ISSN 0289-6257

技報

UNIVAC TECHNOLOGY REVIEW

1987年5月　第13号

日本ユニバック

静電気放電 (ESD) がコンピュータなどの電子機器に与える影響に関しての研究は，従来そのほとんどが帯電した人体または手に持った工具から筐体やフレームに直接的に放電するという前提で行われてきた．本田昌賀の**静電気スパーク放電による電磁妨害作用の考察**は，帯電した椅子や運搬台車等の金属物体間での静電気スパーク放電によって生じる電磁妨害作用 (EMI) についての実験報告であり，金属物体のアンテナ作用とそれから生じた電磁界の特徴および電子機器に対する電磁妨害作用について分析している．

パーソナル・コンピュータのソフトウェアが単機能処理から，いわゆる統合ソフトウェアと進展すると同時に，ホスト・システムとパーソナル・コンピュータが連携動作する垂直統合システムが求められてきた．佐々木茂の**マイクロ・メインフレーム・リンク用統合操作環境**は，このためのシステム・ソフトウェアのアーキテクチャ，および日本ユニバックのワークステーション DS7 での構成例を紹介している．

アプリケーションの拡大とともに既設の端末と仕様の異なる端末（異仕様端末と呼ぶ）の接続の要求が増大している．庭山宣幸の**画面作成データ・ストリーム変換プログラムの評価**は，両端末間での画面データ・ストリーム変換プログラムの持つべき機能と変換方式について考察するとともに，具体例として同氏の開発した MPC 1100 を紹介している．

発電所のタービンの設置計画やスペース・シャトルの油圧系の設計などでは，複雑なシステムの挙動の理解が求められる．J. F. King らの**知識支援による設計ツール—— ESP** は，このような複数領域にまたがるシステムを取り扱う KAD (Knowledge-Aided-Design) ツール ESP の概要，およびその応用例を紹介している．

最近，ソフトウェアの開発環境においても知識ベース技術の導入が活発化している．S. Pontecorvo らの**ラピッド・プロトタイピングを支援する知的ソフトウェア開発環境**は，ラピッド・プロトタイピング用のワークステーションの設計について述べている．本システムは，Lisp 機械の上で実現され，すべてのソフトウェア・ツールが，いわゆるパイプの拡張としてのソフトウェア・バスを通じて情報交換する方式を採用している．このほか知識ベースによる各種ツールの開発も特徴と言える．

現在，ソフトウェアの需要が供給をはるかに上回っており，今後もその傾向が続くことが予想される．最近この問題の解決策として，再使用可能コード・ライブラリが注目されている．K. L. Bruso の**BIBLIO—再使用可能コード・ライブラリ・システム**は，開発部門で共有可能なコード・モジュール・ライブラリを開発する技法，コード・モジュールを管理するためのシステムについて述べている．

従来，"べた書き入力" 可能な連分節変換によるカナ漢字変換は，汎用機では使用されていたが，パーソナル・コンピュータでは使用されていなかった．吉田正行の**カナ漢字変換システム Micro JASTY の開発**は，シリーズ 1100 のカナ漢字変換システム JASTY の DS7 への移植，および Micro JASTY の使用方法，処理概要について紹介している．

カナ漢字変換をはじめとするカナ文字文の機械処理では，未登録語の処理は同形異義語の処理と並んで重要な研究テーマである．派生語のすべてをカバーすることは不可能なため，接頭語辞書と接尾語辞書の拡充が重要である．新谷隆之の**意味的なつながりを考慮した接尾語辞書の作成**は，受けの成分である名詞接尾語に対して係りの語をその漢字表記の意味で分類し接続の条件として与えた接尾語辞書の作成手順について述べている．

ワークステーション DS7 は，端末としての機能の他にパーソナル・コンピュータとしての機能を併せ持つプロダクトである．阿部比呂志の **DS7 におけるコンカレント CP/M** は，コンカレント CP/M* を DS7 に移植する際の技術的課題，および新たに追加した機能について報告する．

* コンカレント CP/M は，Digital Research 社の登録商標である．

年4回発行に当たって

柳生孝昭

本誌は56年2月の創刊以来6年余りを経，本号を以て通算14冊を数えるに至りました．この間，本誌が有用であるとの評価を頂き，弊社としては些かなりとも社会に貢献し得たことを喜びとしております．これも偏に皆様方の御支援の賜物と心から感謝しております．本誌掲載の論文は情報処理の専門企業である弊社の技術の精髄とも申すべきものでありますが，弊社の事業自体もお陰様をもちまして順調に推移し，技術的挑戦の機会は質・量ともに弥増すばかりであります．このことは技術力を最大の拠り所とする弊社に取って真に本懐と申すべく，併せて厚く御礼申し上げる次第であります．

今，この6年間を振り返って見る時，情報処理技術の日進月歩と利用者の要求の多様化には改めて目を見張る思いが致します．この間における技術動向及び御客様の要求に対する弊社の取組の一端を御紹介致しますと，まず主要な基幹商品の提供については

- 汎用超大型機 1100/90 シリーズ
- 世界初の CMOS VLSI を採用した中型機 2200/200 シリーズ
- 第4世代言語 MAPPER
- 統合 OA システム AOA
- 統合ネットワーク・システム Uai
- 多機能端末機 DS 3 及び DS 7
- グラフィックス，ワーク・ステーションを始めとするマイクロ・プロダクト群
- 人工知能ワーク・ステーション KS-301 及び専門家システム開発支援ソフトウェア KEE
- 電子印刷システム VECTOR 21

等が挙げられます．またそれぞれの業界においては高度の情報システム開発に対する意欲が誠に盛んであり，取り分け

- 金融業界における第3次オンライン・システム及び情報系システム
- 製造業界における CAD/CAM 更には FA システム
- 公共的企業体における基幹業務及び顧客サービス・システム
- 流通業界におけるストア・オートメーション

等が顕著な例でありますが，これらの需要に対し弊社はソリューション・システムの提供と個別開発の受託体制の強化を以て応えて参りました．更にかかる商品及びサービス提供力を増強すべく，またソフトウェア生産技術と生産性の飛躍的向上を期して，明63年には都内江東の地に最新鋭の開発環境を備えた開発センターの設立を準備しつつあります．

さて本誌は，もともと弊社とスペリー社（現ユニシス社）の技術的成果を紹介するために発刊さ

れたもので，読者層としては社内外の情報処理技術者，利用部門の技術者及び，大学，研究機関の研究者を想定しておりました．先に刊行後6年目に当たる第10号発行の折，本誌が御客様を始めそのような読者層の要請に十分応え得ているか否かを改めて反省し，目的により適うべく改良を進めようという機運が生まれ，さまざまな角度からの検討を加えて参りました．その結果，内容の刷新と年4回発行の実施が決定されたのであります．

内容についての新しい試みは，たとえば従来からのコンピュータ・サイエンスやソフトウェア工学的な主題に加えて適用業務面の技術を広く取り上げること，社外の研究者・技術者からの寄稿，また時宜にあったテーマによる特集号の刊行等を考えております．

本号はこの年4回発行の第1号であり，引き続き8，11，2月に刊行して行く予定です．弊社としては今後とも改良を続け，皆様のお役に立つ媒体とすべく努力して行く所存でありますので一層の御愛読を賜わると共に忌憚のない御意見，御批判を御寄せ下さいますようお願いする次第であります．

（日本ユニバック株式会社　取締役）

論文

静電気スパーク放電による電磁妨害作用の考察

EMI Influence of ESD Sparks

本 田 昌 實

要 約 本稿は金属物体間での静電気スパーク放電について，金属物体のアンテナ作用と，そこから生じた電磁界の特徴およびコンピュータに対する電磁妨害作用 (EMI, Electromagnetic Interference) について検討を加えたものである．実験から，いくつかの基本的要因が明らかになったので以下に報告する．

Abstract This paper reports on the EMI influence of ESD (Electrostatic Discharge) sparks between metal objects. These objects act as antennas. From our studies on the electromagnetic fields radiated from the antenna, and their EMI influence on a computer, we have found that three fundamental factors are critical.

1. はじめに

ESD が電子機器（たとえば，机上のワークステーションやパーソナル・コンピュータ）に与える影響に関しての研究は，従来そのほとんどが帯電した人体または手に持った工具からコンピュータの筐体やフレーム等への直接的かつ直撃的な放電が行われる状況を前提としてなされてきた．ところが，われわれは実際のオフィスにおいて帯電した椅子や運搬台車等が，コンピュータではなく他の金属物体に接触したときに，コンピュータが誤動作を起こすことを何度か経験している．

一般的に，よく検討の対象とされる伝導電流波形の観測と RLC 集中定数回路網の解析だけでは，このような障害を十分には説明できない．むしろこの問題は，このとき発生した妨害エネルギがどこのいかなる導体も経由せず，空間を介して電子機器内の回路に結合する問題であると考えざるをえない．これを，以下では間接 ESD と呼ぶ．

そこで，われわれは間接 ESD の状況とその影響を確認するため，予備的な実験として図1に示すようなスチール・パイプ部分を有する椅子の接触でパソコンが影響を受けるか

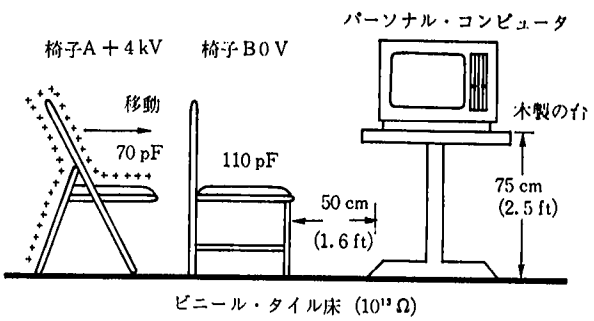


図1 間接 ESD による EMI の発生

Fig. 1 Generation of EMI due to indirect ESD

否かを試みた．その結果，＋4 kV に帯電した椅子A（静電容量 70 PF）が椅子B（静電容量 110 PF）に衝突した瞬間，そこから 50 cm 離れたところにあるパーソナル・コンピュータでフロッピ・ディスクの“READ PARITY ERROR”が発生することを確認した．あきらかに，この椅子から妨害波が出たことがわかった．

間接 ESD の問題をより明確にするため，われわれは金属物体の放射アンテナ作用，そしてそれによって生じる電磁界の時間，空間および周波数領域での解析を必要とした．以下では，そのために行った実験について述べる．

2. 実験方法の概要

2.1 実 験 方 法

帯電した物体として金属製の椅子が金属でできているもう一つの椅子に接触し，放電する場合を想定して一連の実験を行った．通常のオフィスにおいて，これらの椅子はゴム製のキャスタ，床表面のカーペット，ビニール・タイル等によってグランド面（大地）から絶縁されている．

したがって，これら物体の相互間，そして机上に置いてあるパーソナル・コンピュータとの間に直接的な伝導電流を導びくパスはなく，また電荷を蓄えている椅子（キャパシタ）と他の物体との間に“グランド・リターン”も存在しない．

われわれは，このような状況をシミュレートするため，

1) ESD 発生源として2本の銅パイプからなる放電治具
2) 電磁波の観測装置としてオシロスコープ（時間領域），スペクトラム・アナライザ（周波数領域），および ESD 検出器
3) 被妨害電子機器としてパーソナル・コンピュータ

などを用いて，空間を伝搬するすべての電磁波を考察対象として実験を行った．

2.2 ESD 発生源（放射アンテナ）

図2は，グランド・リターンのない ESD 発生源として垂直に吊るした長さの等しい2本の銅パイプから構成される放電治具を示す．銅パイプ(A)の荷電はパイプ直上にある荷電用導電ゴム・パッドを介して行われる．ESD の発生は，この銅パイプ(A)を吊っているテフロン®・ロープを離して自然落下させ，下側の 2.5 cm 離れたところに固定されている銅パイプ(B)に衝突させることにより行う．この時の衝突速度は約 70 cm/sec である．

直流高圧の引きまわしは，すべて導電ゴム（500 kΩ・cm）を用いた．銅パイプ(B)の下部からは非帯電性のプラスチック材料をグランド・プレーンに垂らし，電荷の緩慢な漏洩を行っている．

この2本の金属導体から構成される放電治具は，ESD に際して電磁波を周囲に放射する垂直ダイポール・アンテナとして作用する．なお，銅パイプに至る導電ゴムからの電磁波放射のないことを，ESD 検出器を用いて確認してある．

2.3 電磁波測定方法

この放電治具において，毎秒1回の割合で計 200 回の放電を発生させる．ここから 2m のところに，対数周期アンテナを置き垂直偏波成分を受信する．このアンテナの誘起電圧をスペクトラム・アナライザに入力し，周波数分布と強度を内部メモリに累積し記録する（図3）．

また，別の方法としてコンピュータ内部に侵入した電磁波の様子を調べるため，厚みが 0.15 mm でできている銅の箱（H 42×W 30×D 85 cm）を用意し，これを先の ESD 発生

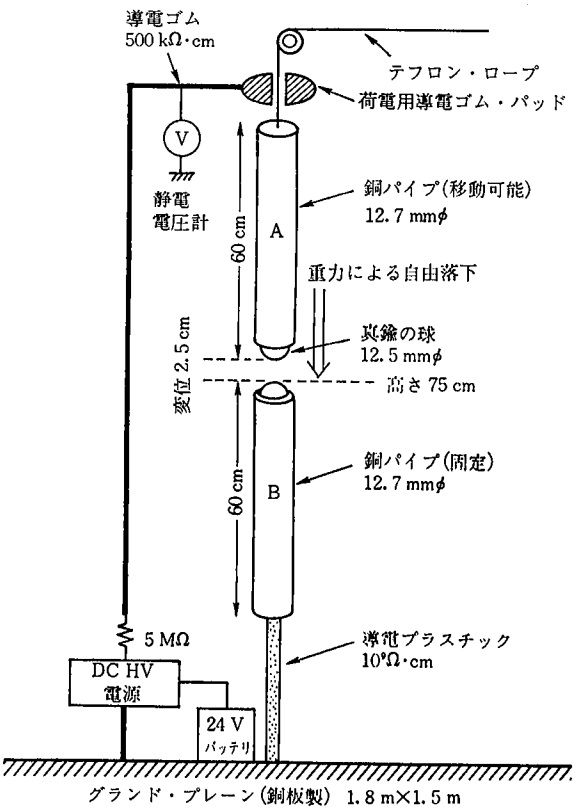


図 2 グランド・リターンなしの ESD 発生源

Fig. 2 ESD generation source without ground return

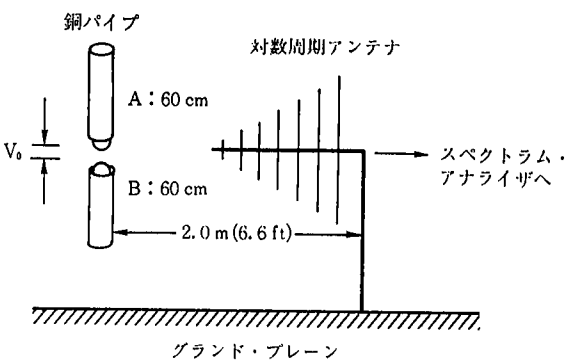


図 3 放射電磁波スペクトルの計測法

Fig. 3 Radiated electromagnetic field measuring method

源から 50 cm のところに置き，幅 20 mm の編組線でグランド・プレーンに接地した．この箱の開口部 (8×28 cm) を銅パイプに対向させ，内部中央部に長さ 10 cm のモノポール・アンテナを上部から垂直に垂らし，先と同様の方法で周波数分布と強度をスペクトラム・アナライザ* の内部メモリに累積し記録した(図 4)．

2.4 パーソナル・コンピュータ・システム

本体と同一キャビネット内にあるフロッピ・ディスク間においてデータの受け渡しを行

* 対数周期アンテナを接続しているスペクトラム・アナライザの管面の dBm 表示は，このとき用いたアンテナ固有の利得 (Antenna Factor) によって補正して読まなければならない．

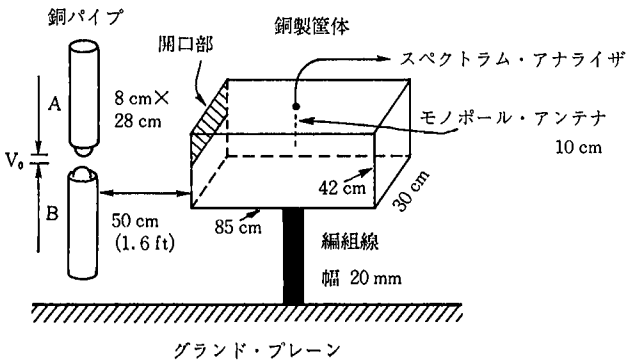


図 4　キャビネット内のスペクトル分布

Fig. 4　Electromagnetic field distribution inside a cabinet

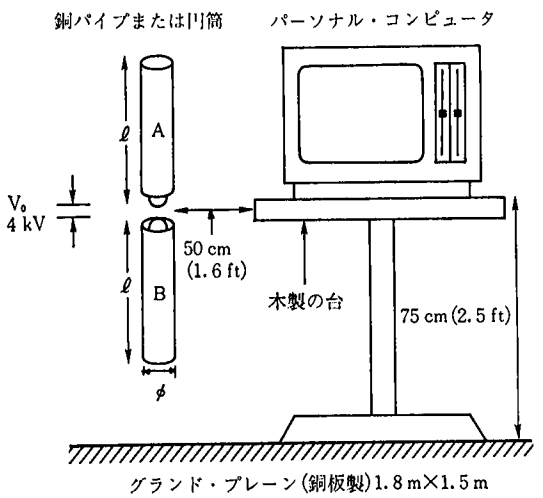


図 5　間接 ESD による EMI の発生

Fig. 5　Generation of EMI due to indirect ESD

い，その処理時間およびデータの内容が，期待した値と異るか否かをプログラムで絶えず監視している．ESD に同期してこのシステムでエラーの発生が検知されたとき，システムが ESD によって干渉を受けたものと解釈した．このシステムは，高さ 75 cm の木製台に置いた．システムの AC 電源は通常用意されている壁の出力コンセントからとり，接地は 1.8 m×1.5 m の銅板製グランド・プレーンに接続した（図 5）．

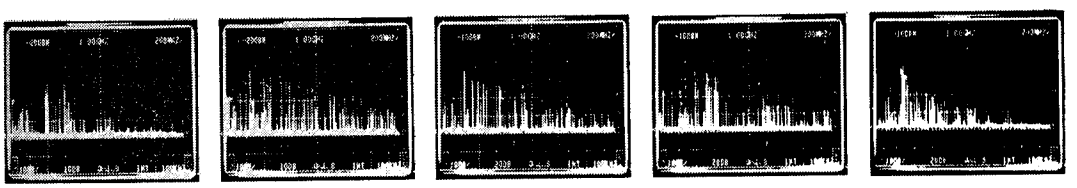
2.5　主要計測機器

1) スペクトラム・アナライザ
   TEK 494 P 21 GHz（外部ミキサなし）
2) オシロスコープ
   TEK 7104 メインフレーム DC ～1.0 GHz
3) 対数周期アンテナ
   ANRITSU MP 636 A 300 MHz～1.7 GHz
4) ポータブル ESD 検出器
   SANKI ES-81 V SINGLE PULSE <1 ns

3. 測 定 結 果

3.1 帯電電圧と放射スペクトラム

長さ 60 cm で自由空間に対する静電容量が，9.0 PF の大地から浮いている状態にある銅パイプにおいて，帯電電圧 (V_0) をパラメータにとり，スペクトラム強度（パワーレベル）と分布がどう変化するかを調べた（図 6）．その結果，図 6 に示すとおり，遠方界（放射項）で受信した放射スペクトラムは非常に広帯域であることがわかった．そして，放射パワーは電圧の上昇と共に大きくなるが，13 kV 程度で飽和し，帯電電圧 20 kV ではパワー・レベルの低下をきたすと共にスペクトラム分布も低域の方向に移行していることがわかった．この傾向は，われわれが過去に行った実験結果[1]ともよく一致する．



V_0: 1 kV　　3 kV　　6 kV　　10 kV　　20 kV

図 6 放射スペクトラムと帯電電圧 (200 MHz/DIV, 10 dBm/DIV)

