初期化・実行・停止をプロセス宣言テーブル中の解析木（構造・命令・条件式）を参照して行い内部状態を更新する．通信文パーサによって，プロセスが生成されたとき，プロセス・インタプリタはプロセスを最初の Read 命令か，GetStateVector 命令まで実行し停止する．スケジューラによってプロセスが起動されるときは，プロセス・インタプリタは，①停止状態から判断し起動したのがデータストリームか，状態ベクトルのいずれかの通信文であったならば，対応するバッファから通信文を取り出し，通信文のトレースに追加する，②それ以前の局所変数とテキスト・ポインタを状態ベクトルのトレースに追加する，③その後でテキスト・ポインタを進めながら命令を実行し，プロセスにある局所変数を更新する，④プロセスを停止する命令（sread, getsv, wait 命令）に達したとき，テキスト・ポインタの進行を止め，停止した命令の種類と次に受け入れる通信文の型（データストリーム名，または状態ベクトル結合名）を格納し制御を放棄する，という順序で処理する．

3) スケジューラ……ランダムにプロセスを走査し，実行可能なプロセスを起動する．すべてのプロセスが通信文待ちで停止するまで，スケジューラは起動可能なプロセスを起動し続ける．起動可能なプロセスがなくなったとき，スケジューラは制御をユーザ・インタフェースに戻す．ただし，途中でスケジューラからユーザ・インタフェースに制御を戻すことができるように，常に監視しユーザ・インタフェースが実行停止指令を受け取ったときに制御を戻す．プロセスの起動条件を以下に述べる．

① 停止条件を満たすデータストリームのバッファに通信文がある．
② 状態ベクトル参照への回答通信文がバッファアにある．
③ 状態ベクトル参照の要求通信文がバッファにある．

プロセス起動の方法には2通りある．一つは①と②の場合にバッファにある通信文をプロセスに与えて実行を再開する方法で，もう一つは③の場合に内部状態の複写を作り返答の通信文を送る方法である．

スケジューラは，また次に示す特殊な場合も考慮している．

① 状態ベクトルの参照による通信しかしないプロセスの起動と停止．
② システム外のプロセスの状態ベクトルを参照するときに，ユーザ・インタフェースに状態ベクトルの複写を要求すること．

4) ユーザ・インタフェース……主な機能を以下に列挙する．

① システムの外で発生している事象を受け取り，通信文パーサへ送る．
② 「システムの外にあるプロセスの状態ベクトルの値を要求する通信文」を通信

文パーサから受け取ると，その状態ベクトルの値を送る手続きを呼び出す．そして，この手続きの実行結果を回答として，通信文に編集し通信文パーサへ送り返す．

③　スケジューラの停止指令などの利用者からの指令を解析し実行する．

5)　ユーティリティ……プロセスの状態ベクトルや通信文の追跡，および現在の状態ベクトルの値等の表示を行う．

4.3.4 簡単な例題

JSLとその直接実行系の理解を助けるために，次に示す簡単な例題をJackson[2]から引用し，JSLで記述した仕様と，その実行結果を示す．例題はSimple Bank Systemである．

「ある銀行は次のような預金を扱っている．顧客は，一人一口座しか持てない．顧客は，まず預金口座を開設（invest）する．それ以後は，口座を閉鎖（terminate）するまで，入金（pay in）と出金（withdraw）を繰り返す．非現実的かもしれないが，過払いを許しており，過払い分の利息は取らない」

なお，機能としては，以下の要求を満たさなければならない．

1)　出金で過払いが発生するつど，例外報告書を出力する．

2)　当銀行のマネージャは，いつでも特定の顧客の残高を参照できる．

図5は，このシステムの仕様の記述例で，図6は，この仕様の実行結果である．この例では，customer_1プロセス・クラスの具体例として各々識別子"jun kato"と"aki sato"を持つ二つのプロセスと，enquiry_fnプロセス・クラスの具体例としての1プロセス（識別子"enquiry_fn"）がある．"jun kato"は，通信文の属性customer_nameの値である．"jun kato"の行動の順序を以下に示す（図6）．

1)　口座開設……(a)

2)　入金　……(d)

3)　引出し　……(j)

"aki sato"の行動の順序を以下に示す．

1)　口座開設……(b)

2)　引出し　……(e)

3)　引出し　……(i)

"aki sato"の行動3)は過払いを発生するので，プロセスは例外報告書を出力する(l)．また，プロセス"jun kato"の状態ベクトル，プロセス"jun kato"のイベント，プロセス"aki sato"の状態ベクトルの各トレースが，それぞれ(x), (y), (z)に示されている．なお，(c), (g), (k)はユーザインターフェイスへの指令（end）であり，通信文パーサへの入力通信文の送信とスケジューラの起動を指示する．

このほか，"data_stream event_name attribute_values ?"は，通信文の入力を促すプロンプトである．

5. おわりに

JSDの仕様の直接実行については昨年報告した．本稿は，COMPSAC '87で発表した論文[10]とパーソナ・ルコンピュータ（DS 7）にJSLサブセットの直接実行系を移植

```
SYSTEM : simplebank
 EVENT
   invest customer_name (customer_name amount)
   pay_in customer_name (customer_name amount)
   withdraw customer_name (customer_name amount)
   terminate customer_name (customer_name reason)
   enquiry (id)
 EVENT END
 DATASTREAM
   c (invest pay_in withdraw terminate)
     : # > _ > customer_1;
   exceptionreport ()
     : customer_1 > _ > #;
   enquiryinput (enquiry)
     : # > _ >  enquiry_fn;
   enquiryreport ()
     : enquiry_fn > _ > #;
 DATASTREAM END
 STATEVECTOR
    cv   (balance) : customer_1 >> _  > enquiry_fn;
 STATEVECTOR END
 PROCESS
   customer_1
   STRUCTURE
   customer_1 SEQ
     DO 1 DO 2
     invest SEQ DO 3 invest END
     DO 2
     customer_body ITR WHILE c1
       movement SEL c2
         pay_in SEQ DO 4 DO 2 pay_in END
       movement ALT c3
         withdraw SEQ
            DO 5
            p_exeptrep  SEL c4
              DO 6
            p_exeptrep  END
            DO 2
         withdraw END
       movement END
     customer_body END
```

図5　JSLの記述例(その1)

```
    terminate SEQ    ;
    terminate END
  customer_1 END
  OPERATIONS
   1: balance := 0
   2: sread(c)
   3: balance := .amount
   4: balance := balance + .amount
   5: balance := balance - .amount
   6: swrite(exceptionreport "overdrawn" .name  balance)
  CONDITIONS
   c1: (withdraw or pay_in)
   c2: pay_in
   c3: withdraw
   c4: balance < 0
  customer_1 END

  enquiry_fn
  STRUCTURE
  enquiry_fn  SEQ
    DO 1
    enquiry_fn_body  ITR WHILE  c1
      enquiry SEQ
        DO 2 DO 3 DO 1
      enquiry END
    enquiry_fn_body END
  enquiry_fn END
 OPERATIONS
 1: sread(enquiryinput)
 2: getsv(cv .id svrec)
 3: swrite(enquiryreport .id "balance is " svrec.balance)
 CONDITIONS
 c1: true
 enquiry_fn END
 PROCESS END
SYSTEM END
```

図5　JSLの記述例(その2)

Fig. 5　Example of a specification in JSL (simple bank system)

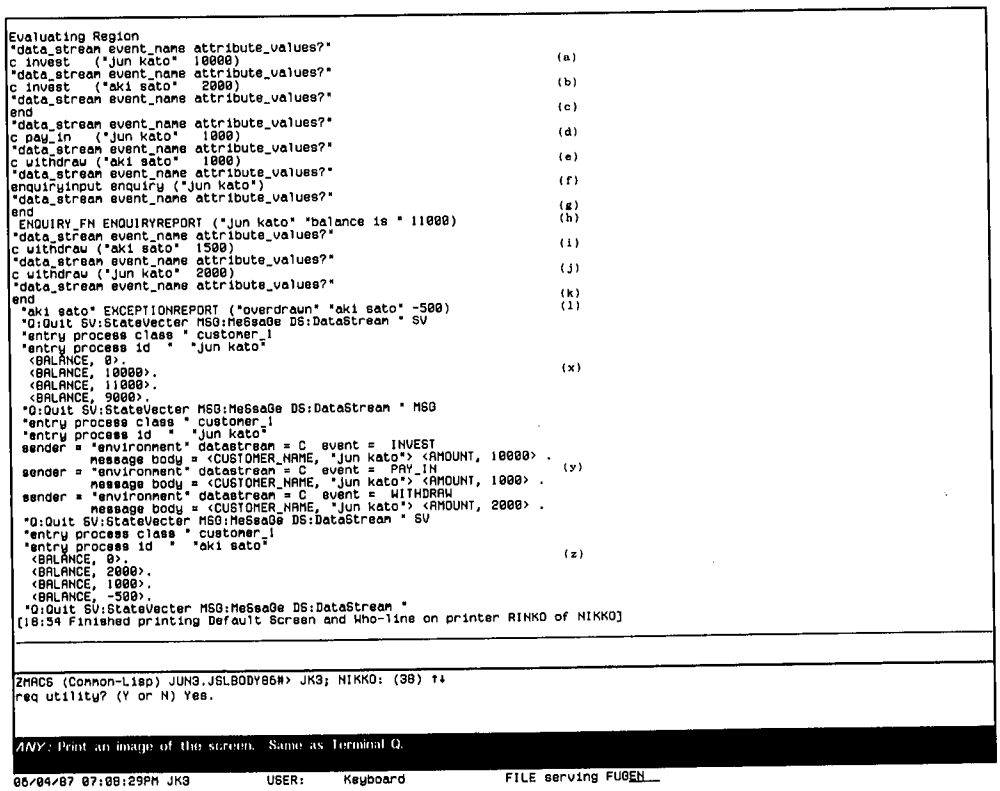


図6 図5の実行例

Fig. 6 Output of simple bank system (ref. Fig. 5)

した報告[9]の一部に加筆・訂正するという形で，昨年の報告以降の進展をまとめたものである．

COMPSAC '87 の論文については，Michael Jackson，Ashly McNeile の両氏から詳細にわたり周到なコメントをいただいた．Jackson 氏のコメントによって，本稿の 3.6.1 項，3.6.4 項を改訂した．また，McNeile 氏のコメントによって 3.5 節を改訂した．

本システムは，システム開発の効果的な道具にするために，JSE の部分系として開発してきた．この開発の成果および，得られた知見を以下に要約する．

1) JSD の仕様記述に言語 JSL を導入した．JSL は，テキスト形式の JSD 仕様に，直接実行に必要な情報を加えたものである．
2) 通信文の通信機構にオブジェクト指向プログラミングの考えを使った．
3) JSD 仕様の実行系を開発し，その直接実行が可能であることを示した．
4) JSL 直接実行系は，小規模なシステムであればパーソナル・コンピュータでも実現できる．

なお，この試行は，JSE の改良と，より良いユーザ・インタフェースを実現するために現在も継続している．さらに，JSD 開発過程の実現段階の支援に使用するために，JSL の機能の拡張に取り組んでいる．また，残りの部分は現在開発中である．

JSEは，KS-301（LISP専用機）で実現されており，Common Lispで記述されている．また，JSLサブセットの直接実行系と，図形エディタの一部はDS 7上でC言語を使い実現されている．

最後に，COMPSAC' 87で発表した論文の初期原稿を読んで頂いた多くの人達に感謝する．とくに，JacksonとMcNeileの両氏および大阪大学基礎工学部鳥居宏次教授，日本ユニバックの山崎利治，澤田晟司の各氏に感謝する．また，COMPSAC '87の論文の共著者である森澤好臣氏，およびJSL直接実行系のDS 7への移植における共同開発者の山住厳氏にも感謝の意を表したい．

参考文献

[1] C. V. Ramamoorthy, et al., Software Engineering: Problems and Perspectives, IEEE COMPUTER Magazine, Oct. 1984.
[2] M. A. Jackson, System Development, Prentice Hall, 1982.
[3] J. R. Cameron, An Overview of JSD, IEEE Trans. of SE Vol. SE-12 No. 2, Feb. 1986.
[4] C. Potts, A. Bartlett, B. Cherrie, R. Moclean, Discrete Event Simulation as a Mean of Validating JSD Specifications, Proc. of 8th ICSE, Aug. 1985.
[5] P. Zave, An Operational versus the Conventional Approach To Software Development, CACM, Vol. 27, No. 2 Feb. 1984.
[6] 澤田，ジャクソン法のための木構造エディタの試作，マイクロコンピュータ研究会報告，Vol. 87 No. 45（1987年6月30日），情報処理学会．
[7] A. I. Wasserman, P. A. Picher, A Graphical Extensible Integrated Environment for Software Development, SIGPLAN Notices Vol. 22 No. 1, Jan. 1987.
[8] 加藤，JSD（Jackson System Development）仕様の実行系の試作，「プロトタイピングと要求仕様」シンポジウム（1986年4月16日），情報処理学会．
[9] 山住，加藤，JSL（ジャクソンシステム開発法言語）インタプリタの開発，ソフトウェア工学研究会報告，Vol. 87，No. 58（1987年9月9日），情報処理学会．
[10] J. Kato, Y. Morisawa, Direct Execution OF A JSD Specification, Proc. of COMPSAC87, Oct. 1987.

執筆者紹介　加　藤　潤　三（Junzo Kato）
昭和46年岡山大学理学部物理学科卒業．同年4月日本ユニバック㈱入社．現在，システム本部生産技術一部所属．

論文 24時間運転

Nonstop Real-time System for Financial Industries

村田豊彦

要約 金融機関においては，顧客サービスの充実を目的にオンラインの平日時間延長や休業日運転が開始されている．

筆者が参加した金融機関のシステム開発では，設計段階からオンラインの長時間延長を意識してシステム構築が行われている．同システムで採用された諸技術は，単なる時間延長対応にとどまらず究極的には24時間運転対応に結びついていくと考える．

同開発では，24時間運転のためのシステムの目標として次の4点を掲げた．

1) オンライン終了という境界を意識しないシステム
2) 静的状態（データベースの更新の停止）を要求しないシステム
3) 処理量の平準化
4) 処理の効率化

本稿では，24時間運転へのアプローチという観点から，問題の把握とその解決事例の紹介を行う．

Abstract Recently an EDP facility in the financial industries is faced with ever increasing, huge volume of online transactions, and the cutoff time of its operation is correspondingly extended. It is not rare that the operation is forced to continue until next morning. Thus its full time operation through 24 hours emerges as a pressing need.

The author had an experience to develop a nonstop real-time system of a customer in the financial industries.

The paper will make clear the problems in developing the system and show how to solve them.

We set the system goals as follows :

1) System can be developed without considering the cutoff time of its online operation.
2) Its operation does not enforce the static files, or in other words the inhibitance of file updations should be minimized.
3) Its work load should be well ballanced by distributing the processes of transactions.
4) Its performance should be improved at its best.