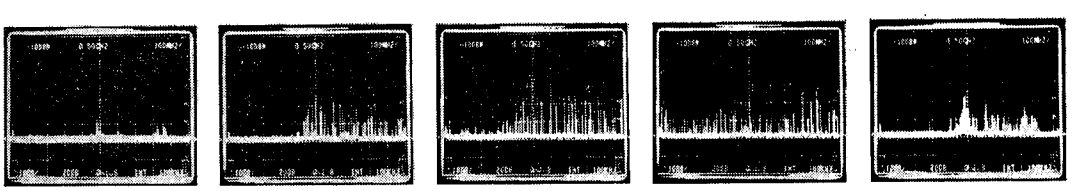
Fig. 6 Radiation spectrum vs. charged voltage (200 MHz/DIV, 10 dBm/DIV)

3.2 帯電電圧とキャビネット内の電磁波

先と同じ銅パイプの帯電電圧 (V_0) をパラメータとし，キャビネットの内部における電磁波のスペクトラム分布を調べた（図 7）．その結果，図 7 に示すとおり 480 MHz 附近に強いスペクトラムが認められ，侵入した電磁波のパワー・レベルとスペクトラム分布は，必ずしも電圧と比例関係にないということがわかった．帯電電圧 9 kV～11 kV でスペクトラム分布とパワー・レベルとも最大となり，それより電圧を上げるとスペクトラム分布とパワー・レベルとも減少の傾向を示し，帯電電圧 20 kV の状態では 2 kV～3 kV の値しか得られなかった．また，帯電電圧と電磁界エネルギとの関係は図 8 で示される．なお，この実験において，このキャビネットを空洞共振器とみなした場合の最低時の共振モード (TE 101) 周波数は，約 610 MHz である．

3.3 物体の形状と EMI

形状の異なる種々の物体間における ESD に際し，パーソナル・コンピュータ・システムがどのくらい妨害を被るかについて実験を行った．この場合，放射アンテナとしての物体の垂直軸に関しての対称性の良し悪しは，ただちにこのアンテナの放射指向性 (BEAM PATTERN) に影響を与えるため，供試物体としては円筒形のものを使用せざるを得なかった（図 5）．この結果表 1 に示すとおり，パーソナル・コンピュータに対する EMI は



V_0: 1 kV　　3 kV　　6 kV　　10 kV　　20 kV

図 7 筐体内の電磁波 (100 MHz/DIV, 10 dBm/DIV)

Fig. 7 Electromagnetic fields inside the cabinet (100 MHz/DIV, 10 dBm/DIV)

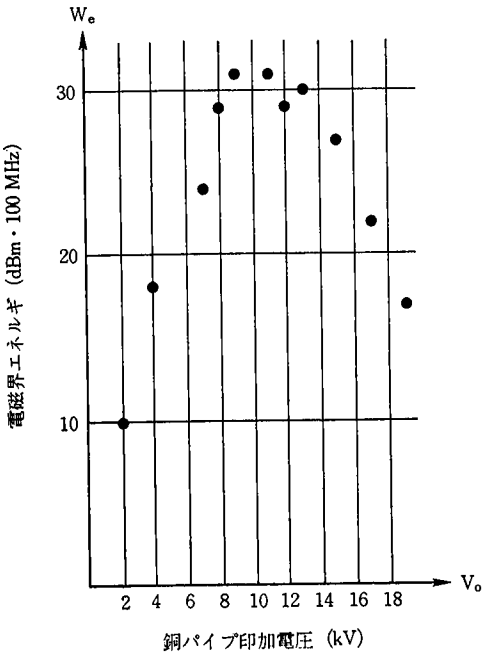


図 8 帯電電圧と電磁波のパワー・レベル

Fig. 8 Electromagnetic energy vs. charged voltage

表 1 物体の形状と EMI の関係

Table 1 EMI vs. shape of object $V_0=4.0\,\mathrm{kV}$, $d=50\,\mathrm{cm}$

物体の形状	長さ(l) cm	静電容量(C) PF	パーソナル・コンピュータの状態
銅の網でできた円筒 $\phi=65\,\mathrm{mm}$	20	7.0	まったく誤動作しない
銅アミ円筒 $\phi=65\,\mathrm{mm}$	25	7.5	たまに誤動作する
銅パイプ $\phi=12.7\,\mathrm{mm}$	30	4.5	しばしば誤動作する
銅アミ円筒 $\phi=65\,\mathrm{mm}$	30	8.0	いつでも必ず誤動作する
銅パイプ $\phi=12.7\,\mathrm{mm}$	60	9.0	いつでも必ず誤動作する

長さと太さが支配的であり，物体の静電容量にはよらないことがわかった．すなわち，妨害作用は物体に蓄わえられている静電エネルギ ($W_0=CV_0^2/2$) に比例していない．

4. 実験のまとめ

4.1 実験結果の要約

1) 金属物体の接触に際して，電荷を受けとる側の物体にグランド・リターンが存在しなくても，スパーク放電が起り電荷が移動する．そして，このとき強力な妨害波が物体の周囲の空中に放出されることがある．

2) 金属物体から放出される妨害波の周波数スペクトラムは，電圧が高くない状況のもとで最大 2 GHz を越えることがある．また，スペクトラムのパワー・レベルと分布は放電寸前の物体の帯電電圧には比例せず，電圧が高まると放射スペクトラムは低域へ移行する．

3) 物体の放射アンテナ作用を増長するパラメータは長さと太さであり，静電容量の大小にはよらない．

4) 金属性のキャビネット（ケース）内部に侵入してくる電磁波のスペクトラム強度と分布は，帯電電圧に比例しない．しかし，キャビネット固有の共振モードで外部から入ってきた電磁波を，より強調する作用がある．

5. 考　　察

われわれが行った実験は，間接的な ESD によって電子機器が影響を受ける状況を再現したものであり，事実そのことが確認された．ここでは，なぜこのような金属物体間のスパーク放電によって電子機器が影響を受けるのかについて検討しよう．

5.1 帯電した金属物体

接地されていない金属物体は多くの場合，人体よりも効率よく，しかも整然とした状態で静電気を蓄えている．とがったところがなければ，放電寸前まで電荷は減ることもない．床および絶縁性のゴム車輪の抵抗値は，10^{13}～$10^{16}\,\Omega$ もある．人体の静電容量は，これら物体の2倍から10倍もあるが，皮ふ表面の抵抗や衣服にさえぎられ整然と分布していない．帯電した金属物体は人体に比べて，よりエネルギの集中が大であると言える．

5.2 スパーク放電（高速スイッチ）

物体の接触に際し，銅パイプの表面に整然と並んでいた電荷は一瞬に相手側に乗り移る．この動きを邪魔するのは，スパーク放電のアークに存在する抵抗成分（または空気の瞬間導電率）だけである．この高速スイッチに存在する抵抗が大きいと，帯電エネルギの大部分は熱や光，音に転換され，しかも放電電流波形はなまってくる（図9，図10）．

この OFF-ON-OFF 動作を行うスパーク放電のスイッチ作用による電流の立上りの速さは，電圧に反比例することがわかっている[1]．

このような CR 放電回路に現れる波形は指数函数で表され，アークの抵抗が少ないほど，それは理想インパルスとみなせる．そして，この波形の立上り部分における高周波電流（伝導電流と変位電流）の周波数成分は，非常に広い範囲にわたることになる（図11）．

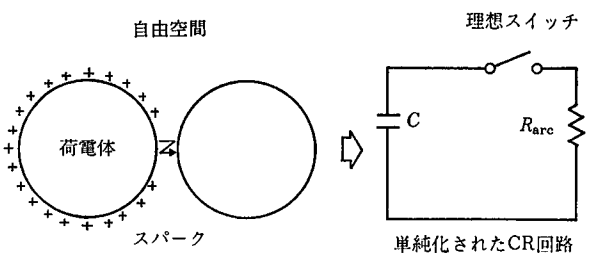
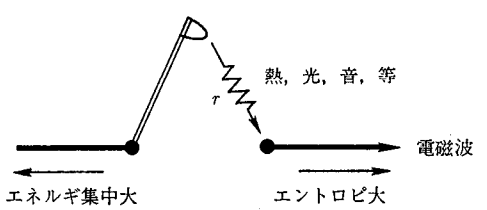


図 9 放電スパークと絶縁金属体の等価回路

Fig. 9 Discharge spark and equivalent circuit of an isolated metal object



r：アークに含まれる抵抗

図 10 高速スイッチとしてのスパーク放電

Fig. 10 Spark discharge as a high speed switch

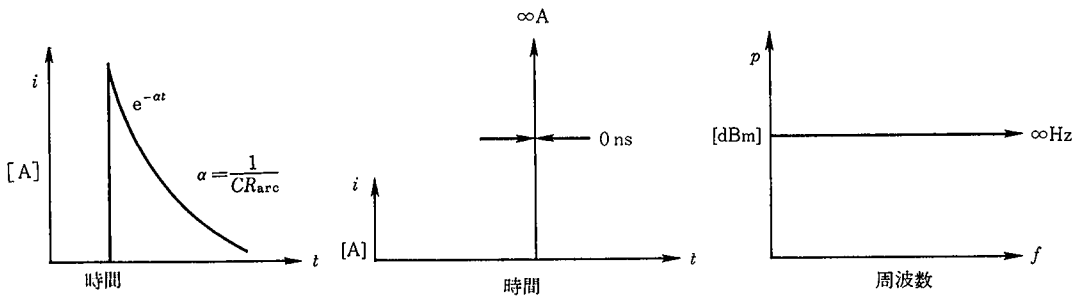


図 11　理想インパルス

Fig. 11　Ideal impulse

5.3　放射アンテナとしての金属物体

高速スイッチとしてのスパーク放電によって 2 GHz を超える高周波電流が生じ，この電流が太くて表面積の大きい金属物体を駆動することにより，広帯域の放射アンテナが形成される．アンテナの実効長や放射効率を決定する放射抵抗と空中線抵抗等は，一般に駆動電流を連続波（繰り返し周期関数）として扱っているため，スパーク放電のごとく，非繰り返し電流がこのような金属物体を駆動した場合の解析は非常にむずかしい．しかし，帯電電圧 (V_0) が 11 kV のとき，距離 2m の点で平均電界強度で 100 dBμV もの値が得られたことは脅威である．なぜならば，この値は通常のオフィス・コンピュータに対するインパルシブ EMI 限界に極めて近いからである（図 12）．

さらにまた，金属物体から放出される妨害波は，主として電界（E）が放射される場合と磁界（H）が放射される場合がある．これは金属物体の大地との位置関係，アンテナ給電点としての物体相互が接触する部分，そして放射エレメントの形状（ループ状か棒状か）等によって決定され，電子機器のキャビネットに対する遮蔽を考える上で，近傍界の問題とならんで非常に重要な，またむずかしい事柄となる．

5.4　至近距離での ESD

コンピュータの至近距離で発生する間接 ESD の方が，直接 ESD よりも強い EMI 作用を呈する場合がある．電界や磁界の形で電子回路に飛び込んだ時の方が影響度合が大きいということは，この“界の姿”が通常とは異なるのではないかとの考えに立ち，これをオシロスコープによる時間軸の観測で検証した．その結果，図 13 の構成において放電をおこしたこのパイプのごく近傍には，図 14 に示すように極性を有するパルス状の電磁界

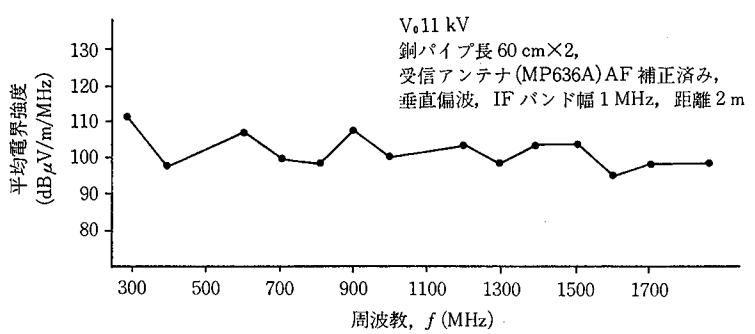


図 12　ESD 源から 2 m の地点での電界強度

Fig. 12　Field strength at a point of 2 m from the ESD source

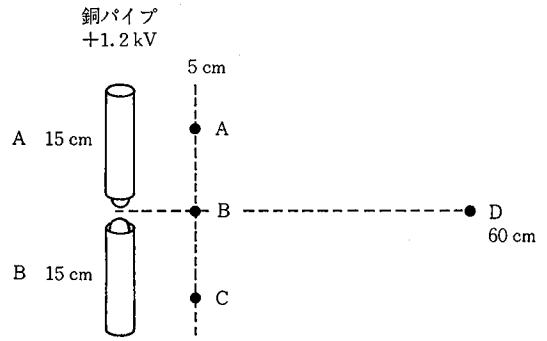
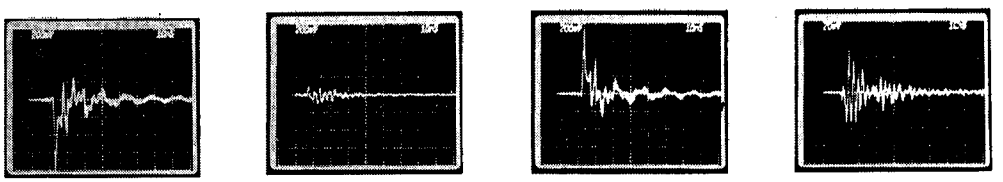


図 13 ESD 源の近傍の電磁界

Fig. 13 Electromagnetic field in the near-field region



A点 200 mV/DIV　B点 200 mV/DIV　C点 200 mV/DIV　D点 20 mV/DIV

図 14 各測定点におけるオッシロ・スコープのスパイク波形

Fig. 14 Osciloscope waveform spikes at various points (10 ns/DIV)

(極性界) が生じていることがわかった.

発生源から遠ざかるに従い，この電磁界は振動するようになり，振幅も急激に減少する．コンピュータが ESD 発生源から 60 cm 未満の範囲にあるとすれば，80 MHz 以下の成分を持つ電磁界は誘導項が優勢となり，距離の 2 乗に反比例して電磁界が強まることになる．この誘導項の働きは，当然のことながら周波数には関係がない．コンピュータの EMI 対策としての遮蔽方法を考える場合，このような ESD による近傍界の問題は強力な磁界の存在と相まって手ごわいものとなり，決してあなどれないものとなろう．

6. おわりに

本稿での実験によって，次の事実が確認された.

1) 大地から絶縁された状態にある金属で作られている椅子や台車は，ESD に際して良好な特性を持つ広帯域放射アンテナとして作用する.

2) 金属物体間のスパーク放電は，アークに含まれる抵抗以外に電流を制限するものがないため，放出された電磁波は非常に広い範囲にわたる周波数分布を有する.

3) 電子機器に対する直接的な ESD ではないにもかかわらず，発生源近傍にあるコンピュータは確実に影響を受ける．その理由は，極性を有する急峻な変化分を持つ電界と磁界が直接電子回路に結合するからである.

4) 物体に蓄えられた静電エネルギの大小と EMI 作用は比例関係にない.

さて，これらの事柄を個々の問題ごとにとらえるのではなく，全体を統合した観点から眺めると，ESD は次の三つの基本的要因を合せ持つということになる.

1) 単発現象 (非繰り返し) であり，するどい指数変化を持つ.

2) 超広帯域周波数分布を持つ.

3) 電子機器に対する近傍界 EMI の問題を持つ．最後になったが，著者はこれらの事

柄を考慮した上で，はじめて実際的かつ有効な ESD 対策が行えるものと信じている．

参考文献 [1] M. Honda, T. Kawamura "EMI Characteristics of ESD in a Small Air Gap", The Proceedings of 6 th Annual EOS/ESD Symposium, Oct., 1984, pp. 124〜130.
〔日本語訳："狭間隙状態における静電気放電の電磁妨害特性——ARP が EMI を決定する"，技報，日本ユニバック(株)，No, 9, 1985, pp. 32〜43.〕

執筆者紹介 本 田 昌 實 (Masamitsu Honda)
昭和 18 年生．40 年北海道大学 工教 電気科卒業．同年日本ユニバック(株)入社，大型計算機の保守に従事．その後，稼働環境，とくに計算機に与える電磁妨害，静電気障害対策の調査・研究に従事．現在，商品開発本部ハードウェア・プロダクト・サポート 4 部サポート・エンジニアリング室所属．
静電気研究懇談会会員，静電気学会会員，電子情報通信学会会員，IEEE 会員，米国電子戦学会 (AOC) 会員，米国 EOS/ESD 協会会員．

論文 マイクロ・メインフレーム・リンク用統合操作環境

The Integrated Operational Environment for Micro-Mainframe Link

佐々木　茂

要　約　ホスト・システムとマイクロ・システム（パーソナル・コンピュータ等）の有機的な結合（データの相互利用と互いのシステムの特性を生かした分散処理）を，マイクロ・メインフレーム・リンクと言う．

パーソナル・コンピュータのソフトウェアが，単機能処理プログラムから複数の単機能処理プログラムが連携する，いわゆる統合ソフトウェア（内部統合システム）へと進展すると同時に，ホスト・システムとパーソナル・コンピュータをマイクロ・メインフレーム・リンクで結んだ垂直統合システムやパーソナル・コンピュータ同士を LAN で結んだ水平統合システムが求められてきている．

統合システムの要件としては，ユーザ独自の運用形態に応じられるだけでなく，既存のシステムとの親和性が高いことが必要である．

本稿では，これらの統合システムを支援する環境を「統合操作環境」と呼び，とくに垂直統合システムとの関係を中心に同環境の構築法について報告している．

本稿の「統合操作環境」は，次の5階層で構成されている．

1) 実行レベル……処理ユニット(単機能処理プログラム)，あるいは既存プログラム
2) オペレータ・インタフェース・レベル……統一的オペレータ・インタフェース．処理ユニットとリンクされ処理プログラム（システム）を構成する．
3) 制御レベル……コマンド処理プログラム．コマンド列で与えられた一連の処理プログラムの実行を可能にする．
4) 動作環境……マイクロ・メインフレーム・リンクに関する処理を行う階層で，垂直統合システムではアプリケーションから見ると，これ以下は OS と考えてよい．
5) OS……ワークステーションのオペレーティング・システム

そして，処理プログラムは，既存（もし，必要ならば新たに開発した）オペレータ・インタフェースと処理ユニットを実行時にリンクすることによって構成される．

なお，本稿では，「統合操作環境」の構築例として日本ユニバックのワークステーション DS 7 での実現例を紹介している．

Abstract An organic connection of the host system and micro system which provides distributed processing allowing the mutual uses of the data and the features of each processing system, is called micro-mainframe link.

As the software on the micro system evolves from the single-function program into the integrated software, the need of the vertically integrated system connected by micro-mainframe link and the horizontally integrated system connected by LAN has become pressing.

This paper reports the architecture and construction method of the environment called the "integrated operational environment" which supports all types of the integrated systems.

The integrated operational environment is configured of 5 layers below.

1) execution level……Processing unit, namely, modular component of a processing program (system) or sometimes the existing software itself

2) operater interface level
3) command processor level
4) operational environment core……The widely defined OS for the vertically integrated system begins from this layer
5) OS……Operating System on the work station.

A processing program is configured by linking an operater interface and processing units dynamically. And the programmed execution of a series of processing programs are facilitated by the command processor.

The paper introduces the integrated operational environment implemented on the work station DS 7 to show the effectiveness of our approach.