1. はじめに

金融機関においては，昭和61年7月末よりオンラインの時間延長，続いて8月第2土曜日からはオンライン休日稼働が開始された．

時間延長や土休運転は，とくにセンター部門に対し時間的余裕を少くする．平日の時間延長によってセンター部門は，オンライン後の各種定常作業やその他必要業務をより短時間で行うことを強いられる．

また，休日稼働は連続した余裕時間帯の減少をもたらし，処理時間の長い，いわゆるロング・ランの実行日がより限られた範囲から選択せざるを得なくなってきた．

各金融機関においては，オンライン処理後の時間短縮などの対策や運用面での対応を実施している．

しかし，オンライン処理時間帯拡大化傾向を考えると，応急的対応ではなく抜本的な対策，つまり24時間運転対応を講じることが急務となるであろう．

本稿では，24時間運転を実現する基礎技術としてオンライン終了後の処理や連続した余裕時間帯に実行していたロング・ランをオンライン中に実行可能とするシステムを取り上げ紹介する．

2. 現　　状

昭和61年7月のオンライン時間延長実施直前での日本ユニバックの金融機関ユーザ調査から，各金融機関のコンピュータ使用終了時刻について表1の結果が得られた．

時間延長が実施されている現在では，コンピュータ使用終了時刻はさらに遅くなっていることであろう．

表1からピーク日の平均終了時刻は平常日に比べ，3時間30分も遅くなっているのが目につく．オンライン終了そのものが遅くなることや，オンライン終了後に大量のセンターカット・データを処理しなければならないことなどに起因すると考えられる．このことからピーク日については，単に業後作業をオンライン中に組み込むだけでなく，処理量の多さという点から，他の余裕のある日に振り分けて平準化を図る必要性を感じる．

オンライン時間帯への作業の取り込みや平準化が推進されてくると，処理の効率がクローズ・アップされよう．

表1　金融機関ユーザにおける業後作業終了時刻

Table 1　End time of computer use in banking industries

ユーザ名	平日終了時刻	ピーク日終了時刻
××銀行	22：00	4：00
××銀行	23：00	0：00
××銀行	22：00	7：00
××銀行	23：30	1：00
××銀行	1：00	3：30
××銀行	22：30	1：00
××銀行	21：30	2：00
××信用金庫	21：00	22：00
××信用金庫	22：15	0：00
××信用金庫	1：00	3：00
××信用金庫	21：30	23：00
××信用金庫	21：00	21：30
××信用金庫	20：30	23：30
××信用金庫	1：00	4：00
平均（30社）	23：00	2：30

3. 問　題　点

24時間オンライン運転が実施されると，従来オンライン処理終了をもって開始した作業や連続した余裕時間帯で行っていた作業の取り扱いが問題になる．

これらの作業を整理すると表2のようになる．

表2　処理時間帯が問題となる作業

Table 2　Jobs not runnable during real-time operations

	オンライン終了時あるいはオンライン終了後作業	連続した余裕時間帯で実施していた作業
システム面	1）ファイル保存	2）ファイル・メンテナンス 3）システム開発 4）システム変更 5）ハードウェア機器の保守
業務処理面	6）精査 7）日次バッチ 8）センターカット処理 9）カウンタとデータベースの突き合わせ 10）月次バッチ	11）金利変更 12）元加処理

以下では，それらをより詳しく見てゆこう．

1)　ファイル保存……ファイル障害に対応するため，データベースをテープに保存することである．通常，オンライン処理あるいは業後処理の終了後，データベースの更新を禁止した静的な状態でのデータベースを保存している．24時間運転になるとファイル保存を開始する時機がなくなること，および静的データベースの状態が存在しないという2点の問題が発生する．

2)　ファイル・メンテナンス……24時間運転になると，連続した余裕時間帯での静的なデータベースを前提にしたファイル・メンテナンスは不可能になる．

3)　システム開発……本番機が24時間使用されていると，本番機およびその環境を使用したシステム開発ができない．

4)　システム変更……OSやその他の基本ソフトウェアなどを切り替える時機がなくなってくる．

5)　ハードウェア機器の保守……24時間にわたって機器が使用されていると，保守を実施する時間がとれない．

6)　精　　査……金融機関においては，各営業店ごとにオンライン処理が終了すると，当日の処理に誤りがなかったか，あるいは全体の計数はどうなっているかを知るために締め操作を行う．従来はトランザクション処理終了をもって実施しているが，24時間オンラインの場合はその時機がない．

7)　日次バッチ……金融機関の日次バッチは，1日のトランザクション処理分の取引明細・日報類の作成が主な作業で，オンライン終了後に開始するのが通常である．オンライン処理トランザクション量が多い日や，業後処理そのものが多い日の日次バッチ終了時刻は非常に遅くなっている．24時間運転ではオンライン終了という時機がないことと，特定日の処理量が多いという二つの問題点がある．

8) センターカット処理……センターカット処理とは，センター内で発生するトランザクションを一括して処理することをいう．金融機関でのセンターカット処理の例としては，自動振替契約にもとずく各種入出金処理や，貸出し金の回収処理などがある．日によって対象となるセンターカット処理の種類は異なるが，数十万件以上のデータを処理しなければならない日もある．通常，オンライン終了後の作業として実施しているところが多いが，24時間オンラインになるとオンライン中に実施せざるを得なくなる．その場合，処理効率やファイルのデッドロックなどが問題となる．

9) カウンタとデータベースの突き合わせ……トランザクション処理が，正常に行われたか否かを検証するために使用されるデータとしてカウンタ値がある．一般にオンライン終了後データベース・レコードの集計値とカウンタ値が合致することを確認する．24時間オンラインの場合，静的なファイル状態が存在しないためデータベース・レコードの集計値とカウンタ値は一致しなくなってしまう．

10) 月次バッチ……月末時点での静的なデータベースをもとに各種月次統計の作表などを行う．月末当日から開始し，月初数日間にわたって作業を行う．24時間運転では，月末時点の静的なデータベース採取が問題になる．

11) 金利変更……金融機関においては，契約時点の利率や預け入れの日数から利息がどの程度かを予想し(予想積数，積数＝残高×日数)，これをデータベース上に保有するシステムがある．この場合，途中で利率が変更されると，予想積数を計算し直す作業が発生する．従来のシステムでは，静的なデータベースをもとにバッチ処理によって対応することが多く，長時間の余裕時間帯がないと作業が実施できない．

12) 元加処理……金融機関では元加処理といって，半期に1回ずつ預金利息を計算し，利息分も預金額に組み入れるという作業がある．これも金利変更と同様に静的なデータベースをもとにバッチ処理で対応することが多い．したがって，長時間の余裕時間帯がないと実施が困難である．

表2では24時間運転の場合に，処理時間帯が問題となる作業を金融機関の具体的な業務処理も含めて整理した．

しかしながら，問題点はこれだけではなく，以下であげるようなシステム無停止そのものがもたらす問題がある．

1) 不要ファイルの蓄積……初期化ブートができないため，利用者が短期的に割り当てたファイルや管理するファイルが，不要であるにもかかわらず蓄積されてしまう．

2) 常駐ランの連続走行……常駐ランが無停止状態で走行すると，OSや基本ソフトウェアのラン管理上の制限値（走行時間や作成するログ件数など）を越えてしまい，ランは停止する．

3) メモリ・テーブルの初期化処理……メモリ・テーブルの中には1日ごとの通番などを管理する目的で使用されるものもあり，通常これらは初期設定時に初期化処理される．システムが無停止のため常駐ランやオンライン・トランザクションが，これらのメモリ・テーブルを使用し続けていると，初期化処理を行うことは

不可能である．

4) CD（現金引出機）・ATM（自動窓口業務機械）など自動機器の管理・監視……金融機関においては，CD や ATM などの自動機器が 24 時間稼働することになるのだが，深夜時間帯の管理・監視が問題となる．自動機器が設置されているすべての場所に，24 時間体制で人員を配置しておくわけにはいかないであろう．

4. 問題点の分析と実現にあたっての考え方

4.1 問題点の分析

3 章であげた問題点を分析すると，ほとんどは次にあげる要因から発生している．

- 要因 1 ……オンライン終了境界の消滅
- 要因 2 ……処理量の特定日への集中
- 要因 3 ……ファイルの静的な状態を前提にしたシステム構造
- 要因 4 ……連続した余裕時間帯での作業実施を前提にしたシステム
- 要因 5 ……システム無停止そのもの

表 3 に問題点と要因との関係を整理する．

4.2 実現にあたっての考え方

24 時間運転を実現するためには，従来オンライン終了後や連続した余裕時間帯で行っていた作業を，オンライン中に実行可能とするシステムの構築が求められる．前述

表 3 問題点と要因との関係

Table 3 Factor analysis of problems

システム無停止そのもの
連続した余裕時間帯での作業実施を前提にしたシステム
静的な状態を前提にしたシステム構造
処理量の特定日への集中
オンライン終了境界の消滅

問題点	要因 1	要因 2	要因 3	要因 4	要因 5
ファイル保存	○		○		
ファイル・メンテナンス			○	○	
システム開発				○	
システム変更				○	
ハードウェア機器の保守				○	
精査	○				
日次バッチ	○				
センターカット処理	○	○			
カウンタとデータベースの突き合わせ	○		○		
月次バッチ	○		○		
金利変更			○	○	
元加処理			○	○	
不要ファイルの蓄積					○
常駐ランの連続走行					○
メモリ・テーブルの初期化処理					○
CD・ATMなど自動機器の管理・監視					○

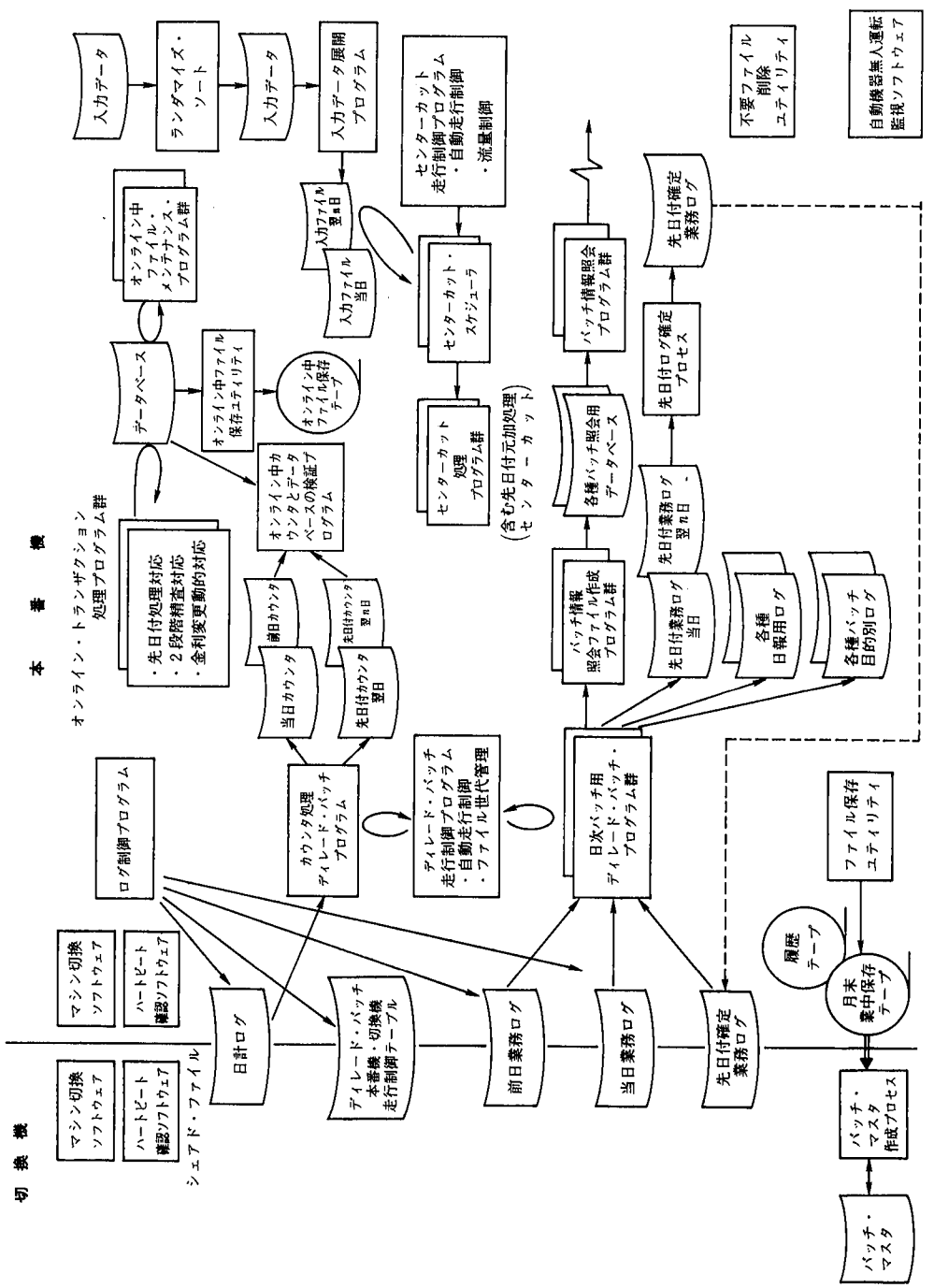


図1　ソフトウェア構成

Fig. 1　Nonstop real-time system for banks

の問題点と要因との関係から，具現化にあたってのシステムの目標を以下のように設定する．

- オンライン終了境界を意識しないシステム
- 処理量の平準化
- ファイルの静的状態を要求しないシステム
- 処理の効率化

処理の効率化は，オンライン時間帯への各種作業の取り込みや平準化が推進されることによる負荷増大への対応で重要なテーマである．

なお，システム無停止そのものがもたらす問題については，運用上の工夫でカバーすることを基本とする．

5. システムの実現

前章の方針にもとずき，具体的にシステムを実現してゆく．本章では，システムの実現について述べる．

5.1 ソフトウェアの構成

カウンタ処理や日次バッチ処理を行うディレード・バッチ・プログラム群（トランザクション処理で発生したデータをバッチ処理するプログラムをいう．通常ディレード・バッチ・プログラムは，トランザクション処理プログラムによってスケジュールされる）とセンターカット処理プログラム群を中心としたソフトウェア構成を図1に示す．

5.2 オンライン終了境界を意識しないシステム

5.2.1 日次バッチ

オンライン終了時刻に依存しない日次バッチ・システムの運行を可能にするため，バッチ処理の対象時間帯を

18時 ～ 翌18時

にする．

この前提を置くことにより，業後バッチはオンライン処理とは関係なく，18時以降の任意の時点から開始可能になる．

なお，同一勘定日時間帯は

0時 ～ 24時

にする．

オンラインと並行して日次バッチを行うために，図2に示すように業務ログ・ファイルを2サイクルもつ．業務ログ・ファイルの切り替えは24時に行う．ディレード・バッチは，前日業務ログ β 分（18時～24時）と当日業務ログ α 分（0時～18時）を対象データとして，日次バッチ用各種ファイルを作成する．ディレード・バッチは，オンラインにわずかづつ遅れながら並行処理しているので，18時過ぎには日次バッチ処理を開始できる．