1. はじめに

最近のターミナル・システムやパーソナル・コンピュータは，文書処理や表計算などの機能を備えており多機能化し，ワークステーションと呼ばれるものが増えてきた．ホスト・システムに接続されていた従来の専用ターミナルの機能は，通信機能をエミュレートする，これらのワークステーション上の一つのプログラムに置き換えられつつある．

最近のパソコン・ソフトの一つの方向として，その代表的ソフトであるワープロ，表計算，グラフ処理プログラム間に共通のファイル構造やデータベースを持たせ，操作にも統一性を備えた「統合ソフト」(内部の統合) への進展がある．また，LAN によるパーソナル・コンピュータ間での横方向の統合（水平統合）も進んできている．

ホスト・システムとの関係においても，この統合化傾向は例外ではない．ホスト・システムのデータベースに蓄積されたデータのワークステーションでの利用，逆にワークステーションで収集したデータのホスト・システムによる処理など，データの共有化が進んでいる．しかも，データ，文書情報，イメージ情報などに対するワークステーションでのマルチメディア処理機能を生かした，ホストとの連携による情報の分散処理が求められてきている．このような縦方向の統合（垂直統合）をホスト・システムとワークステーションの結合で実現することをマイクロ・メインフレーム・リンクと言う．

以上，三つの統合化，すなわち，内部，水平，垂直の統合が求められるに至った現在，これらに対応できるワークステーションの実現が急務となっている．

ワークステーションは，パーソナル・ユースが基本である．マウスやマルチ・ウインドウに代表されるような使いやすいユーザ・インタフェース，すなわち操作環境の提供がその目的の一つである．先に述べたパソコン上の統合ソフトは，操作性を追及した一つの形態である．しかし，それ自体は限られた領域の統合であり，データの共有はあらかじめ定められた数個のプログラム間に限られ，使用者が作成したシステムや，任意の市販のソフトとの融合や併用はできない．

一方，前述したように，ワークステーション上のシステムとホスト・システム上の既存のユーザ独自のシステムとの垂直統合が必要となってきているが，現状ではホストとワークステーションというように思想の異なる二つの処理系の操作の修得が必要となり，両者を使い分けるには，使用者にとって相応の負担と時間が強要される．

そこで，ワークステーションにおけるワープロや表計算などの処理システムの操作中にホスト・システムを意識することなく，その延長としてホストのデータベースのアクセスができること（マイクロからの統合)．また，逆にホストから，ワークステーションの処理システムを起動し，処理を分担する形態が統合システムに求められる（ホストからの統合)．

これらの点を考慮した上で，ワークステーション上の統合システムを構築する技法につ

いて以下に述べる．

2. 統合システム構築に当たっての考え方

統合システムの設計概念は，「統合システム」のとらえ方から出発する．ここで，統合とは，①操作の統一と，②データの分散処理と相互利用，を意味する．

統合システムは，既存の処理システム間の操作を統一し，より効果的なデータの分散処理を行うことである．言い換えれば，既存の個々のシステムをベースに構成されるべきものである．この点から，統合システムでは現状の処理システムにスムーズに融合し，より操作性の優れたシステムへ移行できる構築法が必要となる．また，システムは，業務の処理内容，手順の改善など，運用形態の変化につれ変わる．統合システムの構築法では，これらの変化に柔軟に対応できることも求められる．

図1は，ワークステーションから見た統合システムの構成である．作業の能率や難易度は，その環境に大きく作用されるので，統合システムを構築するには環境の整備が重要となる．そして，この統合システムのための環境を「統合操作環境」と呼ぶ．

「統合操作環境」の目的は，統合システムをユーザが容易に構築できるようにすることである．「統合操作環境」は，図1に示すように既存のユーザ・ソフトや市販のプログラムも含めた，ワークステーション上で実行される処理システム（プログラム）を統合システムの一つの構成要素として包括するための機構であり，5階層で構成されている．

2.1 各処理システムの単機能化*と役割別階層化

一般に処理システムは，複合機能を備えているため内部的には互いに同一の処理を含むことになる．たとえば，図2の処理システムA，処理システムB，処理システムCでは共通の処理（斜線部）が含まれる．このことは，動作時の占有メモリの増大，操作の不統一を引き起こすばかりでなく，重複開発によるコストの増大，保守性の低下となって現れる．そのため，統合システムの構築に当たっては，重複を排除するアプローチである「処

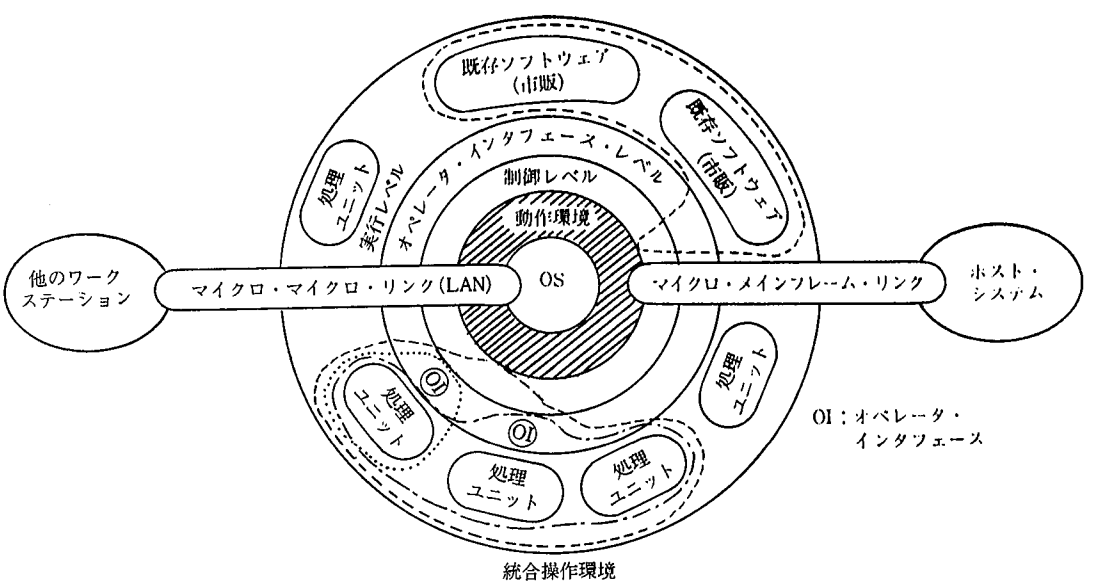


図 1　統合操作環境

Fig. 1　Integrated operational environment

* 処理システムを単機能の処理ユニットに分割することをいう．

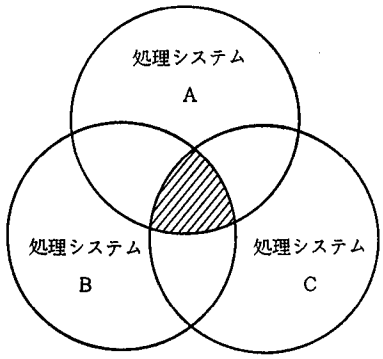


図 2　従来のプログラム構成

Fig. 2　Traditional program strucuture

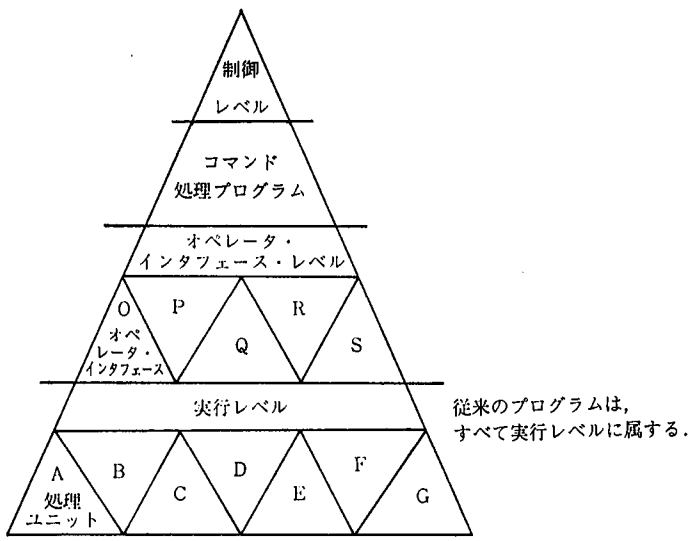


図 3　統合操作環境下のプログラム構成

Fig. 3　Program structure on integrated operational environment

理システムの単機能化と役割別の階層化」が最初のステップとなる．図3にその役割別の階層を示す．実行レベルのプログラムは，単機能化した処理ユニット（プログラム）の集合である．それぞれの処理ユニットは，単一の処理を行い，それ自体，単独でも実行可能である．

統合システムでは，同一処理に対しては同一操作で実行できることがとくに求められる．また，既存システムとの連動には，それらのシステムと差異のないよう，あるいは，より簡便な操作法に切り替えることが操作性の向上につながる．このための役割を果たす階層が，図3に示すオペレータ・インタフェース・レベルである．オペレータ・インタフェース・レベルのプログラムは，複数の処理ユニットと動的にリンクされ，組み立てられ，一つの処理システム（例：従来のパソコン・ソフトの単独処理プログラムなど）を構成し，統一的操作インタフェースを提供する．

制御レベルのプログラムは，これらオペレータ・インタフェース・レベルや実行レベルのプログラムを任意に選択統合し，ターン・キー・システムや複合機能を備えるユーザ・システムを構成する．

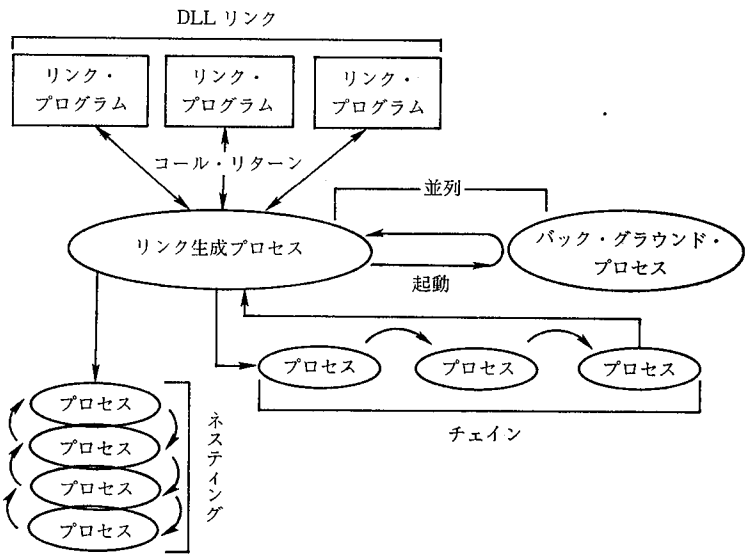


図 4　プロセスのリンク形態

Fig. 4　Process linking varieties

2.2　処理ユニットのリンクと処理システムの構成法

単機能化と階層化の次に必要となるのが統合化技法である．単機能化は，統合ソフトのファミリ化・一体化と反対方向にあり，操作性を高めるには単機能化した処理ユニットを結合し，一体化しなければならない．前述したように，オペレータ・レベルや，制御レベルのプログラムが，実行レベルのプログラムと動的（実行時）にリンクされ，同一処理に対しては同一プログラムを動作させることにより処理システムを構成する．このリンクには，使用形態に応じ図 4 に示す四つの形態，ネスティング，DLL (Dynamic Loading & Linking)，プロセス・チェイン，並列がある．

2.3　動　作　環　境

一般に，オペレーティング・システム (OS) は複数のアプリケーション・プロセスを並列動作させるために必要な同期，資源管理，入出力管理などの機能を備えているが，これらは計算機システムを効率良く動作させるための機構のみであり，マイクロ・メインフレーム・リンクによる統合システムなどの，アプリケーションの構築を指向した管理機能やサービス機能を備えていない．また，OS にその機能を求めるべきでない．「動作環境」は，特定分野のユーザ・システムに共通に必要とされる管理，サービス機能と，それらの機能とのプログラム・インタフェースを提供する機構である．図 1 の斜線で示す部分が「動作環境」である．

3.　DS 7 の「統合操作環境」の実現手法

前述した考え方に基づき，その基本機能を DS 7 に実現した．以下には現在リリースし，動作している機構と機能について，その概要を述べる．

3.1　統合操作環境

並行処理システムは，計算機を効率よく利用するばかりでなく，操作環境を改善する．「統合操作環境」の実現には，その基礎となるオペレーティング・システムの機能がその実現の難易を左右する．具体的には，システム機能としてプロセス・リンク（プロセスの

生成／制御）機能，通信機能，マルチタスク機能，メモリ管理機能などがそれに当たる．

DS7 は，マイクロプロセッサ i8086 系で唯一のマルチタスク OS である，Digital Research 社のコンカレント CP/M 86（バージョン 3.1）を採用しているため，「統合操作環境」の実現は容易であった．

3.2 通信機能の実現手法

垂直統合システムの構築には，ホスト・システムとのパスである通信システムの実現が一つの基礎となる．ターミナルのエミュレーションは，マイクロ・メインフレーム・リンクの最も初歩的な形態である．

一般に，ターミナル機能は，ホスト・システムとテキストの送受信を制御するリンク制御とデータ流量の制御を行うデータ・フロー制御（以下，両機能をまとめてリンク・レベル制御という），およびターミナルの画面を制御するスクリーン制御からなる．

DS7 では，ターミナル機能をスクリーン制御とリンク・レベル制御に機能分割し，二つのプロセスで階層的に構成した．さらに，この二つのプロセス間に通信回線を一括管理する CSX (Communication System Extension) を設け，スクリーン制御プロセスに対し，論理入出力命令で CSX とインタフェースできるようにしている．そのため，COBOL や BASIC などの高級言語によるアプリケーション・プログラム (AP) からも，論理命令で

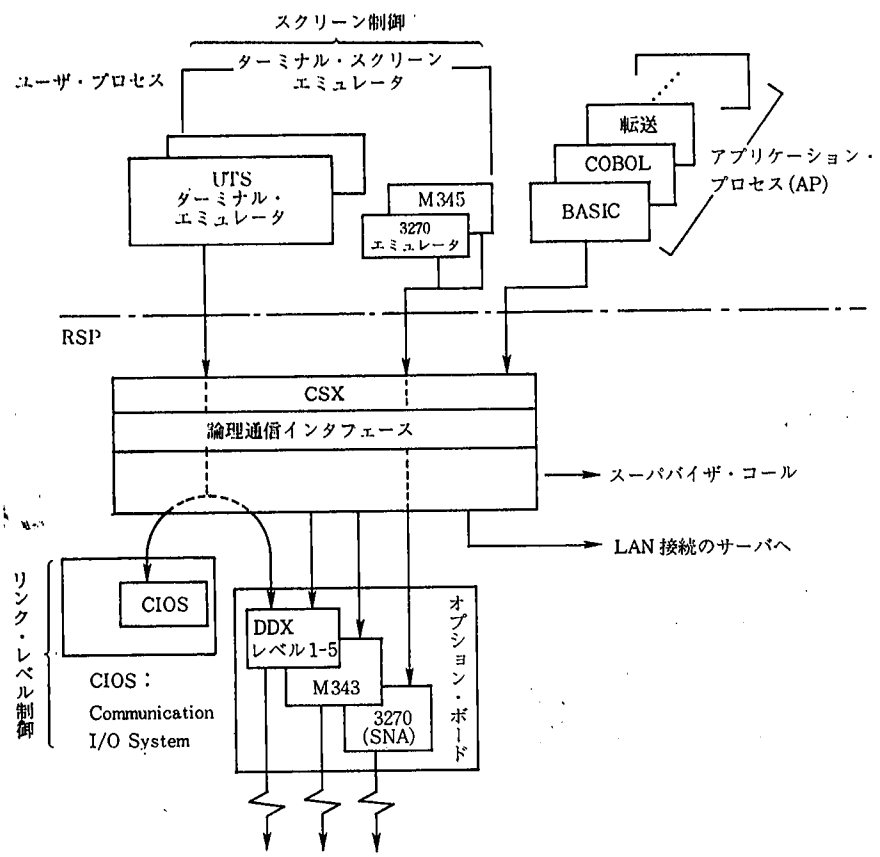


図 5 通信システムの構成

Fig. 5 Block diagram of communication system

CSX と直接にインタフェースすることが可能であり，その結果，AP は同期制御やリンク・レベルの制御から開放され，ホスト・システムとの通信がコンソールへの入出力と同様に容易にできる．また，CSX を介在させたことにより，LAN 接続による通信サーバ・システムの構成や，複数ホストへの同時接続，異種通信プロトコルの並行動作が容易になった．

図5に，ワークステーション DS7 の通信システムの構成を示す．リンク・レベル制御を高優先度の RSP (Resident System Process)，スクリーン制御をユーザ・プロセスで構成した．RSP の階層を占めるリンク・レベル制御のうち，DDX パケット（レベル 1-5）や IBM 3270 などの他社プロトコルは，負荷分散を図るためコプロセッサに（オプション・ボード）により制御する構成とした．

また，リンクとスクリーン機能を分割したことにより，下位のリンク・レベルがネットワーク制御を備える DDX パケット網プロトコルに代わっても，スクリーン制御プログラムや CSX とインタフェースをするアプリケーション・プログラムは，同一のものが使用できる．また，LAN による通信サーバを介したシステム構成においても同様である．

通信回線に対する入出力命令の追加に当たっては，統合操作環境を構築しやすくするためコンカレント CP/M 86 の標準のスーパバイザ・コール命令と同一コールに割り当てて，ファンクション・コードを追加した．

3.3 処理プログラムの単機能化と役割別階層化による処理システムの構築

図6にイメージ処理システム (U-IMAGE) の構成を例として示す．これは，合計七つのプログラムからなる．①は，パソコンと同様の単独な使用形態を与えるオペレータ・インタフェースであり，処理ユニット②から⑤と動的にリンクされ，イメージを主体にした処理システムを構成する．②から⑦の処理ユニットは，イメージの編集，データ圧縮後の

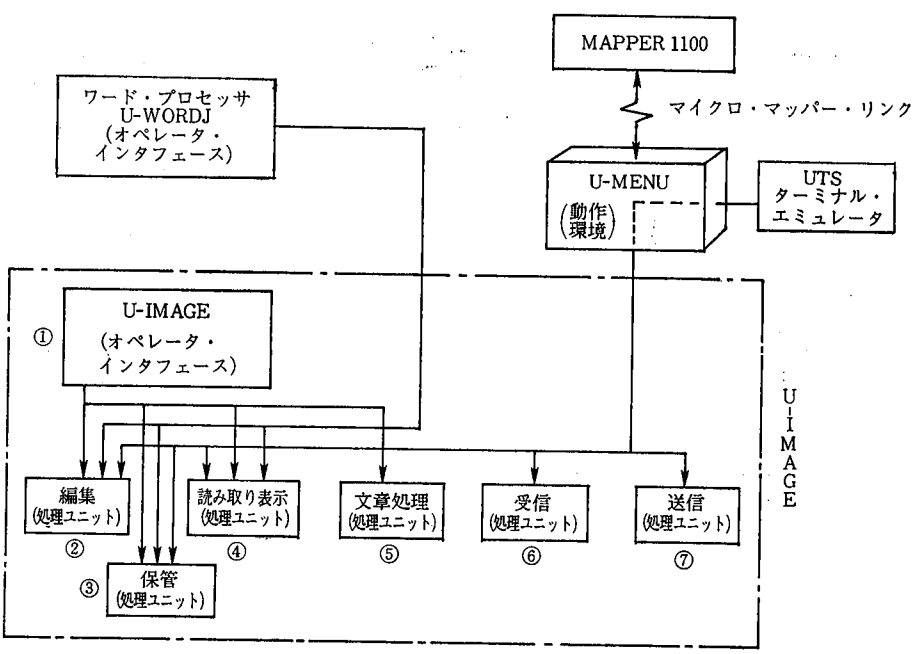


図 6 単機能化と階層化による処理システムの構成例 (U-IMAGE)

Fig. 6 Construction of a processing system (U-IMAGE)