次に，日次バッチ処理自体の削減を目的としたバッチ情報照会システムの制御を図3に示す．

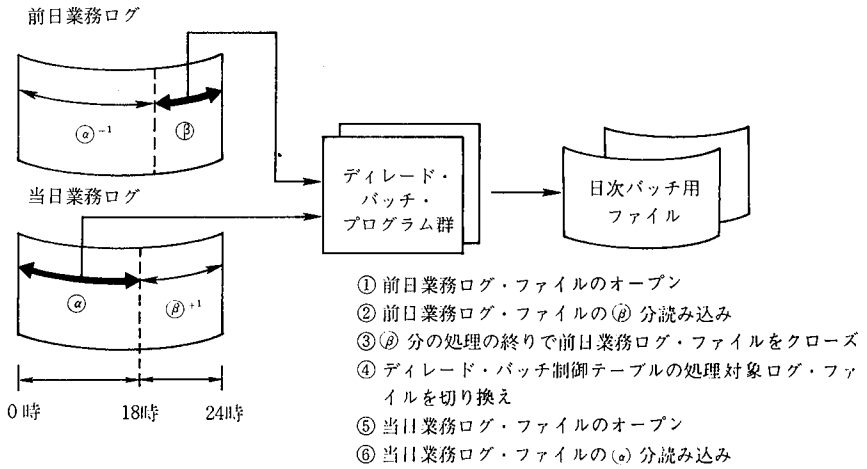
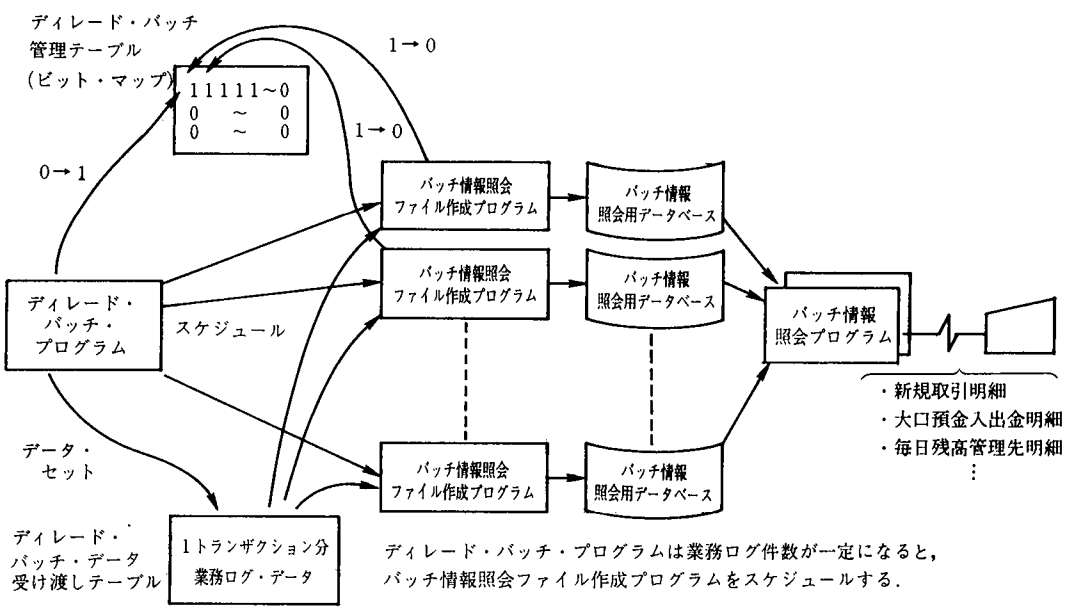


図2　オンライン終了時刻に依存しないバッチ・システム

Fig. 2　Delayed-batch system for nonstop real-time system



① ディレード・バッチ・プログラムは，1トランザクション分の業務ログ・データをディレード・バッチ受け渡しテーブルにセットし，ディレード・バッチ管理テーブル（ディレード・バッチ・プログラムとバッチ情報照会ファイル作成プログラムの同期とりに使用する）のビット・マップをオンにする．ビット・マップの1ビットが，被スケジュール側であるバッチ情報照会ファイル作成プログラムの1本と対応する．

② 被スケジュール側であるバッチ情報照会ファイル作成プログラムは，セットされている業務ログ・データの処理を行う．

③ 処理対象データか否かにかかわらず，ディレード・バッチ管理テーブルのビットはオフ状態にする．

④ バッチ情報照会ファイル作成プログラムは，ループ処理のプログラムで，若干の待ち時間登録を行って一定時間待ち状態を経過したあとに処理の入口に戻る．

以降①～④を繰り返す．

図3　バッチ情報照会システム

Fig. 3　Inquiring system for batch run control information

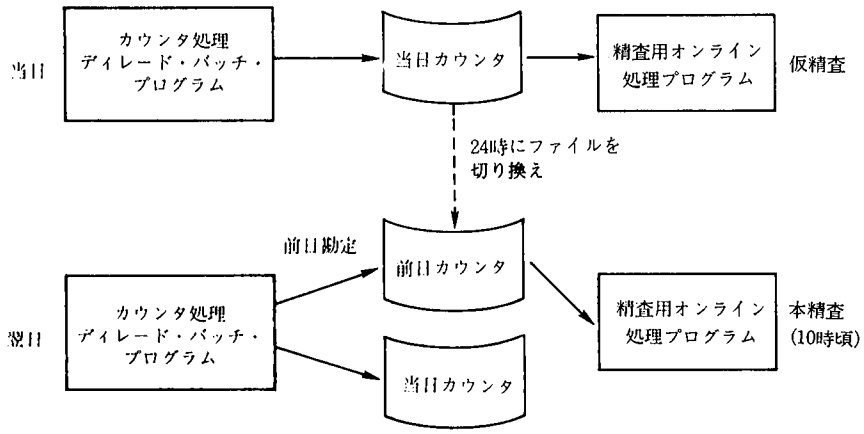


図4　2段階精査

Fig. 4　Counter matching process for nonstop real-time system

5.2.2　センターカット処理

センターカット・データの内で，従来オンライン終了後に実施していた分を前もってオンライン中に処理可能なシステム構造とする．これについては5.3節で述べる．

5.2.3　ファイル保存

従来，オンライン終了後の静的なデータベース状態で実施していたファイル保存操作を，オンライン中の動的なデータベース状態でも処理可能にするシステムを採用する．これについては5.4節で述べる．

5.2.4　2段階精査

第3章で，「金融機関においては，24時間オンラインになると，営業店で精査処理を行う時機がない」と述べた．今回のシステムでは，当日と翌日の2段階に分けて精査処理を行うことによってこの問題を解決している．図4に2段階精査の方法を示す．

なお，オンライン・トランザクション処理プログラムとカウンタ処理ディレード・バッチ・プログラムは，前日勘定取引（あたかも前日に取引が行われたかのように処理する）についても考慮する．

5.2.5　カウンタとデータベースの突き合わせ

従来，オンライン終了後の静的な状態でのカウンタとデータベースを突き合わせていた処理を，オンライン中の動的な状態でも行えるように工夫する．これについては5.4節で述べる．

5.2.6　月次バッチ

第3章で，月次バッチに関し，「24時間オンラインになると，月末日のオンライン終了という境界がなくなるため，静的な状態でファイル保存ができなくなる」という問題を指摘した．動的な状態から月末時点の静的なデータベースを作成する方法については5.4節で述べる．

5.3　処理量の平準化

今までオンライン終了後や他の連続した夜間時間帯に行っていた作業は24時間運転になると，オンライン中の処理として取り込むことになる．とくに業後時間帯に行

っている大量のセンターカット・データ処理の取り扱いが問題である．オンライン・トランザクション処理のピーク日と，これらの処理が重なると，処理量の多さから処理しきれない危険性がある．

本システムでは，ピーク日におけるトランザクションを数日前から処理するため，センターカット取引および端末取引に先日付処理を採用した．

5.3.1 先日付処理

先日付処理とは，何日か先に発生するトランザクションを前もって発生させその処理を行うが，レコードの更新はせず，本来更新すべきレコードの子レコード（先日付レコード）に処理内容を保持するものである．以下に簡単にシステムの内容を述べる．

1) 予約処理……本来の処理の数日前から行われる先日付処理の取引内容は，科目マスタ・レコードの下位に位置するスレーブ・レコードとして先日付レコードを保持し，先日付処理の時点では科目マスタ・レコードや取引明細レコードの更新は行わない．

2) メンテナンス処理……先日付処理の期日到来前に実取引が行われると，先日付処理の内容が変化してしまうため，先日付スレーブ・レコードをつくり直す処理である．たとえば，最初の先日付処理では残高不足などの理由で取引不能であっても，期日到来前の他の取引（たとえば入金取引）によってエラー理由が解消され取引可能になるケースがその例である．取引可能から取引不能になるケースも同様である．

3) 解放処理……先日付スレーブ・レコードが存在している口座に対し，その後の取引で先日付スレーブ・レコードのうちで，すでに期日を経過しているものを検知したら，その時点で該当分を科目マスタ・レコードや取引明細レコードに反映させる．反映済み先日付レコードは削除する．

1)～3)の流れを図5に示す．

先日付処理のために，複数日分のカウンタ・ファイルや専用のトランザクション・ログ・ファイルを持っていることは，図1ですでに示してある．これらのカウンタやログについて，シフト処理やマージ処理により，先日付処理を実現している．

また，月次バッチ・マスタ作成時に，先日付レコードのうちで期日が到来しているにもかかわらず，その後の取引がなくまだ科目マスタ・レコード上に反映されていない分についても考慮する（バッチ・マスタ上でのみの反映処理を行う）．

なお，きわめて大量の処理を行わねばならない元加処理についても，センターカット処理とし，平準化のために先日付処理ロジックを組み込んだ．

5.3.2 センターカット処理

処理量の平準化を狙いとして，先日付センターカット処理を実現したわけだが，それによりある1日をとらえた場合，何日分もの多種多様のデータを処理せざるを得なくなる．運用部門に対する負担増を回避するために，センターカット処理の自動的な走行制御機能が求められる．

さらに，オンライン処理中のセンターカット走行という面から，流量制御機能が必要になる．

1) 自動走行制御……複数日分のセンターカット・データ・ファイルを別々に持ち，

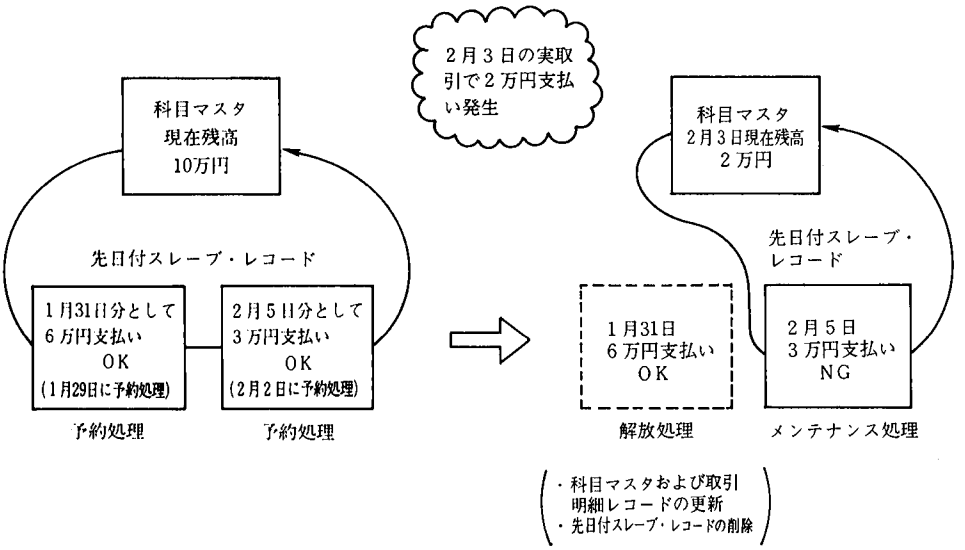


図5　先日付処理

Fig. 5　Predated transaction processing system for banks

何日分かのデータを該当ファイル上に展開しておく．センターカット走行制御ルーチンは，スケジュール制御テーブル内の情報によって優先順位を判定しながら，センターカット・スケジューラの走行を自動制御する．

2)　流量制御……オンライン中のセンターカット走行により，端末側からのトランザクション処理が遅延しないことを狙いとし，スケジューラが取り上げるセンターカット・データの件数をダイナミックに制御する．

この機能を可能にするため，メモリ・テーブルとして，センターカット・ダイナミック制御テーブルを用意し，システム負荷の程度によりスケジュール件数を制御している．

システム負荷の程度は，リモート・シンビオント・インタフェースを使用してOSが管理しているトランザクション・キューの個数を知ることにより判定している．

したがって，センターカット・ダイナミック制御テーブルの要素は，スケジューラが一度にスケジュールする，待ち状態のトランザクション・キューの個数ごとに設定したデータ件数である．なお，流量制御は7秒間隔で機能する．

センターカット・システムを図6に示す．なお，センターカット処理の効率化については5.5節で述べる．

また，3章で述べたオンライン中のセンターカットによるデッドロック発生率の上昇については，次のように考慮している．

データベース処理ルーチン側に，デッドロックが発生した場合，該当アプリケーション処理プログラムはエラー扱いとせず，待時間をはさみつつ内部的に同一データを発生させ，規定回数の再試行を行う仕掛けを取り入れた．なお，この工夫はアプリケーション処理プログラムに何ら影響を与えない．