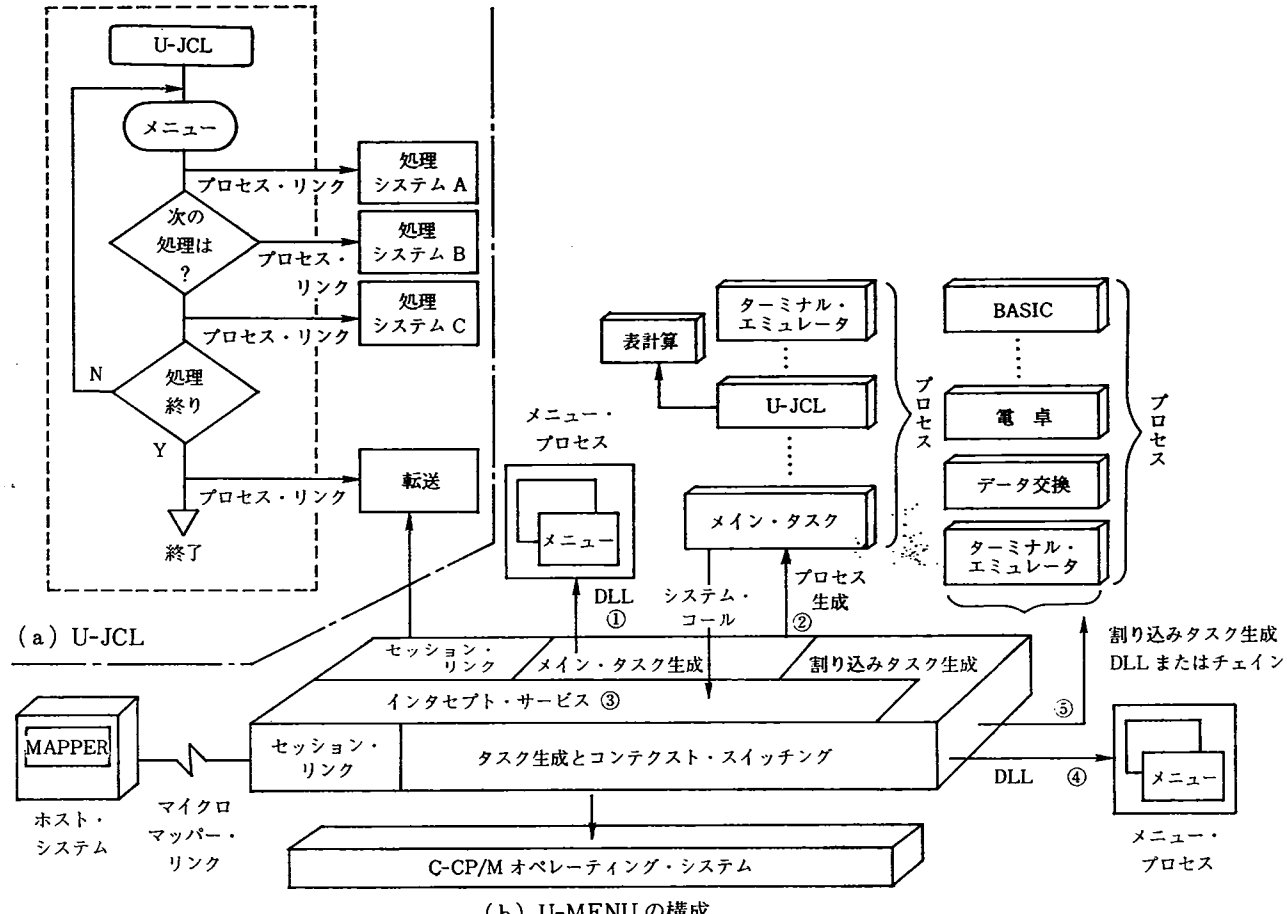


図 7 制御レベル・プログラム U-JCL と動作環境 U-MEMU

Fig. 7 Command processor U-JCL and operational environment core U-MENU

ファイルへの保管，スキャナからの読み取りと表示，文書処理，ホストへの転送など，イメージ処理に関する単機能を果たす．また，これらの処理ユニットは，U-IMAGE が動的リンクによって構成されているのと同様にワード・プロセッサ，またホスト・システムからの命令により起動され，それぞれのシステムの一部として連携動作する．

単機能化と階層化は，操作の統一を図れる他に次の利点がある．まず，小規模のプログラム単位に行えるため，機能の追加・変更や分業による開発・テストが容易となる．また，一般に処理システムの複合度が増すのに比例し動作時の占有メモリが増加するが，単機能化と動的リンクによって，メモリ容量に応じたシステム構成が可能となる．大容量のメモリを備えるシステムに対しては，複数の処理ユニットを並列的にメモリ上にロードし早い応答のシステムにする．逆に小容量のメモリであれば，その途度，必要ユニットをロードする．

単機能化で問題になることは，処理機能を分割する基準である．i80186 はセグメントのサイズが 64 kB が最大である．そのため，プログラムがセグメント間にまたがるプログラム（ビッグ・モデル）と 64 kB 以内に収まるプログラム（スモール・モデル）に分け，プログラム作成しなければならない．C言語では，どちらのモデルも作成可能であるが，ビッグ・モデルは同一プログラムでも，そのコードの大きさは 1.5〜から2倍となる．また，それに反比例して処理速度は低下（15〜25%）する．そのため，DS7 では，処理ユニットの最大のサイズをスモール・モデルで組めるサイズにしている．

3.4 制御レベル・プログラム（U-JCL）

制御レベル・プログラム U-JCL は，処理ユニットなどの連続実行や条件付き実行をコマンド制御文で行うコマンド処理プログラムである．そして，オペレータ・レベルや実行レベルのプロセスを動的にリンクし，複合的な統合システムを実現する．

図7（a）は，条件により処理システムA，B，Cを選択し，処理をした結果を自動的にホスト・システムに転送する例である．点線内が U-JCL の機能である．

4. DS7 の動作環境（U-MENU）の構造と機能

垂直統合システムを実現するための「動作環境」を図7（b）に示すように構成した．U-MENU は，統合システムを構築する上での中核であり，OS から見ると一つのプロセスとして動作するが，各 AP から見れば OS と考えてよい．主な機能を関連プロセスと合わせ，その動作概要を説明する．

4.1 メイン・プロセスの選択と生成

U-MENU のメイン・タスク生成ルーチンはメニュー・プロセスを DLL し（図7の①部分），メニュー・プロセスによって選択された処理システムの子プロセスを生成する（②）．これを U-MENU ではメイン・タスクと呼ぶ．そして，U-MENU は親プロセスとして動作する．メイン・タスクが出力したシステム・コールは，U-MENU のインターセプト・サービス・ルーチンを経由し，OS のスーパバイザをコールする．

4.2 割り込みプロセスのネスティング

インターセプト・サービス・ルーチンは，メイン・タスクの実行を一時中断し，他のプロセスを起動したり，またはメイン・タスクに動的に他の処理ユニットをリンクし，メイン・タスクの一部のサブルーチン，あるいは子プロセスとして生成する機構である．メインのプログラムを変更することなく，処理機能の拡張ができる．

割り込みプロセスは，（1）キー・ボード上のポップアップ・キーの押下時，そして（2）

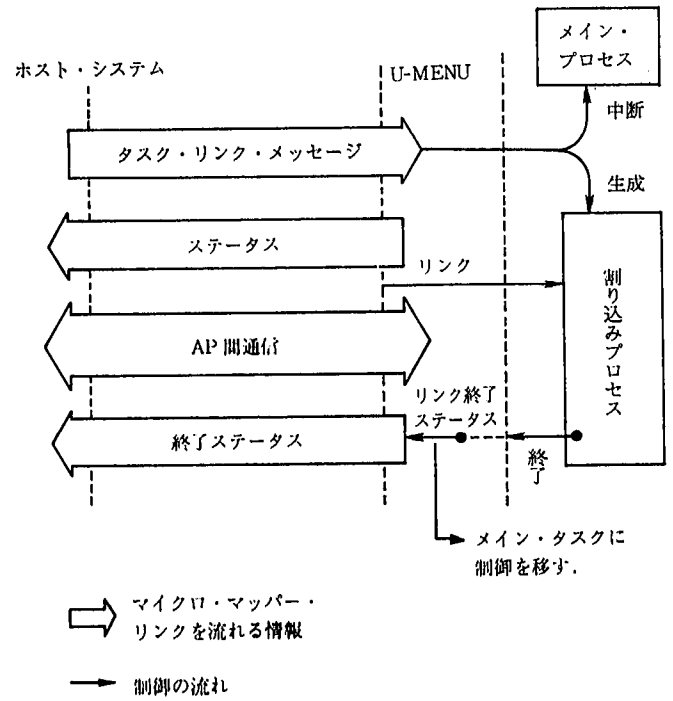


図 8 マイクロ・マッパー・リンク

Fig. 8 Micro-MAPPER-link

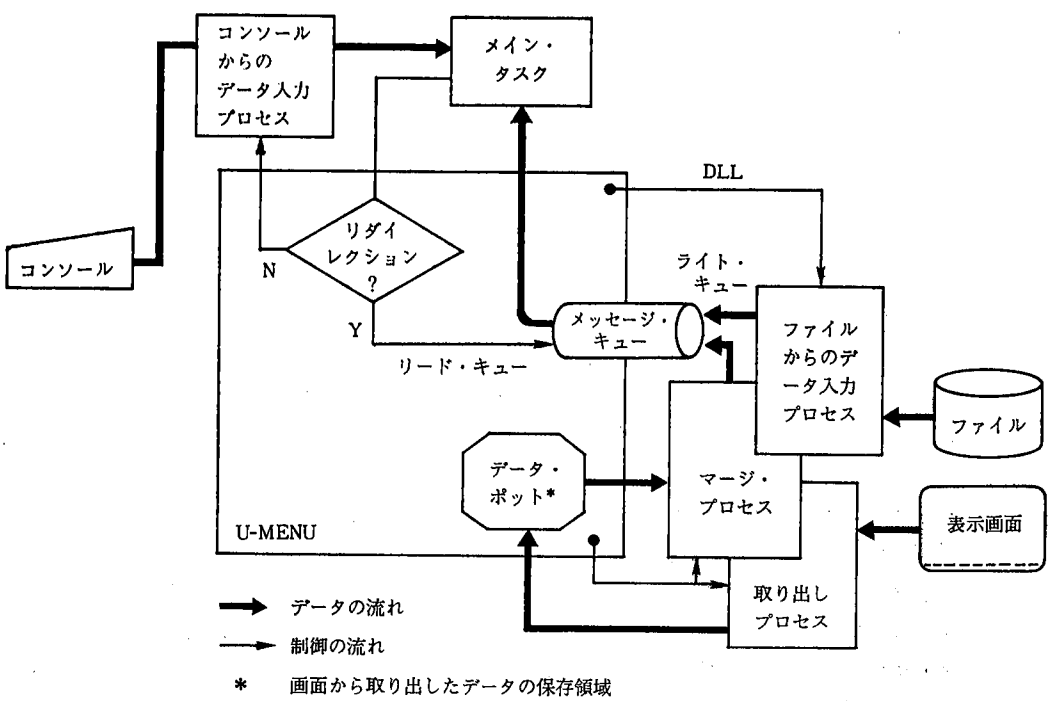


図 9 リダイレクト I/O によるデータの入力

Fig. 9 Data input by I/O redirection

ホスト・システムからのタスク・リンク・メッセージ（図8参照）により起動する．インターセプト・サービス・ルーチンは，通信回線とキー・ボードからの入力をモニターしており(③)，割り込みプロセスの起動要求を，インターセプトする．ポップアップ・キーからの起動要求に対しては，メイン・タスクの生成と同様に，メニューに登録された処理からオペレータが選択する(④，⑤)．ホストからの起動要求は，起動する処理番号，あるいはコマンド名をタスク・リンク・メッセージで指定する．割り込みタスクが終了した時には，メイン・プロセスに戻り，中断された時点から再開する．

4.3 セッション・リンク・サービス

セッション・リンク・サービスは，一連の処理システムを連続して実行する際に，同一の通信回線の割り当てと，その通信回線の状態を変えないまま先行する処理システムから後続の処理システムへと引き継ぐ機能である．また，割り込みプロセスが回線を使用する場合には，メイン・タスクが使用している回線を，一時的にリンクされたプロセスに引き渡す．

セッション・リンク・サービス・ルーチンは，メイン・タスクからの回線のオープン・システム・コールをトラップすることによりセッション・リンクを実現している．

4.4 マイクロ・マッパー・リンク

ホスト (MAPPER) とワークステーション上のアプリケーションを結合するプロトコルを「マイクロ・マッパー・リンク」と呼んでいる（図8）．MAPPER は，このプロトコルを通して，先に述べたイメージ処理の処理ユニットなど，実行，あるいはオペレータ・インタフェース，制御レベルなどの任意のプログラムを呼出し，MAPPER の機能の一部であるようにマイクロ・システムの処理プロセスと連動させる．

4.5 データ交換サービス

DS7 では割り込みプロセスを利用し，処理システム間のデータ交換機能の一部を実現した．

画面への表示とコンソールからの入力は，各プログラムが共通に備えている機能であることに着目し，画面からのデータの切り出しと，入力源をコンソールからファイルへリダイレション（切替）する方法を用いて行う．図9にその機構を示す．簡単な機構であるがメイン・タスクとして動作している，たとえば，表計算プログラム（マイクロ・レポ）等にホスト・システムから受信した画面データ，あるいはファイルを入力として取り込むことができる．

5. 処理システムの構築と動作例

最後に，以上述べた「統合操作環境」上でいかに処理システムを構築していくのか，またその操作や動作の概要について簡単に説明する．

5.1 メニューの作成と定義例

図10(a)はメイン・タスク・メニューの表示画面の例である．また図10(b)は割り込みタスク・メニューの表示画面の例である．

ユーザは処理システムの構築に当たり，メイン・タスクと割り込みタスクの二階層に分けて定義する．各タスクは，その処理手順とそれに対応する表示メニューをそれぞれのメニュー・ファイルに登録する（図10(c)）．

メニューは，処理内容を表す日本語文章により記述し，それに対応する処理手順は，U-JCL によって組み立てたり，または実行レベルの処理ユニットやオペレータ・インタ

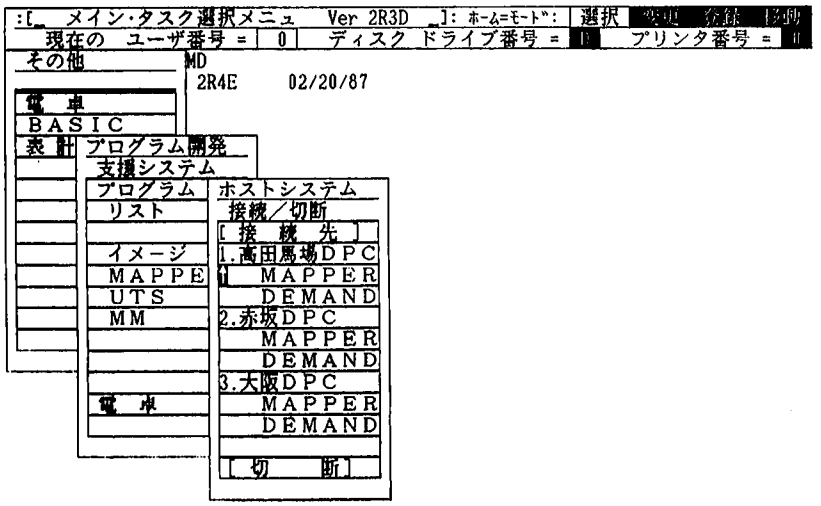


図 10(a) メイン・タスク・メニュー

Fig. 10(a) Main task menu

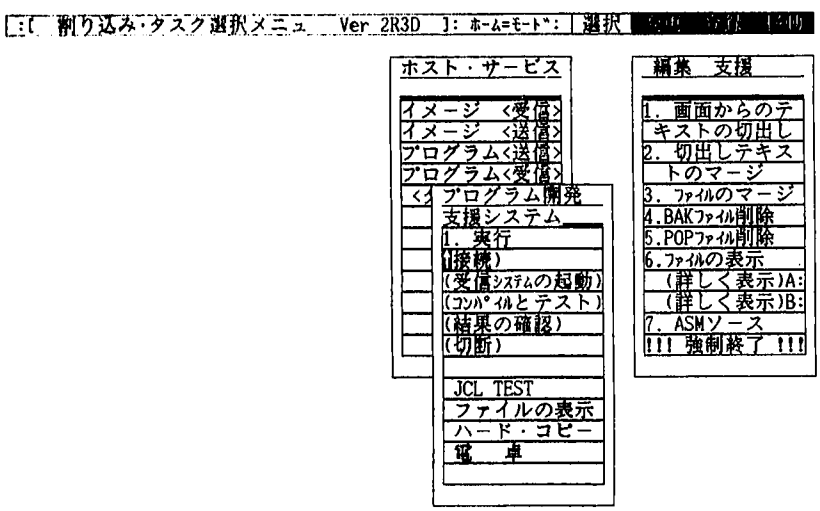


図 10(b) 割り込みタスク・メニュー

Fig. 10(b) Intercept task menu

フェース・レベルのプログラム名を指定する．定義されたメニューは，オーバー・ラッピング方式のウインドウ形式で表示される．最大 8 ウインドウ，最高 96 処理システムの定義が可能である．また，ウインドウの表示位置，個数，メニュー内容は，メニュー表示時，いつでも変更可能である．図 10(c) は，日本ユニバックの Uai (Univac advanced internetwork) ネットワーク経由で接続された複数のシリーズ 1100 上の MAPPER，またはデマンド・システムへの自動接続手順を U-JCL で組み立て，メニュー登録した例である．

このメニューに，たとえばワープロ機能を登録する場合には，処理メニュー定義領域（＊＊表示名＊＊）に“文書作成”などと表記し，そしてこれに対応する処理システムの定義部（＊＊コマンド＊＊）にコマンド名 (WP) を入力することにより作成できる．

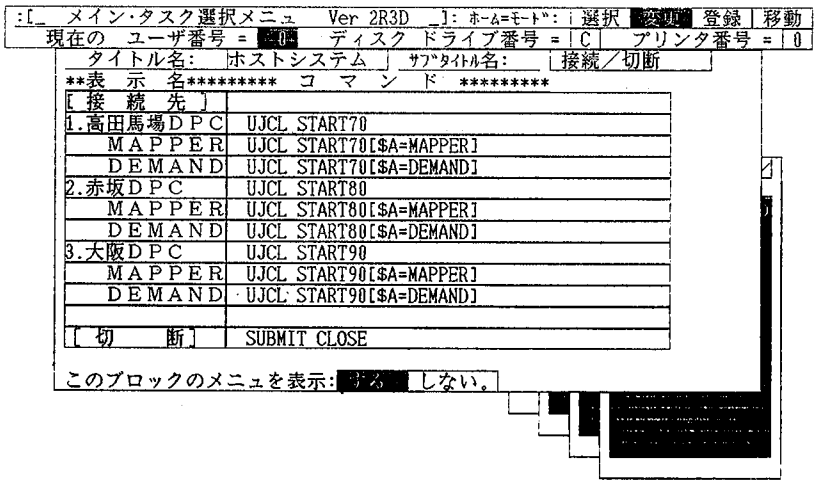


図 10(c) メニュー定義

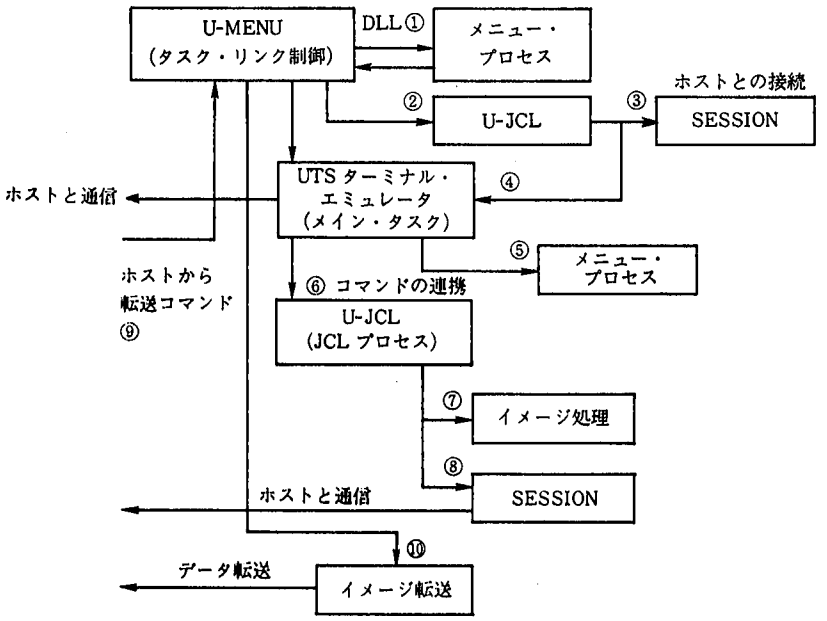
Fig. 10(c) Definition of menu

5.2 イメージ転送システムの動作例

ホスト・システムとワークステーション DS7 とのマイクロ・マッパー・リンクによる結合，並びにその時の各プロセスの動作をより具体的に示すため，イメージ・スキャナで図面を読み取り編集加工した後，イメージ情報としてホスト・システムに転送する簡単な例をあげ説明する．図 11 はこの時の各プロセスの流れを示している．キーボード上のポップアップ・キーを押すと，U-MENU が起動され，U-MENU はメニュー表示ルーチンを動的リンクし（図 11 の①），図 10(a) に示した形式でメイン・タスク・メニューを表示する．

ここで赤坂 DPC（日本ユニバックのデータセンタ）の MAPPER システムと接続するため，ホスト・システム接続／切断メニューの中から赤坂 DPC の MAPPER を選択する．U-MENU は，メニュー表示ルーチンが使用していた領域を開放した後，メニューに定義された（図 10(c)），コマンド処理プログラム U-JCL をメイン・プロセスとして生成する（図 11 の②）．U-JCL では，START 80 というファイル名でコマンドの実行順番が条件文とともに設定されている．U-JCL は，コマンド実行手順に従い，子プロセスとして SESSION プログラムを生成し（③），SESSION プロセスによりホスト処理システム MAPPER と接続する．SESSION プロセスが終了すると，U-JCL は次に UTS ターミナル・エミュレータをプロセスとして起動する（④）．この時点で，オペレータはホストの MAPPER システムを自由に操作できる状態となる．プロセスとしては，U-MENU，U-JCL，ターミナル・エミュレータ (UTS) がネスティング状態にある．