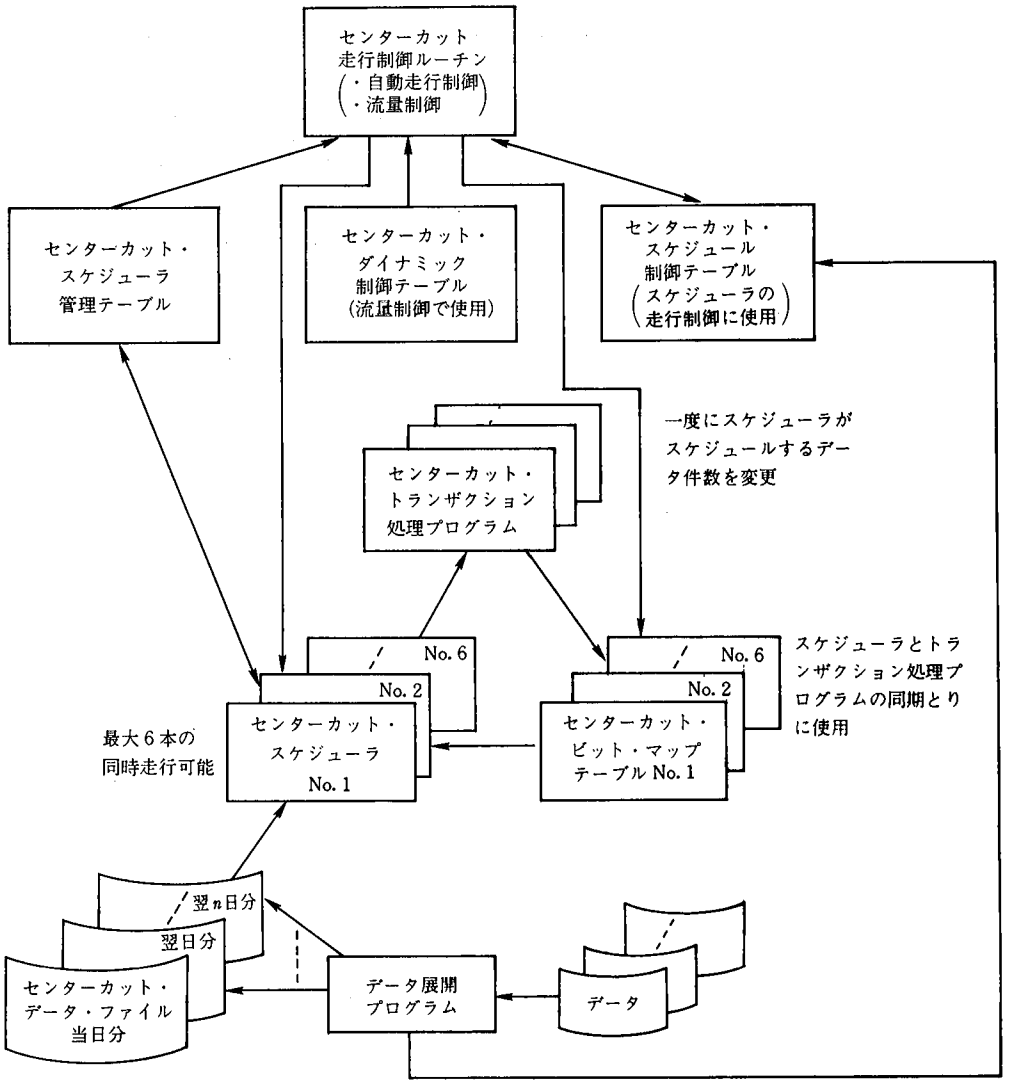


図6　センターカット・システム

Fig. 6　Center-cut system

5.4　静的状態を要求しないシステム

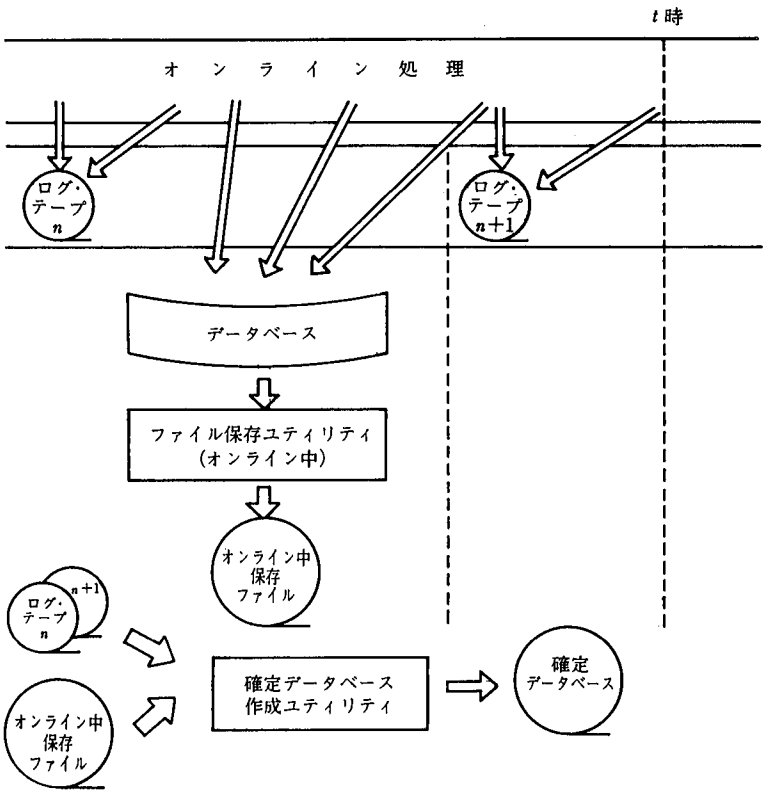
結論から述べると，ファイル保存，月次バッチ，金利変更，元加処理については静的なファイル状態を要求しないシステムが実現した．しかし残念ながら，ファイル・メンテナンスおよびカウンタとデータベースの突き合わせ処理については，目標は達成されず，運用面での対処となった．

5.4.1　ファイル保存

オンライン中のファイル保存の方法と，ある任意の時点の確定データベース作成の方法を図7に示す．

5.4.2　ファイル・メンテナンス

ファイル・メンテナンスは，不要レコード削除プログラムとファイル再編成ユティ



① オンライン中にファイル保存ユティリティを実行する．ただし，テープ上にセーブされたデータベースの内容は，論理的なレコード順になっていないし，かつレコード間の整合性も保たれていない．
② ファイル保存ユティリティの走行時には，テープ n にデータベースのアフタ・ルックスを含むログが書き出されていたものとする．
③ t 時現在の確定データベースを作成する場合は，①で保存したセーブ・テープとログ・テープ n と $n+1$ を併合することによって行う．
④ 確定データベース作成ユティリティは，まずログ・テープ内から必要なデータベースのアフタ・ルックスを抽出し，これをデータベースの物理的な順に分類する．次に分類済みファイルとオンライン中保存テープとの併合を行って確定データベースが完成する．

図7　オンライン中のファイル保存

Fig. 7　Physical dump and merged dump utilities[1] used under online operations

リティを組み合わせて行う．なお，再編成処理中には，該当ファイルを使用するオンライン・トランザクションのみ処理不能となる．手順を図8に示す．

中断時間を短縮するためには，次のような考慮が必要である．

- 24時間，連続的に使用されるファイルについては，ファイルの容量を大きくとらず，店群単位などに細分化したファイル構成とする．
- 複数のファイル間に物理的な相互関係をもたない（Aファイルをメンテナンスすると，Bファイルも同一タイミングでメンテナンスが必要となるということがないようにする）．

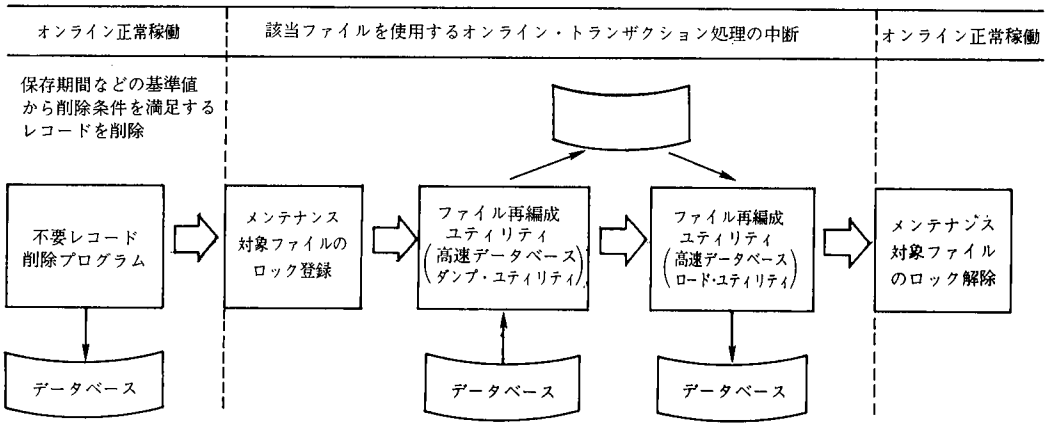


図8　ファイル・メンテナンス
Fig. 8　File maintenance

金融機関におけるファイルの実態を考えると，現在の技術では中断時間が10分近くになる．ある店群のみの中断とはいえ，運用時間帯としては夜間のオンライン・トランザクション量が少い時を選択せざるを得ない．1～2分程度の中断で済ませ，日中に処理可能となるような技術が今後の課題である．

5.4.3　カウンタとデータベースの突き合わせ

本項目についても，ファイル・メンテナンスと同様に，本来の目標である静的状態を要求しないシステムは達成されなかった．

検証プログラムの走行単位をグルーピング（金融機関においては店群単位）して，数日間のサイクルで実行し効率化を図った．しかし，検証プログラムの走行中に，店単位に取引ロックによる中断が発生するという課題が残されている．

5.4.4　月次バッチ

3章で，「24時間運転では，月末現在の確定データベースの作成が問題だ」と述べた．しかし，これについては，5.4.1項にて任意の時点の確定データベースの作成が可能であることを示したとおり解決された．

5.4.5　金利変更

本システムでは，従来のバッチ・ベースでの処理をやめ，オンライン・トランザクション処理の中で自動的に行われるように配慮した．具体的には，金利変更に対してはテーブル上の金利のみを変更すれば良いようにした．そして金利変更後の最初の取引においては，オンライン・トランザクション処理プログラムが，まず予想積数の再計算をしてから実際の処理を行う方式にした．

5.4.6　元加処理

バッチ作業による元加処理からセンターカット方式による処理に変更したため，オンライン中の処理が可能となった．しかも5.3.1項で述べた先日付処理ロジックを組み込んだため，処理量の平準化が実現した．

5.5 処理の効率化

5.5.1 センターカット処理の効率化

実際の処理効率値については省略するが，次に示す工夫により高効率を達成している．

1) 入力データのランダマイズ分類を採用した．つまり，個々の入力データに乱数を付加し，これをキーの一部として位置付けて分類した．分類で使用する乱数は，以下の方法で決定した．N 件目の入力データに与える乱数 Z の計算方法は次のとおりである．

$$S = A + N \times B$$
$$S = X\,(2^{18} - 1) + R$$
$$Z = R$$

ただし A = 7 桁の整数
B = 6 桁の整数
X = 被除数 S を除数 $2^{18}-1$ で除した際の商
R = 被除数 R を除数 $2^{18}-1$ で除した際の余り

この方式により I/O 負荷分散が図れた．

2) マルチ・センターカット方式を採用した．同一種類のセンターカット・データを 2 本のセンターカット・スケジューラによって同時にスケジュールするよう考慮した．

3) センターカット・トランザクション処理プログラムを処理対象のデータベース（元帳）単位に分割し，I/O 負荷分散をさらに図った．元帳が N 個のマルチ・エリアとして分割されている時は，センターカットの入力データをスケジュールする前に，該データが何番のマルチ・エリアに対するデータなのかを判定する．その結果をもとに個々のマルチ・エリアに対応したセンターカット・トランザクション処理プログラムをスケジュールする．

エリア単位のセンターカット・トランザクション処理プログラムのコピー数を 3 にした場合，最大 $3 \times N$ 本のセンターカット・トランザクション処理プログラムが同時走行することになる．

5.5.2 元加処理の効率化

元加処理では金融機関の規模にもよるが，何百万件ものデータを処理する必要がある．5.4.6 項で述べたように，元加処理は先日付センターカット化によって数日間にわたって分散して実施できるようになっているが，その処理効率向上はやはり基本的な課題として残されている．

何百万件ものデータがある場合は，秒当たり 100 件程度の効率が必要である．元加処理センターカットについては，5.5.1 項で述べた対処以外に，センターカット・トランザクション処理プログラムをループ処理型にすることにより，さらに効率アップを図っている．

つまり，1 回のデータ・スケジュールで複数件のデータを処理プログラムが受け，これをループ処理する．したがって，初期処理と終了処理の走行ステップの一部が削減可能となる．この方式により，目標値を達成している．

5.5.3　先日付処理による負荷上昇の評価

従来のオンライン・トランザクション処理では，プロセス上存在しなかった解放処理とメンテナンス処理が，先日付処理の採用によって必要になった．

オンライン・トランザクション処理を行う場合，該当するレコードに何件の先日付スレーブ・レコードが存在しているかによって処理ステップ数は異なる．

しかし，一般に解放およびメンテナンス処理が両方走行すると，先日付処理がまったく行われない場合に比べ，平均的に50％程度ステップ数増となる結果が得られている．

以下のような条件設定を行うと，システム全体での負荷は，先日付処理がまったくない場合に比べ10％程度上昇すると予想される．

［負荷上昇算定条件］

① 先日付処理によるステップ数の増加を A（50％）とする．

② 先日付処理関連取引の全体取引に占める比率を B（70％）とする．

③ 先日付処理関連取引のうち，処理レコードに実際に先日付スレーブ・レコードが存在している確率を C（30％）とする．

以上の条件のもとに，負荷上昇率を求めると，次のごとくなる．

$$\text{負荷上昇率} = A \times B \times C \fallingdotseq 0.1$$

5.6　複数マシンの活用

24時間稼働を考えると，機器構成は大がかりなものにならざるを得ない．複数マシンを活用することによって，24時間運転時のシステム開発，システム変更，ハードウェアの保守などの問題を解決することも可能であろう．中央処理装置の構成として，図9に示すように，切換機，開発機，フロント・エンド・プロセッサ（FEP）機が必要であると考える．また，周辺機器装置もすべて二重化が必要であろう．

5.6.1　システム開発

開発専用マシンの設置によって自由にシステム開発ができる．本番ファイルを使用したい場合は，5.4.1項で述べたオンライン中のファイル保存と確定データベース作成機能によってファイルを作成できるので，これを開発機に展開すればよい．ただし，開発機側のディスク装置容量は，本番機と同等の容量が望ましい．

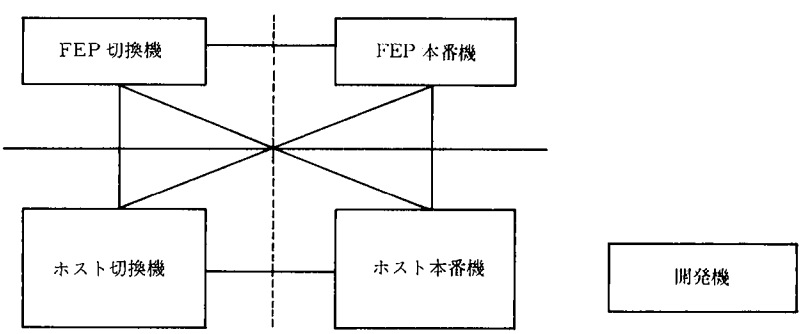


図9　中央処理装置の構成

Fig. 9　CPU configuration for a nonstop system

5.6.2 システム変更

オンライン・トランザクション処理プログラムの入換は，トランザクション処理制御プログラムの既存機能により，オンライン走行中でも可能である．

基本ソフトウェアの入換については，ホット・スタンバイ方式により，瞬断レベルで行うことが基本的に必要である．以下にシステム変更の手順を示す．

① 通常オンライン
② 自動切換モードから手動切換モードへの変更
③ 切換機側のソフトウェア入換
④ 手動モードでのマシンの切換
⑤ 新切換機側（旧本番機）のソフトウェアの入換
⑥ 手動切換モードから自動切換モードへの変更
⑦ 通常オンライン

FEP 機採用により，ホスト側瞬断時の端末側に対する一時的処理中断表示サービスが行える．

機器増設などは，OS 側の新技術によって，中断無しでシステム変更を可能とするケースが今後広がっていくものと予想される．

5.6.3 ハードウェアの保守

ホット・スタンバイ方式や周辺機器の二重化を利用して行う．中央処理装置については，プロダクション・モードでない時に保守を行う．周辺機器装置については，ダイナミック・ダウン，あるいはアップによるシステムからの切離し・結合を利用して保守を行う．

5.7 システム運用

5.7.1 不要ファイルの削除

初期化ブートができないため，不要ファイルの整理ができないという問題があるが，不要ファイル削除ユティリティがすでに存在しているのでこれを利用する．

また，OS が管理するマスタ・ログ・ファイルについては，運用によってファイル・サイクル番号を最大値に切り換え対処している．

5.7.2 常駐ランの連続走行およびメモリ・テーブルの初期化処理

これらの問題点については，3章で述べたが，解決のために1日に1回（24時に），数分間の中断時間を設定する．この時間帯でメモリ・テーブルの初期化処理と常駐ランの載せ直しを行う．業務処理に必要なテーブル上の日付変更もこの時間帯で実施する．

瞬断中の端末側無応答状態をなくすため，5.6.2項と同様に，FEP 機側で，端末側に対し一時的な処理の中断を表示する．

5.7.3 自動機器の管理・監視

自動機器無人監視端末装置を中核とした無人運転監視パッケージがすでにあるので，これを導入し集中監視により営業店側の人手による監視・管理を不要にする．

6. おわりに

本稿のテーマは複雑であり，そのすべてを書きつくすことは困難であった．

筆者の関係したシステムの開発では，目標として掲げたシステムの設計指針をベースに，24時間運転のための各種対応を行うことができた．しかし，ファイル・メンテナンスなどでは，静的なファイル状態を要求しないシステムによる解決ではなく，運用面での対処となった．

24時間運転を実現するために，システムそのものがより複雑になったこと，さらには機器構成についても大規模なものになったことが，課題として残されている．

今後，OSなどの新技術の発展によって，完全な24時間運転に，より近づいていくと信じている．

最後に，このような技術的に興味の深いシステムに取り組む機会を与えて下さった金融関係の諸ユーザの方々と，デザイン・レベルでのアドバイスを頂いた日本ユニバックの喜入博氏に心から感謝の意を表する．

参考文献　［1］ PALDUM解説書，日本ユニバック(株)，1986年9月，pp. 1～12.

執筆者紹介　村　田　豊　彦（Toyohiko Murata）

昭和26年生．49年慶応大学工学部管理工学科卒業．同年日本ユニバック(株)入社．金融機関のシステム開発に従事．現在システム第1本部システム統括1部システム3部システム開発1室に所属．

グローバル・トレーディング・システムの概要

Overview of Global Trading System

伊川　望
N. Ikawa

1. はじめに

今日の金融業界を取り巻く環境は，金利の自由化と業務の自由化の中で大きな変革期を迎えようとしている．とくに昨今の円の通貨力の高まりと共に，企業の信用力向上を背景とした直接資金調達の拡大や，低金利時代を反映した投資家の利回り選好を受けての自由金利商品へのシフト等，銀行の経営基盤も変化しつつある．従来の預金・貸金を中心としたマネービジネスに加えて，今後拡大が予想される直接金融市場をベースとするマネービジネスに対する取り組み，すなわちセキュリティゼーション（証券化）への積極的な対応である．制度的にも昭和58年6月1日より開始された公共債のディーリング，昭和60年10月19日に創設された債券先物市場への直接参加等，業務の緩和が進展しつつある．

一方証券業界においても，手数料の自由化や外資系企業の証券参入という大きな課題に直面している．収益の基盤であった手数料収益の相対的な低下が予想されるなかで，より魅力的な商品の提供や有用な情報の提供という手段で顧客の確保を図ると同時に，自己運用（ディーリング）による収益の確保が図られようとしている．機械化の状況も事務の合理化を主眼とした処理形態から，戦略の一環としての機械化へと大きく変貌しつつあることが，これを裏づけている．

このように金融業界および証券業界のいずれもが自由化の波にさらされているが，さらに国際化の波が急速におとずれようとしている．60年秋にロンドンで実施された市場開放（ビッグバン）の影響や，欧米各国からの市場開放に向けての圧力等の外的要因に加えて，国内からもより安定的な利回りの確保，リスク回避手段の多様化，より有利な資金調達手段の選択のニーズから，海外市場・商品への積極的な対応が求められている．

自由化と国際化という二つの波に対して，ますます重要性を高めているのがコンピュータ・システムによるサポートである．より広範な多量の情報の入手・分析，高度化・多様化する運用調達手法への対応，合理的な投資判断基準の提供，魅力ある商品の開発等，コンピュータ・システムに求められる要求は多い．また，こうしたシステム対応においては，いわゆる“先行者メリット”を享受できる可能性も高い．新しい波に対して，“攻め”と“守り”の両面においてコンピュータ・システムからのサポートの程度が，グローバル・トレーディングの巧拙を支配するといっても過言ではない．

本稿では，最近関心を集めているグローバル・トレーディング・システムを紹介する．

2. グローバル・トレーディングの定義

“グローバル・トレーディング”という言葉は，業態によって，あるいは使う人の立場によってさまざまな意味で用いられているが，本稿では“グローバル・トレーディング”を次の要件を満たすものとして定義する．

第一の要件は「運用と調達の統合化」である．リターンとリスクの調和を目的とした資産・負債の総合管理をベースとし，そこから生ずる資金の運用ニーズ，調達ニーズおよびニーズの実施結果の総合管理を意味する．

第二の要件は「時間と空間を超えた取引の統合化」である．世界に点在する証券・金融・為替市場を時間（時差）と空間（距離差）を超えた一つの世界市場（グローバル市場）として捉え，そこで取引されている多くの商品を対象に運用・調達ニーズに基づいて自由に取引を実施し，取引の結果を総合管理することを意味する．

第三の要件は，「1次市場（発行市場）と2次市場（流通市場）の統合化」である．調達者と運用者の直接金融化指向に対し，“調達側と運用側の双方のニーズの調和を取った商品”の提供を前提とした証券会社，および金融機関を仲介とする1次市場と2次市場の総合管理を行うことを意味する．

3. グローバル・トレーディング出現の背景

グローバル・トレーディング出現の背景として，

第一に企業収益の極大化とコストの極小化がある．このほか安定した資金の運用・調達の要請に対して，国内市場・商品の利用だけでは金利水準，市場規模，市場の特性（商品や価格形成のされ方等），金融制度，危険分散の点から限界があることがあげられる．つまり，より一層の効率化を目指して海外市場・商品に目を向けざるを得ないわけである．

たとえば調達面では，ユーロ市場を利用したデットワラント債，デュアルカレンシ債等の新しい調達商品の起債や，異なる資金調達手段のスワップ取引等が活発に行われていることがあげられる．運用面では，機関投資家，信託運用を中心とする外国証券への投資へのシフトや，海外の金融先物・オプション市場を利用した裁定取引・ヘッジ取引の業務認可等が具体的な動きとしてとらえられる．また，海外現地法人・支店の設立が相次いでいることも海外市場への積極的な対応の姿勢に他ならない．

第二の背景としては情報のグローバル化やリアルタイム化が加速度的に進められていることがあげられる．自社・自行内の国内外のネットワークの整備が進み，本部と海外現地法人・支店間のリアルタイムな情報交換が実現され，またロイター，テレレートに代表される情報ビジネスの拡大によって海外の市況，経済・金融情報がリアルタイムに入手できるようになった．これによって国内と海外の時間と空間の垣根がなくなり，全世界を一つの市場として捉えたグローバル・トレーディングが可能になったといえよう．

4. グローバル・トレーディングを支えるコンピュータ・システムの要件

グローバル・トレーディングを円滑に進めていくために，コンピュータ・システムには四つの要件が求められていると考えられる．

第一に取引（運用と調達，発行と流通）を戦略的に進めていくための支援機能が求められる．世界には多くの市場・商品と無数の運用・調達ニーズが存在する．これらの膨大な情報を入手し，管理し，分析し，最適な取引を実現するためには，コンピュータ・システムが不可欠である．ネットワークを通じて入手した情報の管理・分析をコンピュータ・システムに委ね，コンピュータ・システムを利用して導き出された結果を人間が判断し実行することによって，戦略的なグローバル・トレーディング運営が可能となる．具体的には次の要件があげられる．

1) 取引手法の高度化と多様化への対応……新規調達商品の開発や，異なる調達手段のスワップ，および先物・オプションを利用した裁定取引やヘッジ取引が例としてあげられる．
2) 合理的判断基準の提供……相場分析の各種手法，プログラム売買の各種手法，およびポートフォリオ分析の各種手法の利用が例としてあげられる．
3) 顧客ニーズの多様化への対応……顧客の運用ニーズ，調達ニーズ，スワップニーズおよびポジションの状況を本部，海外現地法人・支店で相互に把握し，各種の取引手法・分析手法を用いて対応の実施を図ることが要請されている．

その際，重要なポイントは，コンピュータ・システムの利用をあくまでも“判断基準の提供”に留め，最終的な判断は，トレーディングを運営する人間や組織（トレーダとファンド・マネージャ）が行うことである．ウォール・ストリートで問題にされている“魔の第三金曜日”や，まだ記憶に新しい昨年10月19日に発生した日・英・米の3市場での連鎖的な株式の大暴落の一因をコンピュータ・システムを利用したプログラム・トレーディングに帰する声もある．

ここでプログラム・トレーディングの功罪を論ずるつもりはないが，コンピュータ・システムからの売買指針を鵜呑みにするのではなく，トレーディング運営者が自分の相場感，環境分析等と照らし合わせて決断し実行するハイブリッド型*の運用が望まれる．

第二に取引を効率的に進めていくための支援機能が求められる．グローバル・トレーディングの目的は，リターンとリスクの調和と安定的な資金運用・調達を世界の各市場・商品を利用して実現

---

* 収益追求を狙いとするアクティブ運用に対して，一定の収益を確保しながらリスクの最小化を図るパッシブ運用がある．前者がトレーダやファンド・マネージャの才覚に委ねた運用が中心になるのに対し，後者はコンピュータ・システムを利用したプログラム・トレーディングが中心となる．ハイブリッド運用はその複合型であり，コンピュータ・システムが導き出した“売買の指針”をトレーダやファンド・マネージャが判断し取引を実施する運用である．

していくことにある．その際，取引をタイムリに進めていくことが重要なポイントになる．ある状況下で最適であった判断も，状況が変化した後ではもはや役に立たないケースが多々ある．いわゆる“相場の世界”においてはそれが顕著であり，売買のタイミングがとくに重視される．したがって，“状況の変化の把握→分析・判断→取引の実施”のサイクルをいかに短時間で実行するかが課題となる．

また，世界の市場を渡っていくと24時間連続した取引が可能になる．CME (Chicago Mercantile Exchange：シカゴ商品取引所) のように1市場内で24時間の取引を実現しようとする動きもある．グローバル・トレーディングが“24時間トレーディング”と称されるゆえんである．

このように24時間運営されている市場に対して，取引をタイムリに進めていくためにはコンピュータ・システムの利用が不可欠である．具体的には，次の要件があげられる．

1) ディジタル化された市況情報のリアルタイムな入手
2) 入手された情報の分析および判断の一部自動化
3) 注文・約定の自動管理

第三にポジションを迅速にかつ正確に把握するための支援機能が求められる．ポジションとは，運用・調達残の商品構成，在り高，コスト，期間の状況であり，このポジションの管理をベースに次の状況を把握することができる．

1) 相場実勢との対比による含み損益
2) 将来の状況の変化（金利・為替・価格等の変化）への適合性やリスクの度合
3) 新たに生ずる運用・調達ニーズやスワップニーズ

ポジション管理の主眼は，商品ごとの在り高とコストの管理が中心であるが，先物取引・オプション取引・スワップ取引等のオフバランスのポジション管理や，裁定取引・ヘッジ取引を実施した際のネット損益の把握を目的とした複合取引のポジション管理も取引手法の高度化・多様化の動きと相まって求められてきている．

管理の単位としては，本部ポジションの一元管理に加えて担当グループ・担当者別の管理や，運用調達の目的別の管理等細分化が求められている．これに対して，海外現地法人・支店での運用・調達活動が活発化しつつある現状では，そのポジションも看過できない状況となっており，本部と一体化した総合ポジション管理の要請も高まりつつある．