ここで，イメージ・スキャナから画面を読み込み編集するため，ポップアップ・キーを押し（⑤），割り込みタスク・メニューを表示する．この中のホスト・サービス・メニューの中から（図 10(b)），イメージ処理の送信を選択する．この処理は，同様に U-JCL で処理手順が組み立てられている．割り込みプロセスとして U-JCL が起動され（⑥），U-JCL は，イメージ・スキャナから図面を読み取り表示する処理ユニットをプロセスとして生成し，読み取り完了後イメージ編集ユニットをプロセスとして生成する（⑦）．編集が終了すると，U-JCL は SESSION プログラムをプロセスとして生成し（⑧），SESSION プログラムにより MAPPER に対してイメージ転送要求が出される．ここで，割り込みプ



① メイン・タスク・メニューの表示と選択
②③④ ホストと接続し，メイン・タスクを起動(メイン・タスク)
⑤ ポップ・アップ・キーを押し，割り込みタスク・メニュー表示と選択(割り込みタスク)
⑥⑦⑧ イメージ処理とホストへのイメージ転送要求(割り込みタスク)
⑨ ホストよりイメージ転送プロセスの起動要求(割り込み要求)
⑩ イメージ転送プロセスによるイメージのホストへの転送(割り込みタスク)
⑪ ホスト・システムの初期画面への復帰(メイン・タスク)

図 11 イメージ転送に使用するプロセスとその流れ

Fig. 11 Flow and processes to transfer image data

ロセスは終了し，制御はターミナル・エミュレータに返る．

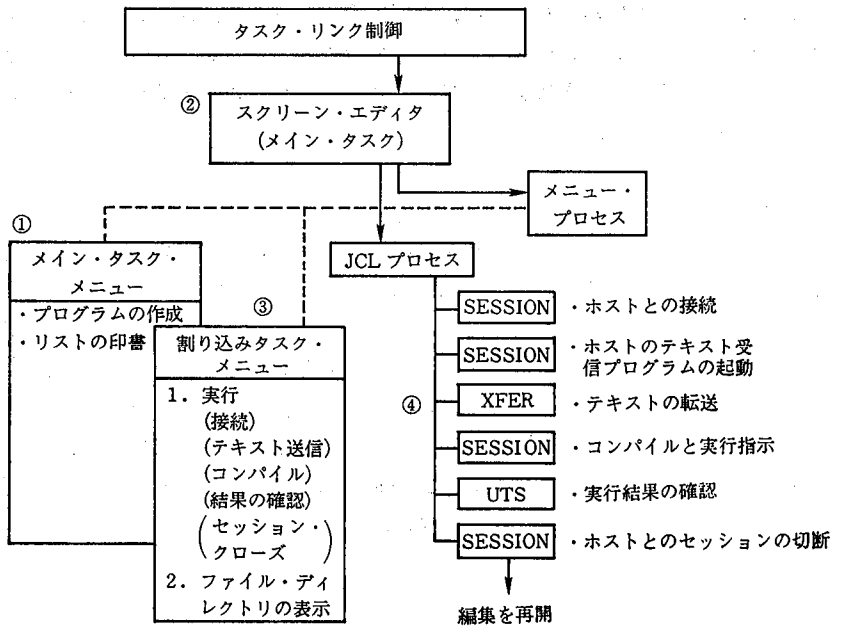
MAPPER システムは，先のイメージ転送要求に対しマイクロ・マッパー・リンク・プロトコルに従って，イメージ転送プロセスの起動を U-MENU に対し出力する（⑨，このとき，オペレータの介入が必要であれば，MAPPER システムに合わせた介入要求を，メニュー選択形式で出力する）．U-MENU は，この要求に従ってターミナル・エミュレータを一時中断し，イメージ転送プログラムをプロセスとして起動する（⑩）．ここで，イメージ転送プロセスによりデータ転送が開始される．

転送終了後は，割り込みプロセスが発生する前の状態に復帰し，ターミナル・エミュレータを使用した通常の操作が行える（⑪）．

5.3 プログラム編集支援システムの動作

図 11 に示した例は，ホスト・システムとの結合のベースであるターミナル・エミュレータをメイン・タスクとして動作させた場合であった．ここでは，市販のパソコン・ソフトウェアであるスクリーン・エディタをメイン・タスクとし，逆にターミナル・エミュレータなどの通信プロセスを割り込みプロセスとして動作させる例をあげる（図 12）．

パソコン用市販スクリーン・エディタは，ホスト・システムの提供するライン・エディタに比較すると，その応答性・操作性が優れているので，ワークステーション側でソース・プログラムを作成し，コンパイルと実行をホスト・システムで行うという作業を組み立てそのプロセスの流れを簡単に追ってみる．



① ポップ・アップ・キーを押し，メイン・タスク・メニューからプログラムの作成を選択
② スクリーン・エディタがロードされ，カット/ペイスト，マージ機能を使用し，プログラム・コードの編集を行う．
③ 終了後ポップ・アップ・キーを押し，割り込みタスク・メニューから任意の処理レベルを選択する．1を選択すると，テキストの送信からセッションのクローズまで実行．
( )の中を選択すると，その処理のみを実行．
④ 終了すると，先の編集の続きから再開

——→ 制御の流れ

----- メニュー・プロセスで使用されるメニュー

図 12　プログラム編集支援システム

Fig. 12　Program editing system

メイン・タスク・メニューから，テキスト編集を選択するまでの操作は，前例と同じである．テキストの編集は，割り込みタスク・メニューの一つである画面からのテキストの切り出し，そのテキストのメイン・タスクへのマージ機能などを活用することにより高能率の作業が行える．

ここで編集作業が終わったことにし，図10(b)を再び見ていただき，この割り込みタスク・メニューの中央のウインドウには，ホスト・システム上で行う作業手順がメニュー(プログラム開発支援システム)で示されている．1の実行を選択すると，回線の接続から編集したテキストの送信，コンパイル，そして回線の切断まで一連の処理を実行する(カッコで囲まれたメニューを選択した場合には，カッコ内の処理のみ行う)．

図12は，このプログラム編集支援システムのプロセスの流れを示している．④はメニューから1の実行を選んだときのプロセスの流れである．ここでは，前例でメイン・タスクとして取り上げられたターミナル・エミュレータが割り込みタスクとして取り上げられている．合計六つのプロセスが，チェインしホストとのパスを使用するが，セッション・パスは，最初のプロセス SESSION で使用した回線が，U-MENU のセッション・チェイン機能により，次のプロセスに引き継がれる．

以上，二つの例に示したように，今回開発した「動作環境」によりメニュー，あるいはオペレータ・インタフェース・レベルのプログラムをホスト・システムまたはワークステーション上に備えることにより，処理ユニットを自由に組み合わせた形の垂直統合処理システムを構築できる．

6. おわりに

この構築法の特色は，マルチタスク機能を最大限に生かした点と個々の機能の単純化にある．現在の U-MENU で実現されている環境は，エンド・ユーザが COBOL などにより，垂直統合を主体とした統合システムを構築しようとする場合，U-MENU とのインタフェースが不足のため，機能を十分に利用できなかったり，内部統合を図るための基本機能であるデータ交換機能が欠けている．また操作性においても応答性や作業の習慣への配慮が欠けている．これらは，改善していかなければならない点である．

最後に，本稿は，先に日経コンピュータ昭和61年5月26日号に寄稿した原稿に，加筆・修正したものであることを付記する．

参考文献 ［1］ Digital Research 社 CCP/M 86 Programmer's Reference Guide

執筆者紹介 佐々木 茂 (Shigeru Sasaki)
昭和23年生．同44年新居浜高専電気工学科卒業．同年住友重機械(株)入社，同45年日本ユニバック(株)入社，大型計算機の保守システムの開発に参加．通信／ネットワーク・システムの開発保守に従事した後，現在商品開発本部のワークステーション・システム部開発室開発支援課長として，マイクロ・プロダクトの開発を担当．

論文

画面作成データ・ストリーム変換プログラムの評価

Evaluation of Conversion Program for Screen Datastream

庭 山 宣 幸

要 約 最近，従来の端末と仕様の異なる端末を，既存のシステムに接続する要請が増加してきている．

この問題の解決に当たっては，システムや端末側の修正を実施せずに異仕様端末を接続可能にする点が肝要であり，これを専用にサポートするソフトウェアが必要となる．

このソフトウェアの基本機能としては，次の2点があげられる．

1) 出力時変換：すなわちシステムから出力される従来の端末用の画面の生成データを異仕様端末用に変換．

2) 入力時変換：すなわち異仕様端末から入力されるデータを従来の端末からのそれと等価な形式へと変換．

本稿では，このような端末画面データ・ストリームの変換プログラムの持つべき機能と変換方式の関係について一般的な考察を試みた．さらに，実際の例として MAPPER 1100 に IBM 3270 準拠端末の接続を可能としている MPC 1100 をとりあげ，実際に評価した．

Abstract Nowadays the case has increased that a foreign terminal is attached to an existing application system.

Generally speaking, a conversion program is developed individually, which makes such a requirement possible without modification of the existing application program and terminal program.

It stands in the middle of the application program and foreign terminal program, and handles data conversion.

In the former part of this report the generalized functions and methods of such a conversion program are discussed. In the latter part MPC 1100 as an example, is evaluated which makes IBM 3270 terminal attach to MAPPER 1100.

1. はじめに

ハードウェアの低価格化，ネットワーク技術の進歩およびニーズの多様化により異機種コンピュータ間の結合や異機種端末接続の必要性は，ますます高まりつつある．

一方，現在のアプリケーション・システムの多くは，オンライン端末からの使用が前提となっており，端末の画面はシステムとエンド・ユーザの接点となっている．このため，エンド・ユーザから見てわかりやすく，使いやすい画面を提供できるかどうかが，システム全体の評価を左右する場合もまれではない．

特定の端末を前提としたアプリケーション・システムに対し，使用範囲の拡大を図る目的で新しい種類の端末を接続する必要が生じる場合が多い．この場合，従来のソフトウェア資産を保護するために既存のシステムには手を加えず，その外側に新端末に対応した変換プログラムを接続させインタフェースをとる場合が多い．

このように異仕様端末間での画面作成データの変換プログラムの開発を考えるとき，実現すべき機能とその実現方式とが問題になる．

本稿では，初めに画面作成データの変換を目的としたプログラムの機能と実現方式について一般的に論じる．後半では MPC 1100 (MAPPER 1100 に IBM 3270 準拠端末（以

下，3270端末と呼ぶ）の接続を可能とするソフトウェア）の評価を行う．

なお，本文中において以下の語句を次の意味で使用している．

- 画面データ・ストリーム：ホスト・コンピュータよりオンライン端末に出力される表示画面を作成する電文の内容．SDS (Screen Data Stream) と略している．
- プロトコル：SDS を作成する上での規約．
- 画面アドレス：端末画面上の任意の文字位置を表現する情報．
- 表示属性：端末画面表示時に指定される表示上の特質．表示／非表示，輝度，入力保護／非保護(端末キー・ボードからの入力に対して)，文字色，背景色などを指す．
- フィールド：端末画面上，共通の属性を持つ画面アドレスの連続した領域，端末により上記の表示属性以外の文字や罫線の属性も含む．
- フィールド情報：フィールドの性質を表す情報．
- フィールド設定位置：フィールドが新たに設定される画面アドレス．
- FCC：当社の UTS シリーズ端末でのフィールド情報を制御する制御コード．Field Control Character の略．

2. 一般的な SDS 変換プログラムの考察

2.1 SDS の特性

SDS 変換プログラムの開発時に考慮すべき SDS の特性として，以下のものがあげられる．

1) SDS は，実際に端末画面に表示される文字コードの他に，次の機能を実現する制御コードより成り立っている．
   - 画面アドレス
   - フィールドの設定
   - 画面編集
   - 周辺機器制御

アプリケーション・システムから端末へ送信される SDS では，

2) 一つの画面を作成する SDS は，必ずしも画面アドレスの小さい方から大きい方へシーケンシャルに作成されているとは限らない．

3) 一つの画面を作成する SDS は，複数のテキスト・ブロックより成立している場合もある．

4) 一つの画面を作成する SDS には，直前の表示画面を前提としてそれの修正を実施するだけのものもある．

また，端末よりアプリケーション・システムへ送信される SDS では，

5) 端末画面へユーザが入力した文字データ以外にも，入力位置を示す画面アドレスや入力フィールドのフィールド情報もアプリケーション・システムへ送信しなければならない場合がある．

2.2 機　　能

SDS の変換を行うプログラムの機能として前項 2.1 の 1) で述べた SDS の特性により，以下の 2 種類が必要となる．

1) 文字コード変換機能……一般に 1 バイトで意味を持つ英字，数字，特殊文字，カナ文字と 2 バイトで意味を持つ漢字の別コード体系への変換を行う．

2) 制御コード変換機能……制御コードの機能の内容により，一つまたは複数の別制御

コードへの変換を行う.

変換対象となるプロトコル間の文字コード，および制御コードの対応付けが完全に行える場合は，上記二つの変換機能により SDS 変換は可能となる．しかし，直接的な変換が困難な場合，一方のプロトコルにおいて表現しようとしている情報の意味を解釈し，その意味に適した他方のプロトコルでの表現に置き換えてやる必要が生じる．これを“意味変換機能”と呼ぶ.

以上，三つの機能の変換処理全体における割合は，対象となる二つのプロトコル間の差異の大きさにより異なってくることが予想される．すなわち，差異の少ないプロトコル間の変換処理では上記 1) と 2) の変換が主となり意味変換の割合が減少する．一方，差異の大きいプロトコル間の変換においては，逆に意味変換の占める割合が大きくなる.

2.3 方　　式

SDS の変換方式の基本的なものとして，次の 2 種類がある.

1) 逐次変換方式……この方式は，変換プログラム本体が入力された SDS を 1 バイトずつ入力バッファの先頭から終端まで，一度の走査で処理しようとするものである（図 1）.

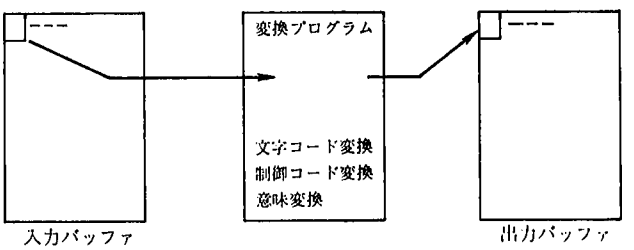


図 1　逐次変換方式

Fig. 1　Sequential processing method

2) 蓄積変換方式……この方式では，入力した SDS をいったん変換プログラム内の中間バッファに特定の形式に展開した形で蓄積したのちに，目的とするプロトコルへと変換するものである（図 2）.

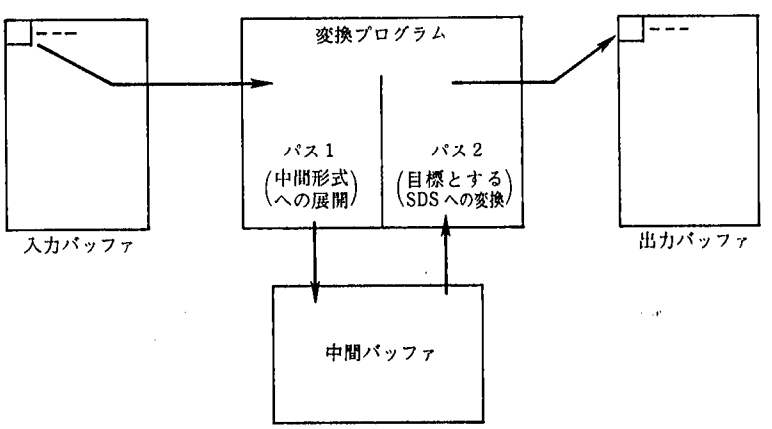


図 2　蓄積変換方式

Fig. 2　Buffering processing method

以上，二つの変換方式について相対的な比較を行うと次のようになる.

1) パフォーマンス……逐次変換方式は一度の走査で処理を終了できるのに対し，蓄積変換方式は入力した SDS の量の多少にかかわらず中間形式に展開し，中間バッファに蓄積しなければならない．このため，蓄積変換方式の方が逐次変換方式と比較するとパフォーマンスは良くない．

2) SDS の変換可能率……逐次変換方式は，前述の文字コード変換および制御文字変換については対応しやすいが，意味変換を行う場合に走査対象の SDS が表示画面を直接表現していないため，意味解釈が難解になる．これに対し，蓄積変換方式は入力した SDS を意味解釈しやすい形式に展開しているため，意味解釈が容易となり変換可能率は高い．

3) 開発工数……逐次変換方式は変換処理が 1 パスとなり構造が単純になる．これに対し蓄積変換方式は，2 パス構造となる．このため，蓄積変換方式のほうが，逐次変換方式に比較して開発工数は多い．

4) 汎用性……逐次変換方式は前述 2) のとおり，蓄積変換方式と比較して SDS の変換可能率は低い．このため，変換プログラムとしての汎用性も蓄積変換方式と比較して小さい．

5) 拡張性……蓄積変換方式は前述 3) のとおり，中間バッファを中心とした 2 パス構造となっている．このため，機能拡張が入力 SDS を中間バッファに展開する部分に関連するものならばパス 1 を，また中間バッファから新しい SDS を作成する処理に関連するものならばパス 2 を修正すればよく拡張しやすい．これに対し逐次変換方式は，構造が単純なために機能拡張に対する修正が，かえって適用しにくい．

以上を，まとめたものが表 1 である．

表 1　変換方式の比較

Table 1　Comparison of both methods

項　　目	変換方式間の比較
1) パフォーマンス	逐次変換方式 ＞ 蓄積変換方式
2) SDS の変換可能率	逐次変換方式 ＜ 蓄積変換方式
3) 開 発 工 数	逐次変換方式 ＜ 蓄積変換方式
4) 汎　用　性	逐次変換方式 ＜ 蓄積変換方式
5) 拡　張　性	逐次変換方式 ＜ 蓄積変換方式

3. 実際のプロダクトにおける評価

3.1 MPC 1100

MPC 1100 (MAPPER PROTOCOL CONVERTOR FOR 1100) は，当社の UTS シリーズと DS シリーズの端末（以下，UTS 端末と呼ぶ）を標準とする MAPPER 1100 に 3270 端末の接続を可能とする SDS 変換プログラムである．

MPC 1100 は，MAPPER 1100 より送られた UTS 端末用の SDS を 3270 端末用に変換し，端末画面に表示する．また，3270 端末に表示された画面より入力され送られてきた SDS を UTS 端末用に変換する．MPC 1100 が変換対象としている二つの SDS は，一般に UTS プロトコル，3270 プロトコルと呼ばれるプロトコルにより規定されている．この二つのプロトコル間の仕様の違いは，そのまま UTS 端末と 3270 端末の端末ハードウェア間の仕様の違いでもあり，表 2 のようにまとめられる．

表 2　UTS と 3270 の端末仕様上の相違点

Table 2　Comparison between UTS and IBM 3270 terminal hardware

番号	項　目	3270端末	UTS 端末
①	フィールド属性の定義	画面上のすべての表示領域は，フィールドとして定義される．文字種別や罫線も画面上のフィールド単位に規定されるフィールド属性である．	表示画面の表示属性に関係する部分が，フィールド単位に設定される．
②	文字属性の定義	文字属性の考え方がない．	漢字と罫線は，表示文字の1文字ごとに規定される文字属性である．
③	フィールド属性の設定	副指令 SF（または SFE 注1）で設定され，表示画面上1バイトを占める．	FCC 文字（US, EM シーケンス）で設定される．表示画面を使用せずに設定できる．
④	文字属性の設定		漢字の場合は，漢字シフトコードを使用する．罫線の場合は，罫線文字シーケンスを使用し設定する．
⑤	表示可能文字数 注2)	フィールド数＋データ数≦1920	フィールド数≦1920 データ数≦1920
⑥	画面アドレス 注2)	画面上の桁位置は最初の桁を0，最終桁を1919とした1次元アドレスで表現される．	画面上の横方向を“桁”，縦方向を“行”と呼ぶ2次元アドレスで表現される．
⑦	画面編集機能 注3)	ホスト制御＞端末制御	ホスト制御＝端末制御
⑧	送信形式	基本的に送信形式は一つのみ． 画面上のカーソル位置に関係なく，変更された部分のみが送信される．	送信形式には3種類（VAR, CHAN, ALL）がある． 基本的には，入力開始記号（SOE）よりカーソルまでのデータが送信される．
⑨	スクロール 注4)	スクロール機能がない．	画面全体，または一部分のスクロールが可能．
⑩	装置アドレス	ステーション・アドレスとユニット・アドレスの2階層アドレスで表現され，印書装置は表示装置とアドレス階層上対等な位置にある．	リモート・アイディ，ステーション・アイディ，デバイス・アイディの3階層アドレスで表現され，印書装置は表示装置の補助装置として位置付けられる．