また，自社・自行のポジション管理の充実のみならず，対顧客戦略上の観点から主要顧客のポジションも併せた管理が望まれている．

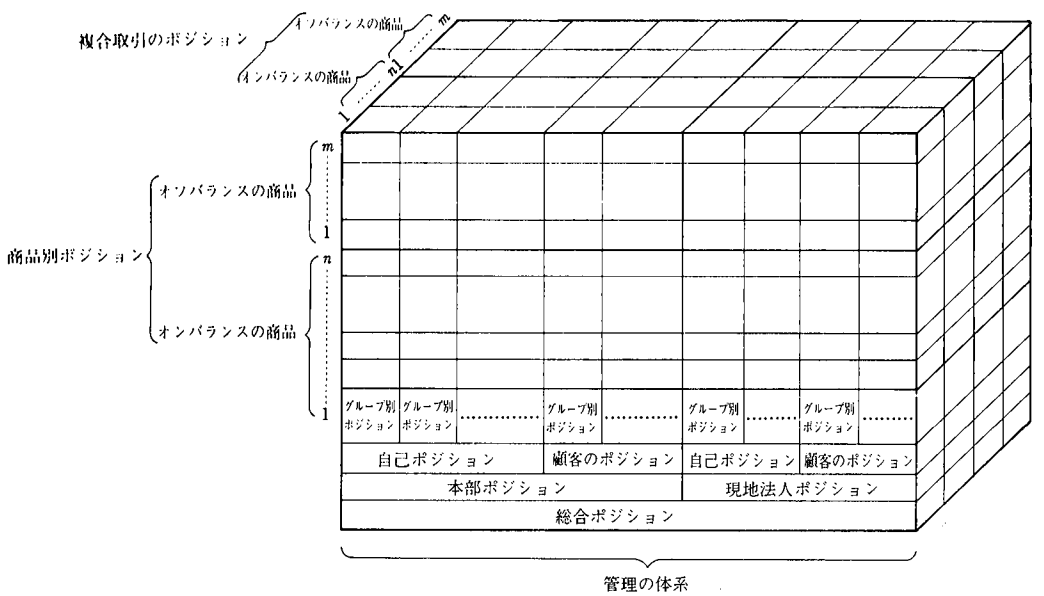


図1 ポジション管理の体系

Fig. 1 Consolidated position management system

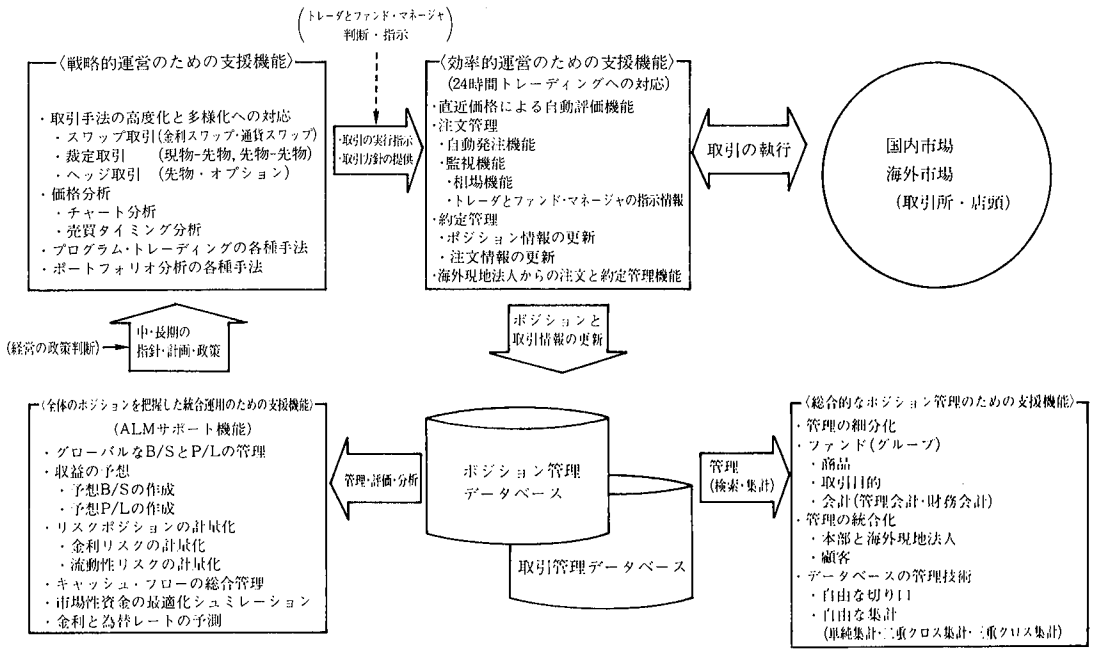


図 2　コンピュータ・システムに求められる機能の相互関連

Fig. 2　Various roles of computer to support global trading

このようにポジション管理に対する要求は多岐にわたるが，管理の体系は図1のように整理される．

ポジション管理と同様に取引の実績管理も重要な要件である．単一商品の売買実績だけではなく，オフバランス取引の実績，裁定取引・ヘッジ取引等の複合取引の実績(コスト，ネット損益)，調達→運用の実績（キャリングコストやネット損益）管理も，精緻な収益とコストの把握のためには不可欠な要素となってくる．

さらにコンピュータ・システムで管理されたポジションおよび取引の情報を“自由な切り口とフォーマット”で随時引出してくることが最終的に求められる．“情報のガラス張り化”である．今日のように目まぐるしく変化する国内外の経済・金融情勢に対して，適切な判断を下していくためには従来の固定的な情報提供では対応できず，利用者の目的に応じ情報が自由にコンピュータ・システムから提供されることが望まれる．

最後に全体のポジションを把握した統合運用のための支援機能が要求される．トレーディング運営においては，日々の業務を最適に遂行していくことが重要であるが，それは中長期的な視野からの方針・計画・政策が明確になってこそ初めて可能になると考えられる．たとえば，将来の金利水準・為替水準，運用・調達のバランス・期間構成，リスクポジション，キャッシュ・フロー等が把握されているか否かによって日々の業務の進め方は大きく異なってくる．

そのために，本部の運用・調達ポジションに加えて，海外現地法人・支店のポジションまで含めた全体のポジション管理を前提とした統合運用のための支援機能，すなわち ALM (Asset and Liability Management) サポート・システムが求められる．ALM サポート・システムとしては

1)　グローバルな B/S と P/L の管理
2)　収益の予想（予想 B/S，P/L）
3)　リスクポジション（金利リスク・流動性リスク）の計量化
4)　キャッシュフローの総合管理
5)　市場性資金の最適運用・調達シミュレーション

が具体的な要件としてあげられる．

グローバル・トレーディングを支えるコンピュータ・システムの四つの要件について述べてきたが，それらの相互関連をまとめると図2のごとくなる．

5. グローバル・トレーディング・システムの機能

以上四つの支援機能の概略を検討したが，この章ではそのうち現在最大の関心事である証券を中心とした運用の検討を進める．したがって，資金とALMについては本稿では検討をひかえる．

5.1 戦略的な運営のための支援機能

5.1.1 相場情報を中心とした機能の展開

戦略的な運営のための支援機能とは，トレーダやファンド・マネージャ等の第一線の売買担当者が，時々刻々と変化する相場の状況を睨みながら売買を実施していく際の合理的な判断基準を提供する機能である．ここで最も重要な情報源となるのが，相場（価格・出来高）の情報である．相場情報が重要視される理由として，次の3点があげられる．

第一にトレーダやファンド・マネージャは，相場状況を睨みながら売買を実施している．したがって，売買の基準となる情報は相場を反映したものでなければならない．いわゆる，“相場は相場に聞く”ことが運用の基本となる．

第二に相場情報は，リアルタイムにかつ継続的に入手できる唯一の公開された情報である．

第三に市場が効率的に運営されているという前提に立つと，経済・金融等のファンダメンタルの変化や多くの投資家達の投資意向は即時に価格に反映されると考えられる．

以上の理由から相場情報，とくに価格情報を中心とした運用手法・分析手法の研究や具体化が進められていると判断される．価格情報を利用した運用・分析の流れは図3のように整理される．

5.1.2 価格分析

価格分析とは価格の変動によってもたらされる各種の変化（価格変動の周期性・転換点・理論値からの乖離等）を敏感に読み取り，“相場のアヤ”を判断することによって売買タイミングを決定するための分析手法である．価格分析の手法は，大きくチャートを中心にした分析手法と，売買タイミングを中心にした分析手法に分類される．

1) チャートを中心にした分析手法……移動平均線やローソク足チャートに代表される図解を中心とした分析手法である．古い歴史を持つ手法ではあるが，結果の判断は長い経験を必要とし，ともすれば主観的な判断に陥りやすい．

2) 売買タイミングを中心とした分析手法……ポイント&フィギャに代表される短期的な売買のタイミングを把握し，結果として利益の極大化を図ることに主眼をおいた分析手法である．最近では，コンピュータ・システムによる数値加工処理をベースにした米国型の売買タイミング分析手法（MTA；モダンテクニカル分析手法）が注目されつつある．

5.1.3 プログラム・トレーディング

プログラム・トレーディングとは，目的に応じて設定された一定のリターンレベルを確保しながら，リスクの最小化を図る運用である．運営にコンピュータ・システムが活用されていることから，この名称が付されている．プログラム・トレーディングは図4で概観されるように，情報の収集・管理，加工・分析，売買指針の提供，取引の執行，取引報告までが一体化したシステムである．

リスクをできるだけ排除し安定的な収益の確保を志向して，わが国においても手法の研究が活発

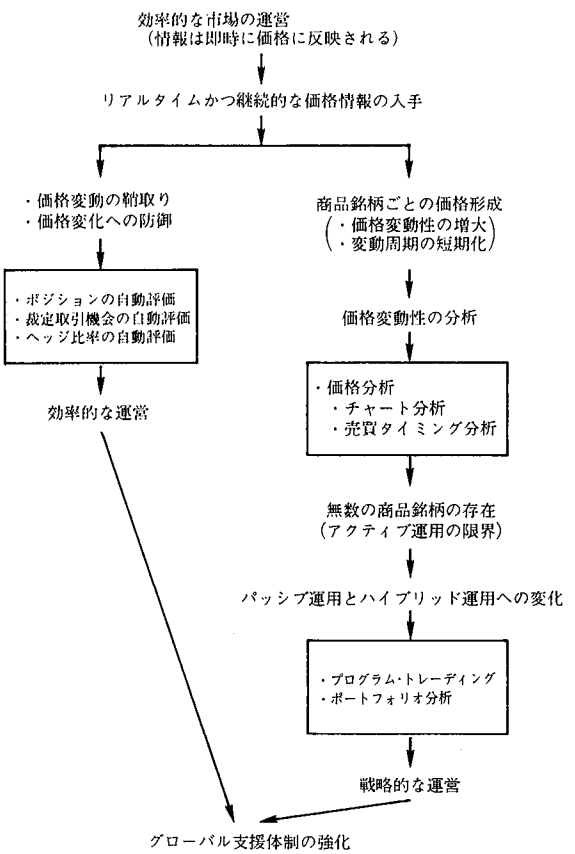


図3 価格情報を中心とした運用・分析手法の流れ

Fig. 3 Operational and analytical methods based on price information

に行われ，すでに一部の証券会社では実用化が図られている．

プログラム・トレーディングの代表的な手法として

① インデックス・ファンド（Index Fund）
② リバランス（Rebalance）
③ ベーシス・トレーディング（Basis Trading）
④ イミュニゼーション（Immunization）
⑤ ポートフォリオ・インシャランス（Portfolio Insurance）

があげられる．

5.1.4 ポートフォリオ分析

プログラム・トレーディングと並んで，最近再び注目されているのがポートフォリオ分析である．ポートフォリオ分析は，ポートフォリオの組み入れ銘柄，および組み入れ比率をリターンとリスクの調和を取りながら決定していく目的で使用される．とくに，中長期の保有を目的とした資産運用に利用されているケースが多い．実用化されているポートフォリオ分析手法としては，次のものがあげられる．

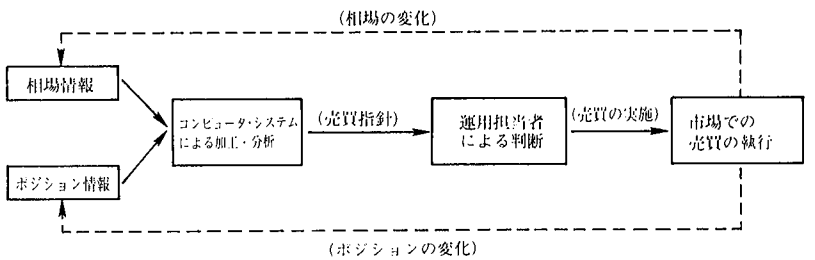
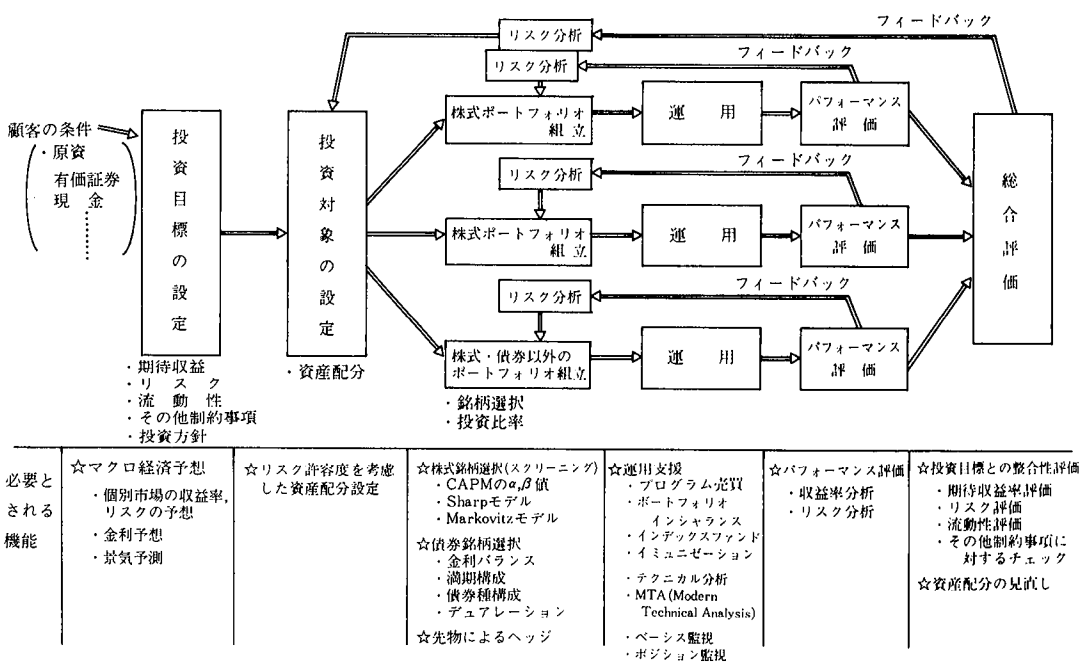


図 4　プログラム・トレーディングの流れ

Fig. 4　Process flow of programmed trading



（日本ユニバック：グローバル・トレーディング・システム検討プロジェクト編）

図 5　ポートフォリオ運用の流れと必要とされる機能

Fig. 5　Porfolio selection system and its components

表1　主要機能とソフトウェアの対応

Table 1　Application packages used for portfolio selection

ソフトウェア名称	手　　法	主 な 適 用 分 野
FMPS GAMMA FMPS/RG	・線形最適化手法	MPT(ポートフォリオ・モデル〔Sharp 法〕) ・資産選択モデル ・株式・債券選択モデル
QP	・非線形最適化手法	MPT(ポートフォリオ・モデル〔Markovitz 法〕) ・資産選択モデル ・株式・債券選択モデル
STAT-PACK BMDP 1100 MASCOT-II SUFICS1100	・統計および予測 ・クロス集計 ・多変量解析 　重回帰分析 　判別分析 　主成分分析 　因子分析 　クラスタ分析 　数量化理論Ⅰ類 　数量化理論Ⅱ類 　数量化理論Ⅲ類 　数量化理論Ⅳ類 ・モデル・シミュレーション 　感度分析 　目標設定 　シミュレーション ・リスクアナリシス	・CAPM(資本資産評価モデル) ・ファクタモデル ・APT モデル(裁定価格理論) ・DDM(配当割引モデル) ・ファンダメンタル分析(相関分析α値,β値など) ・時系列分析 ・ALM(資産負債管理) ・金利予測 ・イミュニゼーション ・ポートフォリオ・インシャランス ・その他

① ポートフォリオ・モデル (Markovitz 法)
② ポートフォリオ・モデル (Sharp 法)
③ CAPM (資本資産評価モデル：Capital Asset Pricing Model)
④ ファクタ・モデル
⑤ APT モデル (裁定価格理論：Arbitrage Pricing Theory)
⑥ DDM (配当割引モデル)

図5は，これらのモデルを組み込んだポートフォリオ運用の流れを示したものである．また，表1には使用される経営科学パッケージの一覧を掲載した．

5.2　効率的な運営のための支援機能

効率的な運営のための支援機能のポイントは二つあると考えられる．

まず第一のポイントは，相場の変化をいち早く把握し売買のタイミングを逸しないための機能を提供することにある．たとえば，裁定取引は現物と先物間の理論価格と実勢価格の乖離を利用する取引であるが，この乖離は相場の流れの中でごく短い時間だけ認識される．したがって，いかに早くこの乖離に気がつくかが裁定取引のポイントになる．

第二のポイントは，24時間トレーディングへの対応である．24時間取引される世界市場に対して効率的な運営体制を取るには，コンピュータ・システムの活用が不可欠になる．相場の変化を自動的に捉えながら発注・約定を管理する機能をコンピュータ・システムに備えることで，人的資源を最少限に押さえた運営体制が可能になり，管理技術の進歩によっては無人化トレーディングの実現も考えられる．

5.2.1　自動評価機能

リアルタイムに入手される相場情報をコンピュータ・システムが管理し

① ポジションの評価
② 裁定取引の評価
③ 売買指標値（売買タイミング分析から出力される指標値）の評価

を自動的に行う機能である．この機能によりトレーダやファンド・マネージャは相場を常に監視する必要はなく，

① 瞬時の鞘抜き
② 裁定取引の取組み
③ 希望値による売買

をタイミングの遅れなく実現することができる．各評価は次のように行う．

1) ポジションの自動評価……直近の価格によ

ってポジションを自動評価し，現在の含み損益の把握および一定の水準以上（以下）の玉（個別取引，複合取引，銘柄）の自動抽出ならびに，音と色による注意信号と共に売買指針（ロット・コスト・含み損益等）の提供を行う．

2) 裁定取引の自動評価……直近の価格によって，市場間，商品間，限月間（先物）の価格差を瞬時に把握し，一定水準以上の乖離に対して，裁定取引の指針（ベーシスやIRR：Internal Rate of Return 等）を提供する．

3) 売買指標値の自動評価……売買タイミング分析によって算出された指標値，あるいはトレーダやファンド・マネージャによって指示された指標値と直近の価格を比較し，直近の価格が上回った（下回った）時に売買指針の提供を行う．

5.2.2 24時間トレーディングへの対応

24時間トレーディングを合理的に運営していくためには，自動発注機能の整備が重要なポイントになる．トレーダやファンド・マネージャの意図を反映した自動評価機能と連動させ，注文を自動化させる機能や，顧客からの注文を海外市場に24時間つなぐための機能がこれに該当する．自動発注機能として，次の三つがあげられる．

1) 発注機能……各市場の立会時間を管理しながら，自動評価機能からの売買実行指示や顧客からの注文を各市場に発注していく．相場の暴騰や暴落の監視，トレーダやファンド・マネージャからの指示による発注停止，あるいは出来状況に応じた発注条件の変更等も必要と考えられる．

2) 約定機能……国内と海外からの約定データによるポジションの即時更新および発注情報の更新機能が求められる．

3) 海外（海外現地法人・支店・顧客）からの注文・約定管理機能……海外現地法人等からの国内市場に対する注文の管理機能で，国内市場への発注および約定結果の通知機能が求められる．

自動発注機能を中心とした情報の流れを図6に整理する．

5.3 ポジション管理のための支援機能

本部・海外現地法人・顧客の総合的なポジション管理を目的とする．ポジション管理は，グローバル・トレーディングの根幹をなすものであり，ポジションの状況把握を，正確にかつ迅速に行う

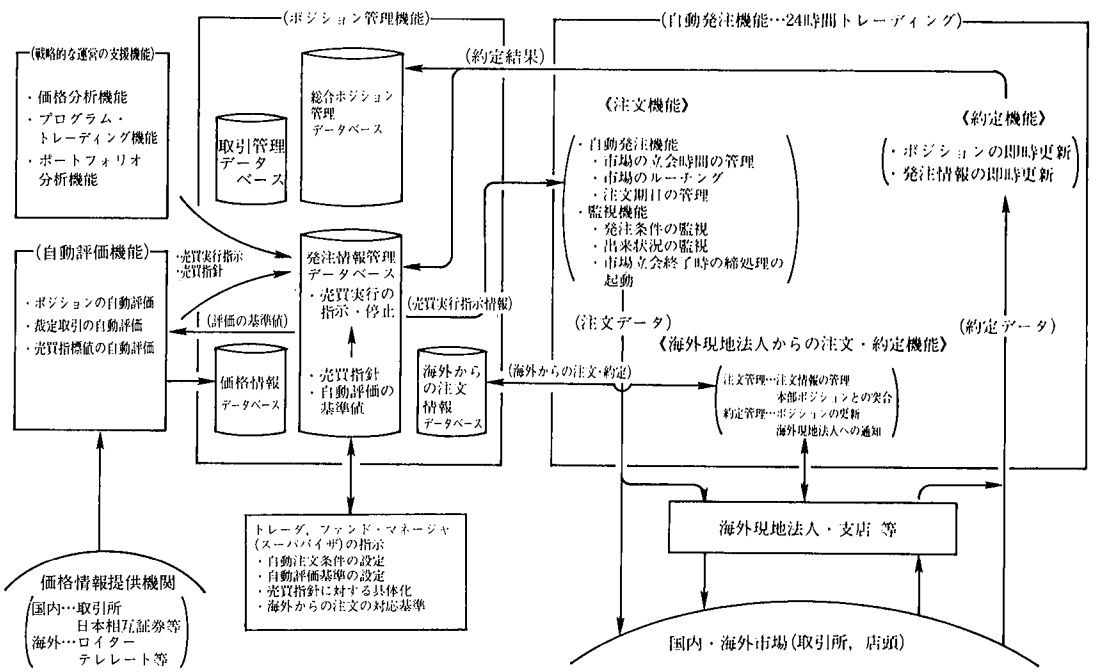


図 6　自動発注機能を中心とした情報の流れ

Fig. 6　Information flow centering around automatic ordering

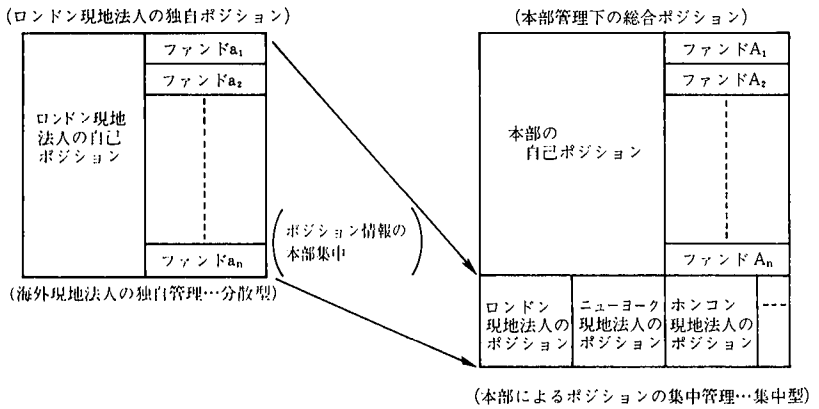


図 7　現地法人独自のポジションと総合ポジションの関連(集中型と分散型の融合)

Fig. 7　Balance of centrization and distribution in position management (companies incorporated abroad vs. head office)

ことがポイントになる.

5.3.1　ポジション管理のパターン

ポジションの管理区分として，大きく次の二つの区分がある.

① 本部管轄分

② 海外現地法人・支店管轄分

この管理方法として，次の四つのパターンがある.

1) 集中型……ブック（元帳）は，本部で一括集中管理され，各現地法人・支店は本部からの運用指示に基づいて行動する.

2) 分散型……本部と現地法人は，各々の所有分のブックを管理し，独自に運用していく．ポジションの相互オファーは売買として扱われる.

3) マーケット特化型……管轄するマーケットに応じて各々ブックを管理する．たとえば円ベースの商品は本部が，ドルベースの商品はニューヨーク現地法人が，マルチカレンシ・ベースの商品はロンドン現地法人が各々ブックを管理する.

4) 日まわり型……ブックは一つで市場の立会時間とともに本部→ロンドン現地法人→ニューヨーク現地法人と持ちまわりされる.

欧米の証券会社においては，日まわりのポジション管理が主流となっているが，国内では集中型と分散型の融合が，実際的な管理の方法として支配的であるようにみうけられる．すなわち，

① 本部への情報集中による統合管理……集中型

② 海外現地法人・支店の独自運用を尊重した分散会計……分散型

の両面からの管理が試行されようとしている．両者の関連を図7に示す.

5.3.2　ポジション管理の体系

証券ポジション管理においては，前述の本部と海外現地法人・支店のポジションの総合管理をベースに

① 新商品の出現（先物・オプション）

② 運用方法の変化（裁定取引・ヘッジ取引・スペキュレーション取引）

③ 国際化への拍車（外国証券取引の拡大）

への対応が円滑に行われる必要がある．これらを考慮したポジション管理は，図8のように体系づけられる.

5.3.3　ポジション管理のポイント

ポジション管理を実施する際には，次のポイントが重要である.

① 運用目的別のポジションおよびネット損益の管理

② 円滑に取引を推進するためのトレーダ別ポジションと，管理を主体とする総合ポジションの層別管理

1) 運用目的別のポジションおよびネット損益の管理……証券取引は新商品の出現（とくに先物とオプション）によって，運用方法に大きな変化がみられる．従来の商品別の取引(買切売切等）から複数商品の同時取引（裁定取引・ヘッジ取引）に特徴があり，今後この複合取引の増加が予想される．これに伴って従

来の商品別のポジションと損益管理に加えて，運用目的別の複合取引のポジションとネット損益の管理が必要になってくる．

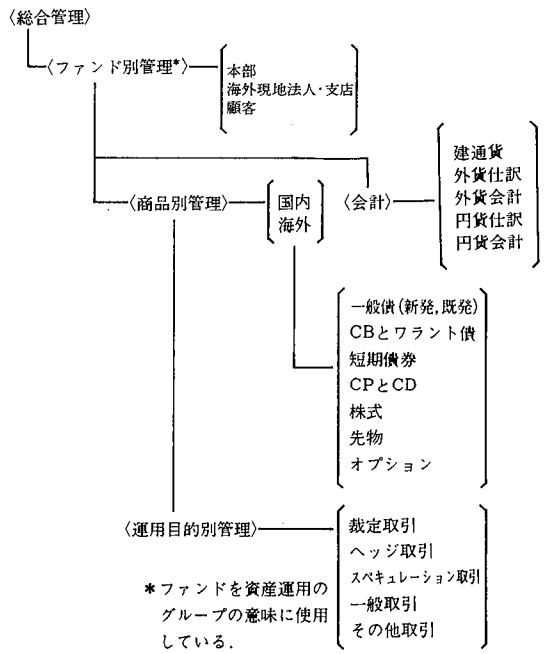


図 8　ポジション管理の体系

Fig. 8　Overview of position management System

2)　トレーダ別ポジションと総合ポジション管理の層別管理……証券取引の運用は，セールストレーダとポジショントレーダに大別されると考える．

①　セールストレーダ：相場の変化を睨みながら瞬時の判断で売買を実施する．ポジションの即時把握が必要になってくる．

②　ポジショントレーダ：全体のポジションを睨みながらセールストレーダへ売買の指示をしたり，セールストレーダの運用成果を評価する．即時性よりも自由な切り口による検索や集計資料の作成（単純集計・二重クロス集計・三重クロス集計）に主眼がおかれる．

このように運用と管理の視点が異なるトレーディングに対して，一つのデータベース構造で対応するには種々の矛盾（効率・情報量等）の発生が予想される．これを解消するためには，総合ポジションとトレーダ別ポジションの層別管理を行い，両ポジション間のリンクを取ることが必要になる（図 9）．

6.　おわりに

現在，注目を浴びているグローバル・トレーデ

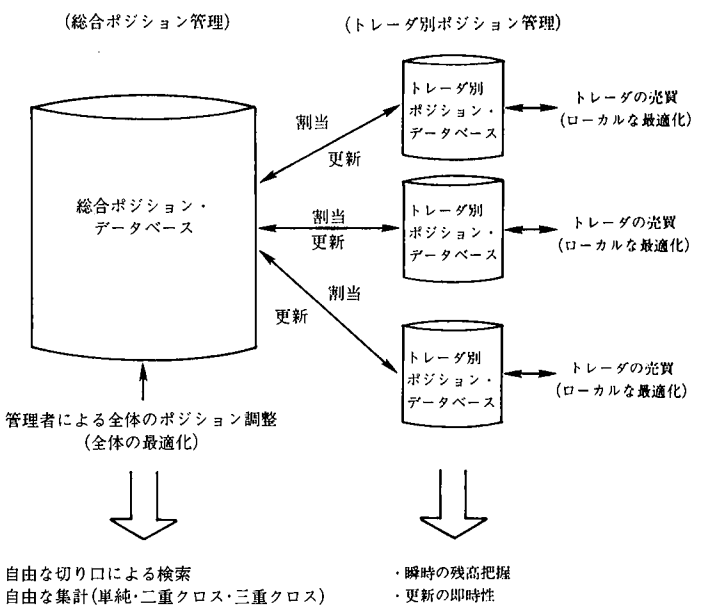


図 9　トレーダ別ポジションと総合ポジションの層別管理の概念

Fig. 9　Layered position management
(consolidated position database vs. individual trader's one)

ィング・システムについて，とくに“証券を中心とした運用”の側面に的を絞って検討を進めてきた．業界を観測するに，グローバル・トレーディングに対するニーズは，予想以上のテンポで高まりつつあり，このシステム化が次期証券系システムの核として各社の大きな課題となっている．このような現状を踏まえ，日本ユニバックにおいても1986年にグローバル・トレーディング・システム検討プロジェクトを発足させ，コンセプトの作成およびシステム化の具体化を進めている．たとえば，Sharpモデルをベースとしたポートフォリオ分析モデルの開発，株式分析モデルの開発，あるいはポジション管理モデルの開発等，すでにいくつかの実績をあげている．

しかし，グローバル・トレーディングとは“運用ノウハウ”が巧拙を支配する業務である．従来の定型業務とは異なり，各社各様の独自性を出した，すなわち各社の個性が反映されたシステムでなければ有用なサポートを得ることはできない．ここにグローバル・トレーディング・システム開発のむずかしさがあり，反面完成時には多くのメリットが期待できる．

したがって，方針としては，一個の商品としてのグローバル・トレーディング・システムの提供ではなく，システム化を早期に実現するために不可欠なプロトタイプ・モデル，コンポネント，ツール群の提供を目標に開発を進めている．ユーザは，プロトタイプ・モデルをベースに独自の運用ノウハウをモデルに反映し，コンポーネントとツール群をアセンブルすることにより，個性を反映した真に有用なグローバル・トレーディング・システムを早期に開発できるものと確信している．

なお，今回は紙面の関係もあって，“証券運用”を中心に記述を進めてきたが，これと同様に重要な課題である資金調達やALMについても，機会を得て報告したいと思っている．

最後に，本稿の執筆にあたって多くの示唆あふれるコメントをいただいた文教大学の栗林訓教授と上智大学の内田晃講師に対してここに謝意を表したい．

参考文献

[1] 栗林訓，ポートフォリオ戦略，東洋経済新報社，1986.