注1)　SF は ‘Start Field’，また SFE は ‘Start Field Extended’ の略である．
注2)　80×24 の画面サイズのタイプ間での比較である．
注3)　ホスト制御とはホスト・コンピュータ側のプログラムからの制御を指す．
注4)　これ以外にも，一般に 3270 端末の機能は UTS 端末に比較して画面編集のための機能が少ない．

UTS 端末を標準としているソフトウェアに対して，3270 端末を接続する場合，表2の内容について十分検討をする必要がある．

3.2　3270 と UTS プロトコル間の SDS 変換プログラムの機能の検討

表2より，3270 端末が“フィールド”を基準とした端末であるのに対し，UTS 端末は“文字”を基準とした端末であることがわかる（表2の①，②参照）．

このことは，SDS 変換の中心が文字属性とフィールド属性との属性変換となることを示している．

以下に，前章で分類した三つの変換機能について検討する．

1)　文字コード変換機能

- ANK 文字（英字，数字，カナ文字，特殊文字）

  EBCDIC コード（3270 端末）と8単位標準符号（UTS 端末）の変換を行う．

- 漢字

  IBM 漢字コード（3270 端末）とユニバック漢字コードの変換を行う．

ただし，3270 端末では，画面表示文字はフィールド属性であり（表2の①参照），ANK 文字フィールドの設定が必要となる．このフィールド設定位置が表示画面上1

バイトを占めるため，UTS 端末の表示とまったく同一の画面を 3270 端末に表示できない（表 2 の③参照）．

2) 制御コード変換機能

- 画面アドレス

  UTS 端末の 2 次元アドレスと 3270 端末の 1 次元アドレス間の変換を行う（表 2 の⑥参照）．

- フィールドの設定

  UTS 端末では，フィールドの設定位置にも文字表示が可能であるが，3270 端末では許されない（表 2 の③参照）．このため，UTS 端末での表示画面を完全に 3270 端末で表示することはできない．

- 画面編集

  3270 端末は，UTS 端末と比較してスクロール機能など画面編集に関連した機能が少ない．このため，UTS 端末での画面表示と同等な表示を 3270 端末で実現することはできない（表 2 の⑨参照）．

- 周辺機器制御

  制御コードの変換，および UTS 端末での 3 階層アドレスと 3270 端末での 2 階層アドレス間の変換を行う（表 2 の⑩参照）．

3) 意味変換機能

上記二つの変換機能では，対応できない部分をまとめると以下のようになる．

3270 端末への画面表示時（UTS⟶3270 変換）では，①フィールド設定位置の文字が表示されず，またフィールド設定位置に入力ができない，②画面編集機能同士が完全に対応しない．一方，3270 端末からの入力時（3270⟶UTS 変換）では，送信形式の違いがあり，3270 端末からのデータでは情報が足りない（表 2 の⑧参照）．

意味変換機能として，これらの問題へ対応する必要がある．

以上の検討より，3270 と UTS プロトコル間の SDS 変換プログラムの機能を実現する変換方式として，逐次変換方式は使用できないことがわかる．

3.3 MPC 1100 の変換方式

MPC 1100 は，前節 3.2 で検討した変換プログラムの機能を実現するため，以下のように対応した．

1) 中間バッファの形式……MPC 1100 では，中間バッファの形式として，図 3 のような端末画面の形式を採用した．

2) 出力（MAPPER 1100→MPC 1100→3270 端末）……3270 端末画面への出力時には，文字属性からフィールド属性への変換結果として生じるフィールド設定位置の取り扱いが問題になる．そこで，中間バッファを使用して次の処理を行うようにした．

① UTS 端末と同じ入力フィールドの桁数を保証するために，入力フィールドの設定を行う際のフィールド設定位置は，UTS 端末の場合の一つ手前に設定した．

② 意味変換の結果，文字フィールドの設定と表示フィールドの設定が同一桁位置となった場合は，属性の論理和を採った．

画面編集コードについては直接的な変換を実施せずに，中間バッファに展開された情報から 3270 端末用の SDS を作成した．

3) 入力（3270 端末→MPC 1100→MAPPER 1100）……MAPPER 1100 への入力処理は，プロトコル間の送信形式の違いを克服するところにある．

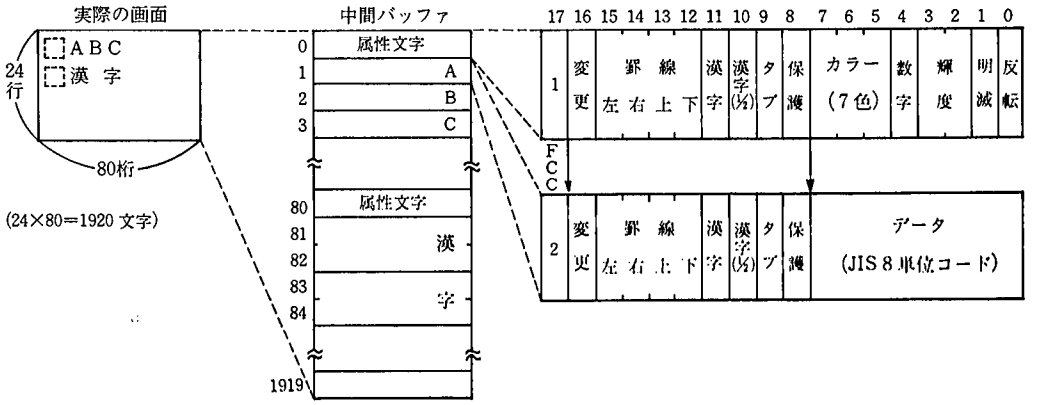


図 3　MPC 1100 の蓄積バッファ

Fig. 3　Format of buffering area of MPC 1100

中間バッファには，常に現在 3270 端末に表示中の画面情報が蓄積されている．実際に端末から送られてくるデータと中間バッファの情報を組み合わせて，UTS 端末が送信するデータと同一のものが作成できた．

また，出力時に実施した意味変換の結果，3270 端末からのデータ中のフィールド設定位置にずれが生じる場合があった．これも中間バッファの情報により矯正できた．

3.4　MPC 1100 の評価

MPC 1100 の採用している SDS 変換方式を紹介したが，ここでは先に評価尺度としてあげた項目につき評価を行う．

1) パフォーマンス……MPC 1100 を経由し接続された 3270 準拠端末と MPC 1100 を経由しない UTS 端末の応答時間を比較すると，次のような結果となった．

UTS 端末：3270 準拠端末＝1.0：1.3～1.5

2) SDS の変換可能率……逐次変換方式では，不可能な FCC や罫線の変換を可能としている．

3) 開発工数……プログラムの開発工数として，蓄積変換方式は逐次変換方式に比較して 8 倍程度の工数を要すると推定される．また，プログラムの総ステップ数は，約 5 倍の規模と推定される．

4) 汎用性および拡張性……3270 準拠端末と一口で言っても，実際にはメーカごとに独自仕様を盛り込んだ変種が多くみられる．とくに，漢字，罫線の設定，表示形式がさまざまである．MPC 1100 は機能別モジュール構造になっているので，作成済みのモジュールの再利用が容易である．また，UTS プロトコルと 3270 プロトコル間のデータ・ストリーム変換以外への適用は，そのモジュールの一部を作り換えることで可能となる．

4.　結　　論

異種プロトコル間の SDS 変換プログラムを設計する場合，逐次変換方式より蓄積変換方式の方が，とくにプロトコル間の差異が大きい場合に有効ではないかと考え，実際に MPC 1100 という“UTS と 3270 プロトコル間の SDS 変換プログラム”の評価を実施してみた．

逐次変換方式と蓄積変換方式を比較することにより，次のような評価結果が得られた．

1) 蓄積変換方式では，処理ステップ数は増加したが，以下の二つのメリットが得られた．
   ① データ・ストリームの変換率が向上した．
   ② 中間バッファを中心とした変換アルゴリズムが簡単になり，品質が向上した．
2) 蓄積変換方式でも，パフォーマンスの劣化は比較的少ない．
3) 蓄積変換方式における中間バッファの形式として，画面形式が望ましい．MPC 1100では，UTS プロトコルと 3270 プロトコル間の最も際立った特徴である文字属性と，フィールド属性間の変換を考慮したバッファ形式をとった．
4) 逐次変換方式は，パフォーマンスが高く開発工数が少なくて済むメリットがあるので，プロトコル間の差異が少ない場合に適する．

5. おわりに

本稿で述べたような画面データ・ストリーム変換プログラムは，大規模なアプリケーション・ソフトウェアに比較するとプログラムとしての規模も小さく，体系的に考慮されにくい種類のソフトウェアであろう．しかも，この種のソフトウェアの開発においては期間的な制約が厳しく，開発条件も限定されている場合が多い．その結果，過去の開発経験の蓄積が少ない分野となっている．本稿が，今後開発される同種のソフトウェアの設計開発の一助となれば幸いである．

最後に本稿の作成においてアドバイスを頂いた平野吉延氏と，MPC 1100 に関して援助を頂いた広田雅史氏に感謝する．

参考文献 [1] MPC 1100 (3270) ユーザ・ガイド，日本ユニバック，481205528-0，1986 年 9 月．
[2] UTS 4000 ターミナル・システム・ファミリ，UTS 50 シングル・ステーション解説書，日本ユニバック，481845009-0，1983 年 6 月．

執筆者紹介 庭山宣幸 (Noriyuki Niwayama)
昭和 31 年生．53 年 3 月群馬大学工学部情報工学科卒業．同年 4 月日本ユニバック(株)入社．主にコミュニケーション分野のソフトウェア・プロダクトの品質検査，テスト・ツール開発に従事．58 年より MAPPER 1100，MPC 1100 の開発・保守を担当．

論文 知識支援による設計ツール——ESP

ESP——A Knowledge-Aided Design Tool

J. F. King, E. M. Hushebeck

要 約 複雑でしかも時間と共に変化するシステムでは，複数の領域にわたる技術がその基盤となっているため，一人の人間がその全体を理解することは非常にむずかしい．しかし，最近，このような動的システムをよりよく理解する新しいアプローチとして人工知能技術が登場し，“知識支援による設計” (KAD, Knowledge-Aided-Design) ツールが使用可能となった．

KAD システムのあるものは，“知識によって増強された推論ツール” (knowledge-enhanced reasoning tool) であり，グラフィックスによって表現され，“拘束の伝播” (constraint propagation) によって解ける問題を取り扱うシステムとして定義される．本稿では，ESP (Expert System Planner) と呼ばれるこのような KAD システムについて述べる．

ESP は汎用のシステムであり，その知識はモジュールの中に含まれる．モジュールは，KAD システムのビルディング・ブロックであり，最も低レベルのオブジェクトあるいはアクターとして定義される．モジュールでは，入力，出力，プロトコル，モジュール間の関係，ルール，グラフィック表現，アニメーション，オブジェクトに関する各種の状態情報を記述する．また，モジュールは，メッセージ交信 (message passing) によって情報を交換する．なお，再使用可能モジュールは，モジュール・ライブラリに入れられ維持されるため，すべてのユーザがワーク・シート (work sheet) と呼ばれる画面の上で，アクセス，定義，移動，削除，接続，拘束条件の指定，グループ化，保存，実行などを行える．

ここでは，KAD システムに必要な機能，および各種の問題への ESP の応用について採りあげてみたい．

Abstract Complex, dynamic systems that span several of expertise challenge any single individual's capacity to comprehend the total system. The technology for novel approaches to a better understanding of dynamic systems is provided by the emerging discipline of Artificial Intelligence, through Knowledge-Aided Design (KAD) tools.

A KAD sysem is defined to be a knowledge-enchanced reasoning tool designed to solve problems that can be portrayed graphically and solved through constraint propagation. This paper describes a KAD system, called ESP.

ESP is a general purpose KAD system where knowledge is contained in modules. A module, the building block of the KAD system, is defined to be the lowest level object or actor. Modules define input, output, protocol, functional relationships, rules, appearance, animation, state and status for objects. Modules exchange information through message-passing. The reuseable modules are maintained in module libraries for all users to access, define, move, delete, connect, specify constraints, group together, save, and execute in a Worksheet region.

The specific features that constitute a KAD system and the application of ESP to some diverse problems will be discussed.

1. はじめに

いくつかの異なる技術領域を基盤に持つ複雑なシステムの理解は，挑戦的経験と言えよう．巨大で複雑なシステムを理解する能力を増強するために，KAD ツールが人工知能の研

究成果として登場してきた．KAD ツール（“知的コンピュータ支援設計ツール” (intelligent computer aided design tool) とも呼ばれる）は，問題解決のために記号表現やグラフィック表現を採用した“知識によって増強された推論ツール”として定義される．

KAD システムの有用性は，現実の物理的装置の実現性検討モデルの構築・修正・試験に時間と費用をかけることなしに，動的システムの理解を可能にする能力にある．

KAD システムの利用によって，対象システムの構成要素のグラフィック表示と構成要素間の相互作用の検討が行える．このほか，システム全体の性能はもちろん，構成要素に対する外部からの影響も予測可能である．

本稿で述べる KAD ツールは，ESP (Expert System Planner) と呼ばれ，問題を最も理解している技術者，科学者，プロジェクト・マネージャによって直接使用されるように設計されている．

使用者は，ESP の知識ベース・ライブラリ（知識パック：knowledge-pack とも呼ばれる）を利用して，とくに専門知識を持っていない技術分野のシステムをも記述できる．KAD システムは，工学，数学，コンピュータ科学，生物学，物理学，経営科学を含む広い応用範囲を有している．

2. ESP の課題

ESP のような問題解決ツールでは，次の三つが主要関心事となる．

1) 知識表現
2) 知識獲得
3) 効率

第一に知識表現が課題となる．コンピュータを利用して問題を解くためには，まず問題を形式化する（体系的に表現する）必要があるが，形式化は常に困難であった．AI は，パターン指向，フレーム，スクリプト，意味ネットワーク，述語計算等の多種の知識表現方法をもたらしたが，今日使用されている知識表現においても，まだ困難さが残っている．つまり，一つの知識表現だけではすべての状況に適用できないので，各種の方法を混用する傾向にある．

第二の課題は知識獲得である．知識獲得では，人間の知識の判別，獲得，付号化，検証，妥当性検討などが研究対象とされる．現在，時宜を得た効率のよい知識獲得法の検討が進められている．知識の汎用化（領域従属でない）と“再使用可能な符号化された知識”に関する基礎研究が進められている．また，知識の検証と妥当性の検討の問題では，獲得された知識の正当性と完全性を取り扱うが，対象領域が特定されない一般的なケースでは，より困難になる．

第三の効率の問題も重要である．これについては，迅速な問題定義と解決が現在も課題として残されている．本稿では，この三つの問題への ESP による対処で使用された技術について紹介する．

3. アプローチ

複雑な動的問題の形式化によく向いていると思われる知識表現法に，Luc Steels[7] のアプローチがある．Steels は，「推論システムは，メッセージ交換による対話機能を備えた複数のプロセッサを結合した大規模ネットワークであり，制御とデータを分散させた並列システムと考えるのが最もよい見方なのだ」と述べている．

Steels は，この考え方に基づき XPRT システムを MIT の AI 研究所で製作している．Steels は，XPRT 中のオブジェクトの表現としてフレームを採用し，お互いに通信するオブジェクトのネットワークとして問題を捉えており，XPRT を用いて機械や電気に関する各種の問題のモデル化に成功している．そして，この考えを Alan Borning[1] と David McArthur[6] は，さらに発展させた．

Borning は，Stanford 大学にいた 1980 年代初期に SMALLTALK を使用し，ThingLab というシステムを開発しており，これを拘束指向 (constraint-oriented) シミュレーション・システムと呼んでいる．そして，Borning は ThingLab によって，いくつかの幾何と物理の問題のモデル化に成功している．Borning は，これらのモデルをグラフィック・ディスプレイに表示したが，このことは問題の理解を容易にするのに大変有効であった．

一方，McArthur は，Rand Corp. 時代に Phillip Klahr と Sanjai Narain と共にオブジェクト指向言語 ROSS を開発している．McArthur は，Steels の取り上げた問題より複雑な問題，たとえば戦略的航空侵犯迎撃シミュレータ[4] や戦術的地上戦闘シミュレータ[6] の実現に Ross を使用し，成功を収めた．なお，McArthur は，この推論プロセスをモデル準拠推論 (model-based reasoning) と呼んでいる．

本稿の ESP は，拘束の伝播というアーキテクチャに基づいており，問題の定義にネットワーク表現アプローチを採用している．この方法を用いると，多くの問題を取り扱い可能な部品に分割でき，システム全体を“より小さな問題”（部分問題）のネットワークとみなせることが，明らかにされている．なお，この場合，各部分問題はネットワークを構成するノードとみなされ，しかも各ノードはお互いに情報交換機能を有するものと考えられる．このほか，ネットワーク表現を用いると，問題の形式化に関し，多段階の詳細さ（不完全なレベルのものさえも）を持つ表現が可能になる．さらに，ネットワーク表現は，メッセージ交信と併用することによって，仮説の検証と拘束の解決を可能にする．

さて，上述の部分問題をモジュールと呼ぶ．モジュールは，KAD 環境下での最低レベルのオブジェクト，あるいはアクターである．また，モジュールは，汎用 (generic) としての性格を持つことから，他のアプリケーションで再使用するため，ライブラリに入れられる．ライブラリ探索機能で支援された知識の再使用の機構は重要であり，これによってよく知らない専門分野の知識に対してもアクセス可能となり，知識獲得のサイクルに要する時間と費用を節減できる．このほか，これによって既存の知識を拡張できる．

このシステムで採用したネットワーク表現と，商用あるいは使用者が定義した推論ソフトウェアとを，容易に結び付けることができる．このため，述語計算，パターン指向，不確実な知識に基づく推論 (plausible inference) の機能を併せ持つ混合型ネットワーク・システムも，この知識表現と整合性がよい．

また，ThingLab における CAD 風のインタフェースは，使用者の理解力をかなり強化する．このため，ESP はそれを取り入れている．ESP では，使用者はグラフィック・オブジェクトの形でモジュールを表現し，各モジュール間の図形的な関係を設定することによって，問題の表現，問題を解くプロセス，求められた解の状態に至るまで画像で見ることができる．

このほか，伝統的なソフトウェアのアルゴリズムやデータベースの多大な資産を考慮し，ESP では従来のソフトウェアをネットワーク表現に変換する機能を有している．

4. システムの概要

ESP システムは，2個の構成要素，つまり ESP コアとモジュールで構成されている．ESP コアは，問題の定義・表示・解法の枠組である．なお，問題それ自身の実際の記述は，ユーザが定義するモジュールのコード部（モジュール・コード中）に組み込まれている．そして，このモジュール・コードは，ルールが推論エンジンの外部にあると同様な意味で，ESP コアの外部に存在する．

4.1 ESP コア

ESP コアは，モジュール・コードを支援するための次の基本命令を備えている．

1) 画面表示とメニュー
2) モジュール構造の定義
3) ユーザが作成したモジュール・コードへのアクセス
4) モジュールのインスタンシエイション
5) モデルのグラフィック・インタフェース
6) メッセージの経路指定とプロトコル記述
7) モデルの保存と復元
8) 支援パッケージの初期化
9) 知識獲得

ESP コアでは，ワーク・シートと呼ばれる主画面 (main display) と各モジュールで実行される基本操作のメニューを提供する．また，各モジュール固有の操作を定義しメニューに組み込むこともできる．

また，モジュールの基本構造を与えるのが，ESP モジュールである．ESP モジュールは汎用であり常時 ESP コア中に存在し，ESPコアの 外部で定義された他のモジュールと混用 (mix) できる．なお，ESP モジュールでは，次のメソッドが提供される．

1) マウス・センシティビティ
2) 接続（メッセージの経路指定に使用）
3) モジュールのグラフィック表現のワーク・シート中の位置とその移動
4) モジュールの状態と状況