[2] P. J. Kaufman編著，内田晃他訳：金融・為替・商品のテクニカル分析，東洋経済新報社，1987.

[3] LIFFE編，新日本証券債券部訳，金融先物の活用…実践のテクニック，東洋経済新報社，1986.

[4] 金融財政事情研究会編（第10版），金利裁定取引，金融財政事情研究会，1986.

[5] 浅野幸広，ポートフォリオインシャランス，証券アナリストジャーナル（62.6）.

[6] 日興リサーチセンタ，ポートフォリオインシャランス，投資月報（62.11）.

[7] H. M. Markovitz著，鈴木雪夫監訳，ポートフォリオ選択論，東洋経済新報社，1981.

[8] W. Sharp著，小野二郎他監修，日本アナリスト協会訳：現代証券投資論，1983.

[9] シティ・バンク，日本ユニバック共著，ALMシステムの概要，1987.

（金融システム開発1部）

Cliff B. Jones 著

"Systematic Software Development Using VDM"

Prentice/Hall, xvi+300 pp., 1986.

伝統的な工学分野では，設計図や論理回路図のように厳密な仕様を書く方法が存在し，技術者はそれを駆使しているといわれる．ところで，計算機ソフトウェアの分野では，明確な仕様記述が普及していないばかりか，その必要性さえ十分認識されているとはいえない．多くの仕様書と呼ばれているものは，中途半端な図と文章で書かれている．

わが国では，西暦2000年にはソフトウェア技術者が何十万人も不足すると喧伝され，対策が討議されている．技術者であれば仕様を正確に書き，それを読む能力が不可欠であると考えると，そこで論じられている「技術者」は，技術者というよりは単なるソフトウェア従事者にすぎない．

また，このほかにプログラミングを容易にするための処方箋として，「第4世代言語」やAIに対する期待もまだ冷めてはいない．しかし，これは仕様の形式化を通して，正しいプログラムをつくろうとする努力とは方向が異なっている．

1986年以来，本書を数人の仲間と勉強しているが，こうした動向を眺めつつ，形式的仕様がどう役立つかを，本書を読みながら模索しているところである．

本書の筆者は，「まえがき」で，「システムの開発は，どんなものでも，何が必要であるかという仕様をつくることから始めなければならない．このような仕様がなくては開発担当者は，そのシステムの利用者の要求に対して確固たる信念がもてない」と述べている．

ところで，仕様を厳密にそして簡潔に書くためには，形式化するのがいちばんよい．それは，プログラム言語の構文記述のためのバッカス記法を思い出せば十分であろう．多くの仕様記述に役立ち，これほど簡単でわかりやすい方法があれば好都合であるが，仕様はそうやさしくない．そこでVDM（ウィーン開発技法，Vienna Development Method）の出番となる．

VDMによるソフトウェア開発は形式的仕様を作成し，その実現のために証明つきの段階的詳細化を組織的に適用する．仕様から実現へ至る設計のプロセスでは証明義務を生じるが，これに対して証明系を用意している．

もともと，VDMはプログラム言語の仕様記述とその処理系の作成のために，D. BjørnerとC. B. Jonesらが IBM ウィーン研究所で開発したもので，PL/Iの形式的定義に使われたVDL（Vienna Definition Language）を発展させたものである．使用実績があり，方法は洗練されている．Ada処理系の開発に使われているほか，STL（Standard Telecommunication Laboratory）などの産業界で使われており，その報告がある[1][2]．

本書はVDMに関する3冊目で，産業界での教育成果を取り入れ，より組織的系統的に書かれていて，本書のみでVDMを理解することができる．特別な数学的知識は不要で，論理と集合に関する必要事項は本書から学ぶことができる．前提知識がある人なら容易に読み進むことができて，われわれのように足掛け2年もかけて勉強することはないだろう．

著者のJonesは，各節の主要なアイデアは1時間の講義で十分であると書いている．全部で31節あるから演習を考慮しても，大学の講義とすれば1年間といったところか．

本書は，10章から構成され，本文は約260ページである．付録として論理規則，データの性質，証明義務規則，用語説明，記号解説，文献紹介がある．

第1章　論理記法
第2章　証明
第3章　関数
第4章　演算と集合記法
第5章　合成対象と不変式
第6章　写像記法
第7章　列記法
第8章　データ詳細化
第9章　データ型についての補足
第10章　演算の分解

以下，章を追って簡単に紹介する．

第1章で，まず論理に関連する基本概念と記号を学ぶ（23ページ）．

第2章は，「形式化する効用の一つは，正しさに

ついて厳密な議論ができることである」という理由で，sequent を用いた証明について学ぶ（27 ページ）．

第 3 章は関数についてである．課題対象を抽象データ型として把握する場合には，関数の定義方法は一つの基本となる．これには，間接的な定義方法と直接的な定義方法がある．例をお目にかけよう．「自然数の有限集合の中の最大値を与える関数」を両者で記述する．

間接的仕様は以下のとおりである．

maxs（s：set of N）r：N
pre. s≠{ }
post. r∈s∧∀i∈s· i≤r

第 1 行は関数の名前と型，つまり，maxs は，自然数の集合 s から自然数 r への関数であることを示している．第 2 行は前件（pre-condition）で，s が空集合でないことを示し，第 3 行は後件（post-condition）で s に属し，すべての s の要素 i に対して i ≤r であることを示している．

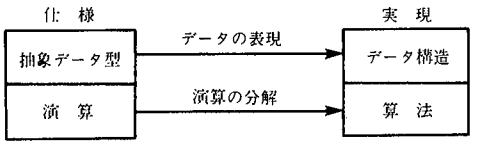
直接的定義は以下のとおりである．

maxs(s)≜let i ∈ s
　　in if card s= 1
　　then i
　　else max(i, maxs(s−{i}))
max(i, j)≜ if i≤j then j else i

let……in……は局所定義であり，card は要素の数を返す関数である．直接定義が間接定義を満たしていることを証明すること，つまり証明義務が発生する．この例のような再帰を用いた直接定義の証明には数学的帰納法を使う（33 ページ）．

第 4 章から第 7 章までは，基本的なデータ構造に対する議論である．まず，集合記法を導入し，集合たちの直積・直和・写像・列・状態機械などを，それらが許す演算とともに解説する．証明義務のために証明系の中に，これらを扱うための公理や推論規則を追加する（36＋27＋27＋30 ページ）．

以上が仕様作成のための部分で，第 8 章からが実現へである．いわゆる段階的詳細化であるが，ここでは reification という用語が使われている．仕様は，集合・写像・列・状態機械とその上で定義された演算によって記述されている．演算を定義する台を，まずプログラム言語が許すデータ構造によって表現する．これがデータの具現（data reification）である．ここでの証明義務は，表現の適切さ（adequacy）で，任意の抽象データに対して少くとも一つの表現が存在することを示すことである．演算は表現の上の演算を連接・選択・反復などによって合成する．ここでは，合成演算がもとの演算になっていることを示す証明義務がある．



第 8 章がデータの具現，第 10 章が演算の分解についてである．第 9 章は補足で，実現の歪みを議論する．歪みとは，演算列によって判別できない異った二つの状態が存在することで，過剰仕様になっている状態である．ここでは，これを避けるためのアイデアを示す（29＋16＋27 ページ）．

本書は，論理や証明に慣れていない読者には，少々とっつきにくいかもしれないが，節ごとに付いている演習問題（残念ながら解答はついていない）に挑戦していけば読了できる仕組みになっている．

英国では，VDM をはじめ，Z, CSP, JSD などの形式的方法に関する講習会の実施や教材の販売が一般プログラマを対象として，政府が後押しする法人で行われている．

日本で，プログラムを安易につくる方法に血道をあげている間に，海の向う側では地道な活動がなされているようである．これは，将来大きな技術格差につながるかもしれない．ご心配の向きに本書をお薦めする次第である．

[1] M. I. Jackson, Developing Ada Programs Using the Vienna Development Method (VDM), SOFTWARE-PRACTICE AND EXPERIENCE Vol. 15, 1985.

[2] D. Bjørner, C. B. Jones, VDM—A Formal Method at Work, Springer-Verlag LNCS No. 252, 1982.

[3] D. Bjørner, C. B. Jones, The Vienna Development Method：The Meta-Language, Springer-Verlag LNCS No. 61, 1978.

[4] C. B. Jones, Software Development：a Rigorous Approach, Prentice/Hall International,1980.

（ソフトウェア生産技術一部　峰尾欽二）

安西祐一郎著

知識と表象
—人工知能と認知心理学への序説—

産業図書，B 5 判，viii＋302，1986，
2100 円．

AI の先駆者の一人である H. Simon は，1980 年に機械の学習に関する優先すべき研究課題を五つ設定し，その第 1 に「コンピュータを使ってシミュレートすることによって，人間の思考過程を理解すること」をあげている．

著者は，Carnegie Mellon 大学留学中に Simon 教授と共同研究を行った経験をもつ，主に認知心理学の立場から 10 年以上人工知能に取り組んでいる研究者である．

本書では，「知識」と「表現」という二つの概念を軸に，コンピュータ・サイエンスと認知心理学の両方から，経験科学としての人工知能研究の考え方について述べている．

本書は，第 1 部「人工知能の基本問題」，第 2 部「認知心理学からの経験」，第 3 部「人間の知性とコンピュータ」，第 4 部「コンピュータ・サイエンスからの経験」の 4 部構成になっている．全体の流れとしては，人工知能の基礎的問題を認知心理学の観点から述べることに始まり，次第にシミュレーションや実現の道具としてのコンピュータの話題に移ってゆく．

第 1 部では，現在までの人工知能研究を振り返り，「インタラクティブな学習機能のための〝知識の構造化可能性〟と〝意味敏感性〟を持つシステムが，並列性と直列性を備えたアーキテクチャの上で，論理的・経験的な知識表現に基づいて構築されること」を将来の人工知能システムの一つのイメージとして描いている．

また最近の脳研究の発展についても触れ，人工知能研究が，生理学を始めとするいろいろな分野における脳の働きの理解への努力と，将来は自然に結びついていくだろうと予想している．

第 2 部では，まず人間の思考の構造・方法・表象の三つの観点から認知心理学のこれまでの歩みについて概略紹介する．次に人間の情報処理機能における知識の果たす役割を取り上げ，知識表象と知識獲得について，物理の問題を学生に解かせた経験をもとに詳細に述べている．

本書の表題にもなっている「表象」(Representation) とは，解釈の過程において新たに構成される情報構造のことである．著者は，「人間は，外界からの情報と自分の知識，とくに自分にとって適切だと思う一つの表象をもとに推論したり，行動を起こしたりする」という仮説を主張している．また，この表象を生成するための知識は，極めて経験的なものであるとし，これを獲得する方法として類推，階層的統合化，文節化をあげている．そしていづれの場合も常に何らかの構造をもって，既有の知識と対応づけて新しい知識を獲得する（知識の構造化可能性）と述べている．

第 3 部では，コンピュータによる文章作成を例にあげて，人間の認識力についての研究の必要性を説くとともに，学習における知識の役割について述べている．つまり，人間は行為や認識に伴う経験を対象化することによって知識を獲得し，その知識に基づいて，ものごとを認識し次の行為を計画していく．学習過程を「変化する知識を中心とした行為と認識の循環サイクル」として，とらえるべきだと主張している．そして「外界を認識する際に，人間が自分の目的と認識の間の関連づけを効果的に行うために，目的に基づいた選択的注意を払っている」という事実から，著者は獲得済の知識によって，外界がいかに敏感に意味づけられるかを述べ，これを「知識の意味敏感性」と呼んでいる．

また著者は，認識研究の方法としてコンピュータ・シミュレーションが使えるようになってからほぼ 30 年たったが，その研究成果は大であると評価している．著者はその意義を，①心を情報処理システムとみなすという見方を与えたこと，②認識に関する多くの機能をバラバラにではなく，統合的に理解するための方法を与えたこと，③論理的基礎のはっきりした研究方法論を与えたこと，④私達に「人間とは何か」についてもう一度見直すきっかけを与えてくれたこと，の四つであったと述べている．

このほか，著者の研究の狙いが，「目標の生成やプラニングなどの〝実験的にすでに確かめられている認識の機能を持つ情報処理システム〟や，〝行動の表出としての画面上の点の軌跡を実際に出力するような情報処理システム〟をコンピュータ・プログラムの形で作り，それによって人間の認識や行動についての一つの説明を与える」ことにあ

ったと述べている．

第4部では，コンピュータ・サイエンスの分野での人工知能の研究を，プロダクション・システムを通じて紹介するとともに，なぜプロダクション・システムが利用され続けてきたかという質問に答え，知識獲得の研究にそれが欠かせない道具であるからと述べている．また，現時点で研究者が使用できる知識表現言語を紹介し，①言語としての構造が単純であるか，②計算効率が高いか，③知識と思われることを自然な形で記述できるか，という観点から評価すべきであると主張している．

そして，考えている問題やその解決プロセスを，オブジェクトとその間のメッセージ交換によって素直に表現できるオブジェクト指向型言語を取り上げ，ニュートン力学の問題解決システムの例を用い，その認知的特徴がどのように生かせるかを検討している．この例は，認知科学の観点からCAIシステムを取り扱った点で，興味深くかつ示唆に富むものと思われる．

以上，本書の概要を述べたが，認知科学からのアプローチは，現在のエキスパート・システムの持っている問題を解決する一つの方向を示しているだけでなく，これまでのコンピュータ・システムに欠けていたヒューマン・インタフェースを実現する上でも非常に役立つと考えられる．本書が，そのための一つの手引として，より多くのシステム開発者に読まれることを期待する．

（知識システム開発部　大野浩史）

ISSN 0289-6257

技　報
UNIVAC TECHNOLOGY REVIEW
No. 16

発	行	日	昭和63年2月29日
編	集	人	柳 生 孝 昭
発	行	人	富 田 和 夫
発	行	所	日本ユニバック株式会社 東京都港区赤坂2-17-51 〒107 TEL(03)585-4111（大代表）
頒	布	価	格	1,500円
印	刷	所	三美印刷株式会社

コンピュータを使わないビジネスなんて、もう思いつかないほどです。暮らしの中でも、いろいろなところでコンピュータは活躍しています。うまくすれば人とコンピュータは、もっといい関係を作っていけるはず。その可能性はビジネスの中にも、暮らしの中にも、どんなところでも無限にひそんでいそう。そういう可能性に向けて私たちは、豊富なシステム構築でつちかったノウハウをベースに、一歩進んだシステム作りを推進します。

企業の個性をシステム化する

UNIVAC

日本ユニバック 東京都港区赤坂2-17-51 〒107 ☎03(585)4111
日本ユニバック情報システム
東京都港区赤坂2-17-22赤坂ツインタワー本館 〒107 ☎03(587)8111