ESP コアは，外部のモジュール・コードをアクセスしワーク・シート中にインスタンシエイトできるように，ライブラリの定義と探索のメソッドを用意している．また，ESP コアはワーク・シート中のモジュールに対するグラフィック表現を，定義する機能も有している．もっとも，モジュールで，このメソッドを必ずしも使用しないでもよい．このほか，メッセージの経路指定とプロトコルの処理の基本機能は，ESP コアによって与えられる．なお，メッセージ経路の接続はユーザが定義するが，メッセージの伝達と処理は ESP が行う．もっとも，これについてもユーザが定義できる．

また，ESP コアは，ワーク・シートの環境下で構築されたモデルを保存および復元する機構を備えている．保存関数は，それが呼ばれた時点でのモデルのスナップ・ショットをとり，一方，復元関数はそのスナップ・ショットをワーク・シートに書き戻す．

4.2 モジュール

ESP の基礎は，モジュールである．ユーザは調べたいと思うアイデアをモジュールとして表現し完全に制御しつつシミュレートしてみることができる．モジュールの例としては，ハードウェア・システムの設計者に対する論理ゲート，プロジェクト・マネージャに対するアクティビティ，認知心理学者に対するニューロンがある．このようなモジュール

の構造の中に，入力，出力，プロトコル，関数間の関係，プロダクション・ルール，画面，シミュレーションを行うのに必要な状態情報を取り扱う機構が含まれている．

モジュールは，フレーバー[8]のインスタンスであり，個々のモジュールは，ユニークで互いに独立である．また，モジュールは情報を“接続”（つまり，ソフトウェアによる通信リンク）によって共有する．このようなモジュール間の通信をメッセージ交信と呼ぶ．モジュールでは，内部で生成された情報や他のモジュールから要求された情報を他のモジュールに伝達する．

ESP においては，問題をより取り扱いやすい部分問題に分割する技術はとくに重要であり，これによってユーザはモジュールを使用して，より複雑な構造のシステムを構築できる．この概念を階層化 (layering) と呼び，わかりやすいレベルでのシステムの開発やテスト，新しく開発された構造の保存や，次にそれらを利用したより複雑な構造のシステムの構築などを可能にしている．

また，モジュールとしては，詳細さの異なるどんなレベルのものも開発できるので，モデルも詳細さの異なる複数のレベルの混じったものが構築できる．また，階層化プロセスで有用な ESP の機能は，グルーピング (grouping) である．グルーピングによって，数個のモジュールを一つのクラスに割り当てることが可能になる．しかも，このクラスのモジュールの属性は，以前のモジュールの属性全体を保持したままである．なお，生成されたこのクラスは，その後は単一のモジュールとして働くことになる．

5. 応　用　例

本章では，特定のアプリケーションの開発について述べる．アプリケーションとしては，スケジューリング，論理回路，水雷対策戦術計画を取りあげる．ここでは，モデルの定義・構築・実行の技術について説明する．そして，ESP のアーキテクチャが最も適している問題に焦点を当てる．

5.1 スケジューリング

本節では，問題の定義にネットワーク表現，問題の解決にパターン指向による推論を利用した例を示す．

ESP を使用してモデリングを行う場合の最初のステップは，解決すべき問題が何かを決定することである．たとえば，スケジューリングへの応用では，作業，作業間の関係，制約（資源やルール等の）が規定できることが望ましい．なお，ここで求めるべき結果は，開始（および完了）時期，使用される資源，費用である．そこで，本例では，モジュールとして activity（アクティビティ），time-line（時間尺度），resource（資源），cost（費用），sum（合計），label（ラベル），display（表示）に関するものが選択された（図 1）．

まず，アクティビティ・モジュールが選択され，開始と完了の時間，期間，必要とされる資源の型と量に関する情報を含むように設定された．また，選択された各アクティビティは，左端と右端にそれぞれ開始ノード（円）と完了ノード（円）を持つ水平線として表示された．なお，この線の長さは，作業期間を表している．また，本アプリケーションで採用されたスケジューリングの戦略は，「先行アクティビティがすべて完了するまでは，後続のアクティビティが開始できない」というものであった．このほかの制約としては，アクティビティが完了するためには，必要とする資源が利用できることが求められていた．

さて，各アクティビティの先行—後続関係は，“接続”によって表現される．もし，ア

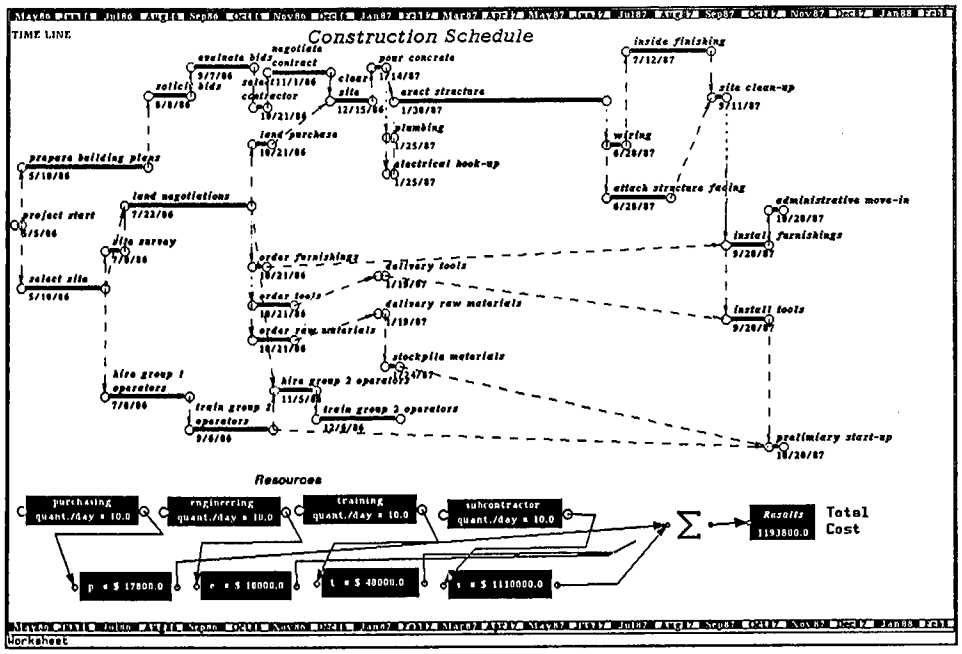


図 1　建設スケジュール

Fig. 1　Construction schedule

クティビティの完了ノードが，他のアクティビティの開始ノードに“接続”されている場合は，前者のアクティビティは，後者の先行アクティビティ（後者は，前者の後続アクティビティ）である．ESPの観点からは，先行アクティビティは，メッセージの発信者，後続のアクティビティは受信者である．どのアクティビティが，どのアクティビティに“接続”されているかという情報は，“接続”が完了した時点で各アクティビティ・モジュールのデータ構造に付加される．

また，各アクティビティ・モジュールでは，共通および個別ルールの定義を支援している．なお，本例でのルールの記法としては，OPS 5[2] の構文を採用した．もちろん，他の推論エンジン用の構文（たとえば PROLOG）も利用できよう．また，ルールはアクティビティ・モジュールがワーク・シートにロードされるとき，同時にロードされるテキスト・ファイル中に入れられ維持される．

時間尺度モジュールは，各アクティビティの表示線の長さに意味を持たせるために必要である．つまり，時間尺度は，時間に対するピクセルのスケーリング・ファクタを与えるものである．このモジュールは，汎用属性（たとえば，接続・移動等）が，ほとんど使用されていないという意味で異例である．なお，時間尺度は，ワーク・シートの先頭行か末尾行のいずれか，あるいは両方にラベル付きスケールとして表示される．

次に，資源モジュールが選択され，日ごとに利用可能な資源の型と量を含むように設定された．このほか．単位費用，資源の割り当て可能時間の最小値と最大値，費消される資源の最小値と最大値も資源モジュールの情報として含まれる．本システムでは，使用者は，いかなる戦略も選ぶことができたが[3]，ここでの資源割り当ての戦略は，初到着一初受益 (first-come-first-serve) を採用した．また，資源の全使用量は，内部（インスタンス）変数によって追跡された．なお，資源に関する外部との“接続”は，資源を必要とする個々のアクティビティ・モジュールに対してそれぞれ一つ，またプロジェクトの費用を追跡

する個々の費用モジュールに対してもそれぞれ一つ存在する．なお，資源は資源の型を示すラベル付きの塗りつぶされた長方形で表示されている．

費用モジュールの設計の主な目的は，資源の費用の累計を与えることにあった．費用モジュールでは，このモジュールと“接続”されるすべての資源モジュールの費用の累積値を表示する．このほか，費用モジュールは，資源費用の累積値の計算で利用できるように，利益とオーバヘッド等の項目も表示する機能を有している．なお，費用モジュールは，“接続”のために一つでも資源を与えられると，そのドル値が付され，塗りつぶされた長方形として自分自身を表示する．

ラベル，合計，表示の各モジュールは，このアプリケーションでは，あまり重要でなく，それぞれコメント，費用の合計の計算，費用合計の表示を行うのにのみ用いられている．

さて，本例の実行は次のような手順で行われる．

まず，ワーク・シート操作メニューの中の RUN コマンドがマウス選択され，アクティビティの一つがクリックされると OPS 5 用の作業記憶エレメントが作成される．一度，作業記憶エレメントが作成されると OPS 5 が実行される．そして，該当するすべての共通および個別のルールが呼ばれ，その結果がメッセージとしてワーク・シートに通信によって戻される．

5.2 TTL 論理回路

ここでは，問題の定義にネットワーク表現を，その解法に拘束の伝播を用いた例を紹介する．さて，本ケースでは各種の TTL 回路のシミュレーションを行う．したがって，ここで必要な基本モジュールは，論理ゲート (NAND, NOR, XOR など)，スイッチ，出力ランプである．なお，例としては，2 進—2 進化 10 進デコーダを取り上げる（図 2）．この例では，各スイッチのオン・オフを指定することによって 2 進数を設定し，それを復合

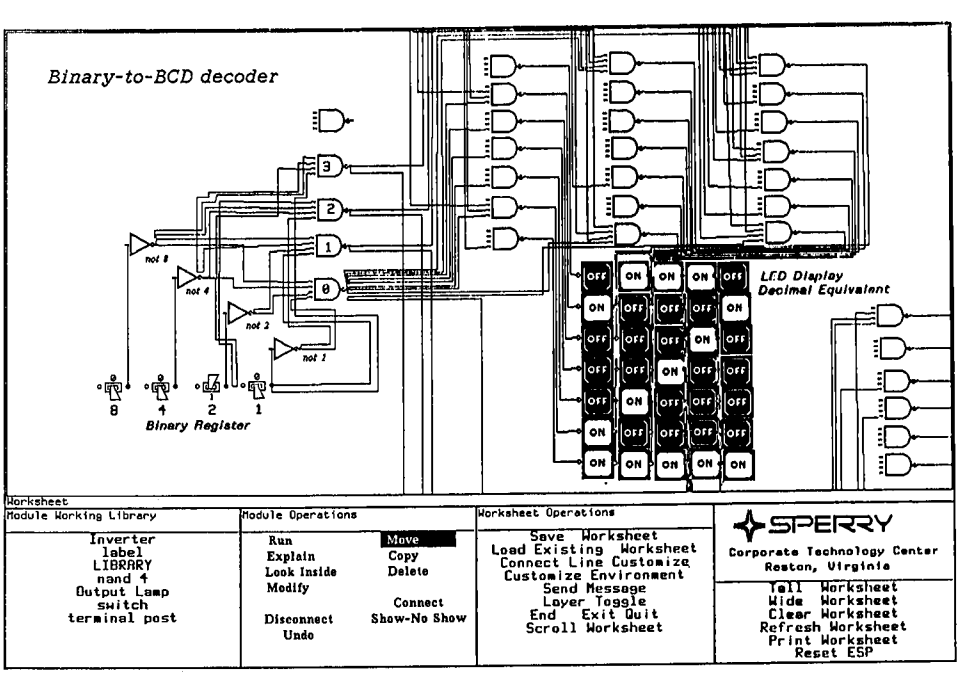


図 2 2進—2進化10進デコーダ

Fig. 2 Binary-to-BCD decoder

し，等値の10進記号に変換することが課題である．

このアプリケーションでのスイッチ・モジュールは，スイッチがオンかあるいはオフの状態かで，“0”か“1”を発信するように設計されている．各スイッチの記号を，オン（アップ）あるいはオフ（ダウン）の状態を表現するアイコンとして表示し，モデル中にアニメーションを導入した．同様に，出力ランプは二つのアイコンの内の一つを表示することによって，受けた記号が“0”か“1”かを示している．また，各アイコンの付近の小さな円は，他のモジュールに“接続”用エレメントを付加するためのグラフィック・アイテムである．

本例では，NAND ゲートとインバータも，一般の回路図でよく見られるような記号のアイコンを使用して表現されている．そして，これらのゲートでは，入力信号として“0”あるいは“1”を発信する．これらの信号はネットワークを伝播され，最後に一連の出力ランプ（シミュレーテッド LED ディスプレイ）上に10進数の画像（本例では“2”）として表示される．

本例の実行の手順は，次のとおりである．まず，RUN コマンドをマウス選択し，次にスイッチをクリックし，2進数をそのオン・オフによって表現すると，上記の一連のイベントが開始される．

5.3 機雷対策戦術計画

この問題は，問題の定義にネットワーク表現，問題の解法にパターン指向と拘束の伝播を用いた例である．本例は，アクティビティの期間が，問題の拘束条件に従属している点で，前出のスケジューリングの例と異なっている．

さて，本問題では，2人のシステム・アナリストの知識が利用された．一人は，機雷対策の専門家（海軍の退役軍人）で，他の一人は機雷対策の経験を持たない知識技術者であった．解決すべき問題は，使用者によって機雷が敷設された仮想の水路を掃海するための計画の立案，および任務実行のシミュレーションであった．なお，本例では機雷の敷設されている水域と，その敷設濃度が諜報活動によって得られていることが仮定されていた．そして，水路を掃海する資源（掃海艇）と，作戦開始時の掃海艇の位置も指定されていた．このほか，掃海資源の能力（たとえば，掃海艇の移動速度，機雷の探索速度，機雷の種別判定と安全化の能力など）は既知としていた．

システムの開発に当たって，機雷対策の専門家は作戦のシナリオを立て，一方，知識技術者は ESP で必要なモジュールを構築した．このシミュレーションでは，二つの視点(viewpoint) を使用しており，一つは空間的視点（図3）で，他の一つは時間的視点（図4）であり，それぞれ画面上で表示される．

なお，図4上で掃海艇（すなわち，MCM, MSH, RH 53-D など）に割り当てられた任務は，水平の棒で表現され，棒の長さで任務の期間を表現した．任務の期間は，資源の能力と成功のしきい値（ユーザが指定）によって決められた．また，水路はいくつかの部分水路に分割され，掃海艇はそれらの水域を掃海する任務を割り当てられた．そして，任務を遂行する順序は，一つの任務の完了（左端のノード）から他の任務の開始（右端のノード）への矢印によって指定された．なお，各水域での機雷の種類判別と安全化作業の回数は，乱数発生関数によって割り当てることもできたが，今回はユーザが指定した．このほか，本演習の作戦のルールは OPS 5 で記述された．

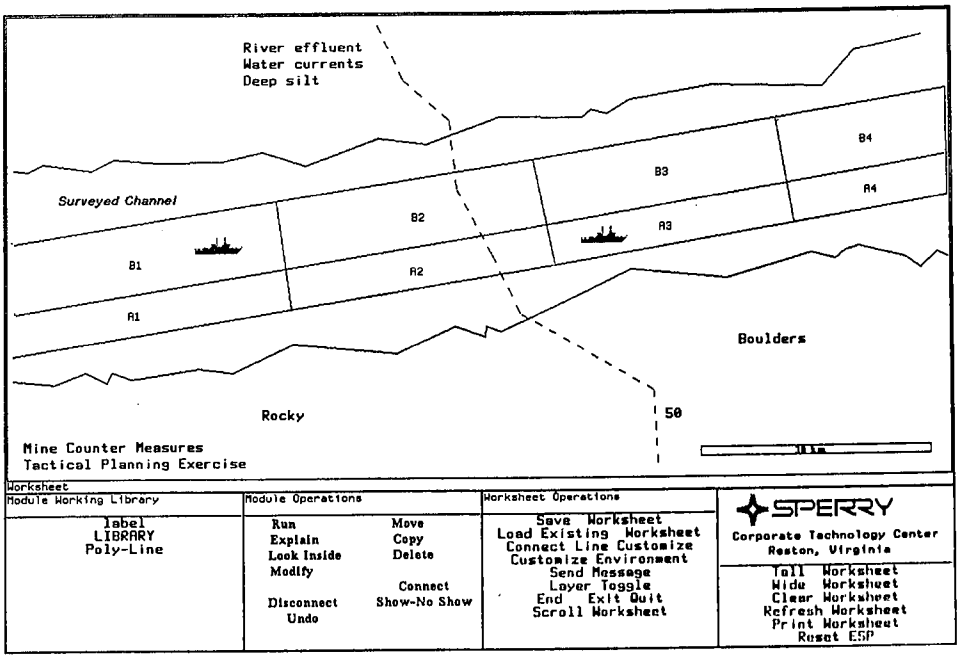


図 3 機雷対策演習の地形

Fig. 3 MCM exercise configuration

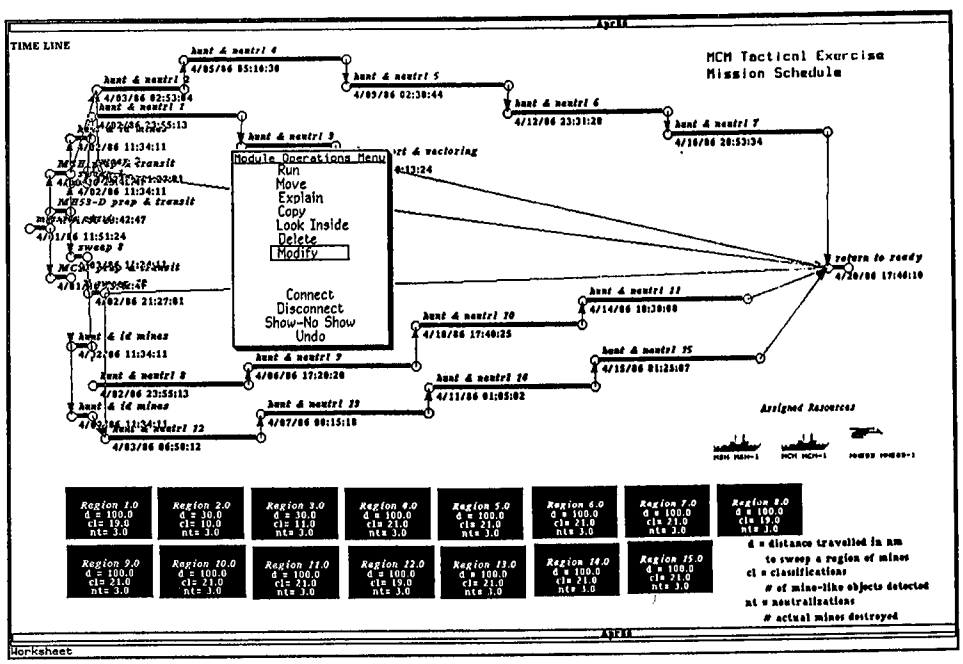


図 4 機雷対策任務スケジュール

Fig. 4 MCM mission scheduling

6. お わ り に

本プロジェクトの結果は，「問題をモジュール分割して知識のチャンク（塊）に還元するという考えが，有効な知識獲得の技術である」という主張を確信させるのに十分役立っ

た．また，一般にこの種のシステムを利用して問題を解くという行為を通じ，当然の帰結として知識の内容を付け加えることになる．実際，ESP が LAN 上で使用できるようになり，モジュール・ライブラリが中央に集中化された時点で，すべての使用者が新しく作成されたモジュールにアクセスし，すばらしい問題解決環境を作り上げたことを目撃した．さらに，現在ではより多くの人が使用することによって，知識の検証と妥当性チェックの機能が備えられるようになっている．

本プロジェクトの当初の目標は，問題の形式化と理解のプロセスをできるだけ明瞭にしようということであった．ESP で採用されたモジュラーなネットワーク表現は，自然で柔軟性に富むため，速かに理解されるようである．

本稿の最後に，ESP で実証されたように思われる原理をまとめると，次のようになる．

1) 大規模で複雑な問題は，まず最初に複数の比較的小規模な単純な部分問題を注目することによってアプローチされ得る．
2) 複数の小規模な問題を継ぎ合わせることによって，より複雑な問題を近似できる．
3) 小規模な問題は，ローカルな結果をより大規模で複雑なシステム（たとえば，ネットワーク）の他のメンバー（問題）へ伝えることができなければならない．
4) 見ることのできる問題は，そうでないものより容易に理解され得る．

（技術研究部　高橋　肇 訳）

参考文献

[1] A. Borning, "The Programming Language Aspects of ThingLab, a Constraint-oriented Simulation Laboratory", *ACM Transactions on Programming Language and Systems*, Vol. 3, No. 4, Oct. 1981, pp. 353～387.

[2] L. Brownston, R. Farrell, E. Kant, N. Martin, Programming Expert Systems in OPS 5, Addison-Wesley Publishing Company, Reading Mass., c 1985.

[3] J. F. King, "Heuristic Model for Resource Allocation in a Contention Environment", *Proceedings of the Instrument Society of America*, Houston, Texas, Oct., 1984.

[4] P. Klahr, D. McArthur, S. Narain, "SWIRL: An Object-oriented Air Battle Simulator", *Procceedings of the Second Annual National Conference on Artificial Intelligence*, Pittsburgh, 1982, pp. 331～334.

[5] P. Klahr, J. Ellis, W. Giarla, S. Narain, E. Cesar, S. Turner, "TWIRL: Tactical Warfare in the ROSS Language", Rand Publication, R-3158-AF, Oco., 1984.

[6] D. McArthur, P. Klahr, S. Narain, "ROSS: An Object-oriented Language for Constructing Simulations", Rand Publication, R-3160-AF, Oct., 1984.

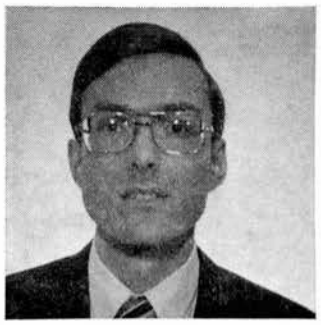
[7] L. Steels, "Reasoning Modeled as a Society of Communicating Experts, MIT Artificaial Intelligence Laboratory, TR-542, Jun., 1979.

[8] D. Weinreb, D. Moon, "Objects, Message Passing, and Flavors", *Lisp Machine Manual*, Jul., 1981, pp. 279～313.

執筆者紹介　J. Fred King

Kansas 州の Pittsburg 州立大学で数学を専攻し，B. A. と M. A. を取得．Houston 大学で数学の Ph. D. を取得．システム工学に関し 15 年の経験を有しているが，最近の 5 年間は AI の応用に従事．現職は，Reston にある Unisys 社の Corporate Technology Center の技術スタッフとして，知識ベースによる問題解決ツール ESP (Expert System Planner) 開発のリーダを務めている．なお，ESP は，同氏が，Houston の NASA Johnson Spacecraft Center で，スペース・シャトル関連のアプリケーション用に開発した RPMS (Resource Planning and Management System) の拡張である．

このほか，Houston 時代には，Artificial Intelligence Laboratory の設立に参画したほか，AI，パターン認識，発見的問題解決に関する研究プロジェクトの主担当者であった．



Eric M. Hushebeck

Pennsylvania 州立大学と Johns Hopkins 大学で各々コンピュータ・サイエンスの B.S. と M.S. を取得. 過去8年間は, 大型コンピュータとミニコンピュータのシステム・ソフトウェアの開発に従事. 現在, Unisys 社の Reston の Corporate Technology Center の技術スタッフとして働いており, Unisys と Symbolics の Lisp マシン上の ESP の設計と実現を担当している.

Sperry 社 (現 Unisys 社) には 1982 年に入社, 現在までの主な仕事はフライト・シミュレーション・システムの開発であり, それ以前は National Security Agency で国家機密会話型データベース・システムの開発に従事していた.

論文

ラピッド・プロトタイピングを支援する知的ソフトウェア開発環境

A Knowledge-Based Software Development Environment for Rapid Prototyping

S. Pontecorvo, J. Krohnfeldt

要 約 ラピッド・プロトタイピング用のプログラミング環境を提案する．この環境は，既存のプログラム開発と検査のためのツールと方法を一つの枠組の中にまとめたものである．本稿では，このシステムを実現するための専門家システムを提案している．なお，このシステムの重要な部分は LISP 機械の環境のもとで実現されている．

Abstract This paper describes the design of a rapid prototyping workstation. The workstation is a collection of software tools which combine to provide a powerful environment for software development. The tools provide top-down and bottom-up program design capabilities, access to reusable software modules, consistency, syntax, and spelling checkers, and code debugging modules. A unique aspect of the workstation design is the software bus, the central server with which all of the software tools communicate. The software bus provides a uniform way for diverse tools to exchange information. This is similar to the Unix philosophy, in that it encourages collections of tools that can communicate with each other. It improves on this idea in that it strives for much more sophisticated communication and runtime control between tools. The workstation relies on ideas from CAD, CAE, and LISP machine programming workstations. Some of the tools are knowledge-based tools, such as MicroScope, a knowledge-based code analysis tool. It is expected that the system will continue to increase in power as such techniques become more commonly used in the system.

1. はじめに

本稿では，ラピッド・プロトタイピング用のワークステーションの設計について記述する．ワークステーションは，ソフトウェア開発に強力な環境を用意するために統合化された一つのソフトウェア・ツール・セットである．ツールはトップダウン型とボトムアップ型プログラム設計の機能，再利用可能なソフトウェア・モジュールへのアクセス，一貫性，構文および綴りのチェッカー，コードの虫とりモジュールなどを備えている．ワークステーション設計の特徴の一つは“ソフトウェア・バス”で，すべてのソフトウェア・ツールがそれと通信する中心的なサーバである．本ワークステーションでは，ソフトウェア・バスによって異なる多種多様なツールが情報交換するための一定の方法を提供している．このことは相互通信可能なツールのセットを推奨している Unix 哲学と似ている．ツール間のより高度な通信と実行時制御を目指している点で，ソフトウェア・バスは Unix の考えを改良したものとなっている．本稿のワークステーションは，CAD, CAE および LISP 機械のプログラミング・ワークステーションなどにおける考え方に負うている．ツールのいくつかは，知識ベースによるコード分析ツール MicroScope のような知識ベース・ツールである．本システム内で，そのような技術がより一般的に用いられるようになると，システムの能力が次第に強力になってゆくものと思われる．なお，この開発プロジェクトの目標は，

1) ラピッド・プロトタイピングのために洗練された便利な機構を用意することによっ

* Unix は米国 AT & T Bell 研究所で開発した OS の名称で，AT & T がライランスしている．

て，プログラマの生産性を高めること．

2) 多数のコード群の作成，分析および管理に知識ベースの技術を適用すること．

3) ツールを一定の方法で結合し，実験する作業台を用意すること．

4) ソフトウェア工学の新技術が利用可能になったとき，それを取り入れられる拡張可能な枠組を用意すること．

であった．

さて，ソフトウェア工学の分野では，プログラムの生産性や開発・保守の研究を何年間も推進し，プログラム設計のための環境を改善してきた．人工知能の研究[1][2][3]におけるいろいろなツールも，大規模なプログラミング・システムの開発と検討を行うプログラマに新手法を用意している．これらのツールと技術のねらいは，プログラム開発者からプログラミングの日常的な細かな作業を解放し，より興味深くかつ高度な仕事を行えるようにすることである．筆者は，種々のプログラム開発のためのツールや技術の成果を数多く集めて結合する枠組を用意してくれる作業台を考えた．そしてこの作業台を利用するとプログラマは既存ツールをプログラム開発に利用でき，新ツールが利用可能になるとすぐに環境の中に取り入れることができる．本稿では，ラピッド・プロトタイピング用のワークステーションの設計について述べ，このシステムの実現の進捗状況についても報告する．

2. 方　　法

多数の方法がプログラム開発作業と関係するようになってきた．ラピッド・プロトタイピング・システムは，その能力を発揮するためにこれらの方法の中から，いくつかを取り入れている．重要な方法を要約すると以下のようになる．

1) ラピッド・プロトタイピング……廉価な記憶装置や高速シンボリック・プロセッサ (LISP 機械等) が，ますます利用できるようになるにつれ，ラピッド・プロトタイピングは今までの手間のかかる方法に代わる実際的な方法となった．会話型かつインタプリーティブな言語の環境の下で，再利用可能コード，高級言語による仕様，高級な虫とりツールを活用することによって，ラピッド・プロトタイピング環境はソフトウェア・システムの迅速な開発を支援する．

2) 知識ベース・システム……知識ベース（あるいは‘専門家システム’）技術は，複雑な問題や曖昧にしか述べられていない問題を扱う時の大きな潜在的力の源泉として浮上しつつある．このシステムでは再利用コード・データベースの維持，コードの分析と最適化，システム開発の管理のために知識ベース技術を利用する[3][4][5][6]．

3) オブジェクト指向プログラミング……ラピッド・プロトタイピング環境の設計で中心となるのは，オブジェクト指向プログラミング[7][8][9]の考えである．このプログラミング環境では，プログラムやデータを表すのにオブジェクトを利用する．オブジェクトは関係する実体をまとめて，それらに対する一定のインタフェースを（‘メソッド’によって）用意することでデータ抽象を与える．さらに，‘クラス’階層によって階層構造も実現できる．

4) 層別設計 (layered design)……一つの設計に対する代替案，あるいは旧版のプログラムに対する新版のプログラムは重複する情報をたくさん持っている．設計に対する変更を層[2]として実現すると，それぞれの層は前層から重複する部分の情報を引き継ぐことができ，プログラム開発を漸進的 (incrementally) に進められる．また，同じ情報を何版にもまたがって持たなくても，複数の版を同時に持つことができる．オブ

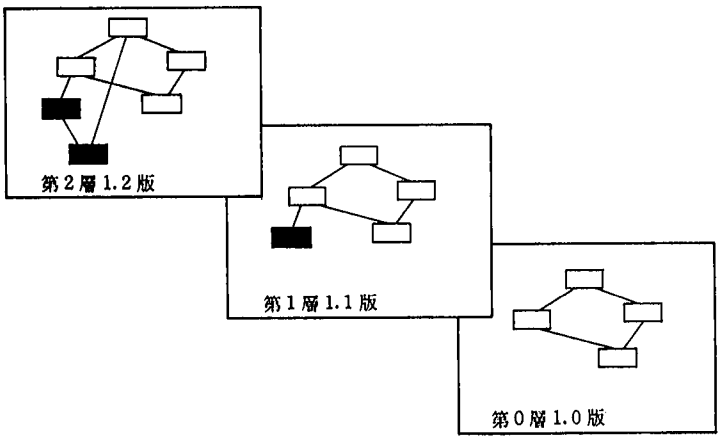


図 1　層別プログラム設計

Fig. 1　Layered program design

ジェクト指向システムは，クラス階層と抽象化能力を備えているので，層別ソフトウェア・システムを実現できる．

5)　CAD (Computer Aided Design)……この 10 年間，集積回路の設計に使うツールが開発され，大規模で複雑なチップの設計とそのプロジェクト管理に利用されてきた．これらの CAD ツールには，ソフトウェア設計に必要な‘もののたとえ’(metaphor) がたくさんある．たとえば，トップダウン型およびボトムアップ型の設計が支援されている．また，混合水準型 (mixed-level) の開発やシミュレーションができるので，設計階層をまたがって，いろいろな抽象の水準で設計ができる．このほか，図形処理や設計分析ツールのおかげで設計の品質や正しさを手早くフィードバックすることができる．以上のような CAD ツールの‘たとえ’を本ラピッド・プロトタイピング・ワークステーションに取り入れる．

6)　アイデア処理 (idea processing)……アイデア処理とは，記号データの構造的な操作に対する一般化である．原始的なアイデア処理系は単なる下書きの道具にすぎないが，もっと洗練されたアイデア処理系は，任意の振舞いや抽象的な記号情報も扱える．そのようなシステムとして Xerox Note Cards システム[10]がある．ノート・カードは，1 個のアイデアを表す（アイデア・サイズ）テキストや図形を含むオブジェクトである．ノート・カード型（テキスト・カード，スケッチ・カード，問い合せカードなどがある）の継承階層によって異なった種類のノート・カードが定義される．また，ノート・カードをリンクして任意の構造を構成できる．ラピッド・プロトタイピングの環境では，ノート・カードの‘たとえ’はプログラム，プログラムの構成単位，プログラムの虫とりなどの開発の全段階に適用される．

ラピッド・プロトタイピング・ワークステーションは，知識表現の基本単位を実現するためにオブジェクトを利用している．このほか，このワークステーションは他分野の技術的成果を利用して，プログラム開発工程に大きな力をもたらすことを狙っている．

3.　ラピッド・プロトタイピングの環境

ラピッド・プロトタイピングの環境は，LISP 機械[11]のワークステーションと LISP で書いたソフトウェア・システムとからなる．ソフトウェア・システムは，四つの構成要

素，つまり利用者インタフェース，コード・データベース，ソフトウェア・バス，およびツールの集合体である．利用者インタフェース，コード・データベース，ソフトウェア・バスは，ソフトウェア開発環境の‘核’システムを構成している．ツールは，プログラム分析と保守作業を行う専門のプロセッサ，すなわち専門家である．またツールはソフトウェア・バスにつながっており，ツールと核システムまたはツール同士の通信は，このバスを経由してなされる．特殊な型のツールとして知識エージェント (Knowledge Agent，以下では KA と略する) がある．これは，特定のプログラム開発作業を行う小規模の専門家システムである．ソフトウェア・バスも小規模な専門家システムで，現在バス上にあるツールとそれらとの通信方法を知っている．

利用者インタフェースは，プログラムの設計，虫とり情報の制御と表示，およびプログラムの標準出力を含むすべての水準でのシステムとのやりとりを支援する．利用者インタフェースは，LISP 機械の環境から次のものを利用している．それらは，“ZMACS” 編集プログラムとそのモード・パッケージ，“Zetalisp” ウインドウ，虫とりとコマンド処理パッケージ，LISP 機械の強力なグラフィック・プリミティブ・セットである[12][13][14]．利用者インタフェースには標準のアクセス・プロトコルがあるので，いろいろなアプリケーションがその機能を利用できる．

コード・データベースは，‘ユニット’と呼ぶデータ・オブジェクトからなる知識集約的なデータベースである．再利用可能なソフトウェア・ルーチンのコードのほかに，そのルーチンの用法・機能・変形・相互接続の知識を持つ．このコード・データベースは文脈によって番地付けができ，最適な方法で特定のコードが見つかるように編成されている．コード・データベースはクラス階層になっている．なお，コード・データベースは，‘ユニット’を常にアクセスできるように特別のプロトコルを用意している．

ソフトウェア・バスは，ソフトウェア開発環境で中心的な役割を果たす．これは，伝統的なハードウェアのバスとよく似ており，相対的に独立のものが相互に通信し，共通の資源を共有できるためのプロトコルを用意している．このことによって将来開発されるソフトウェアとの上位互換性が可能となる．

ツールと KA は，ソフトウェアの開発中の大半の作業を行う．KA はソフトウェア開発過程の特定の仕事のみ行う専門家システムである．それは，手続き，発見的方法，デー

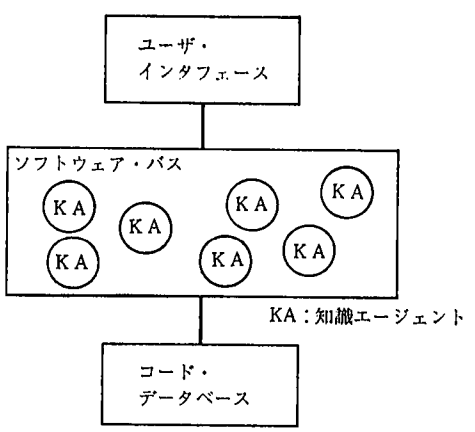


図 2 ソフトウェア開発環境システムの体系

Fig. 2 Software development environment system architecture

タの形式で特定の領域の知識を持ち，主としてソフトウェア開発者の相談役や補佐役として働き，その領域での未解決問題の解決と，その領域に関係するすべての制約の充足を保証する．いくつかの KA は自律的に走るが，他のものは利用者や他の KA と絶えずやり取りする必要がある．

4. コード・データベースとプログラム開発

コード・データベースは，プログラムの説明と情報の集まりである．資源が許す場合は，それは実際には異なったデータベース・ファイル（あるいはサーバ）上に置かれ，分散した構造をとってもよい．この場合，大域的なデータベースは，アプリケーションに共通して再利用される機能の貯蔵庫となり，個々のアプリケーションは，現在開発中のコードを含む一つ以上の局所的データベースを持つ．コード・データベースは，アイデア・サイズのプログラム・ユニットの層別階層 (layered hierarchy) となっている．プログラム・ユニットは，現在開発中の基本的な設計要素を表しており，次のようなものである．

1) プログラム，あるいはプログラム部品についての一つのアイデアまたはアイデアの集合
2) 一つあるいは複数のプログラム・ユニットについての関係集合
3) プログラムあるいはプログラム部品を擬似コードで記述したもの
4) プログラムあるいはプログラム部品をオブジェクトの水準で記述したもの
5) 複数プログラム・ユニット間の接続関係を書いた詳細な仕様
6) ある言語で書かれたプログラムあるいは副プログラム

プログラム・ユニットは，プログラム開発のアイデアから特定の原始コードに至るまでの全過程を表すのに使われる．このユニットを実現すると，ラピッド・プロトタイピング・システムは強力で柔軟性の高いものとなり，システム全体の統一的な基礎を得る．ユニットは，設計水準をまたがっても不変なので，設計を異なる抽象水準で同時に行うことができる．たとえば，設計のある部分がまだ非常に抽象的なアイデア段階なのに，ある部分はすべてコード化されていてもよい．設計済みの二つのサブシステム間のインタフェース情報を与えると，利用者はあたかも一つの完成したシステムとして実際に実行できる．このことから設計作業の迅速なフィードバックが可能となり，ソフトウェア開発周期を一段と短縮できる．

典型的な設計プロセスは，次の 4 段階を含む．すなわち，ブレイン・ストーミング，設計，洗練，検査である．この 4 段階は通常，任意のところに点在し，互いに関係を及ぼしあう．ブレイン・ストーミングは，アイデアの探索，代替戦略の考察，問題に関連した情報を大まかにまとめることなどである．設計はアイデアのより具体的な表現への変換，具体的な表現の上における関係の導出，導出された関係を結んで粗いネットワークにまとめることなどを含む．洗練は抽象表現をより具体表現に，粗結合のネットワーク関係を完結した構造にすることである．それはアイデアを擬似コードに，擬似コードを原始コードにする過程である．検査はノード間の関係が矛盾していないこと，ノードの内部の挙動が正しいことを証明することである．もっとも高位の水準（アイデア）での検査の典型例として関係についての無矛盾性維持 (truth maintenance) の作業があり，低い水準ではプログラム実行によく似ている．設計プロセスは，アイデアを次のような段階を経て原始コードに変換する．

1) ブレイン・ストーミング……バラバラのアイデアを集めてユニットにする．

2) 「Y が真のとき，X は決して真ではない」とか「事象 W は事象 V の発生する時点より前に発生する」のようなユニット間の簡単な関係を設計する．
3) 2)で設計した関係に関し，その無矛盾性維持を検査する．
4) アイデア・ユニットを擬似コード・ユニットに洗練する．
5) アイデア間の関係を洗練することで，擬似コード・ユニット間の関係を設計する．
6) 記号実行または直接データ入力によって擬似コードを検査する．
7) 擬似コード水準のユニットと関係を，オブジェクト水準のユニットと関係に洗練する．
8) 直接実行または記号実行によってオブジェクト・ユニットを検査する．
9) 必要ならオブジェクトを手続き的な原始コードに洗練する．
10) 直接実行によって手続き的なコードを検査する．

このプロセスのいろいろなステップにも，設計の問題がありうることを思い出してほしい．このために異なる抽象水準間のインタフェースをとるには，相当に複雑な検査ツールが必要となる．現時点では，利用者がほぼ完全なインタフェースを作成しなければならない．また，利用者がシステム内の洗練操作をすべて進めなければならない．知識ベース・ツールがもっと精巧なものになれば，洗練と検査の多くのプロセスを知識ベース・ツールが補助することになろう．

5. ソフトウェア・バスとツールの利用

ソフトウェア・バスは，本開発の中でも革新的なものの一つで，システムの全構成要素の中枢的な通信設備となっている．中央処理装置と周辺サブシステム間に通信チャネルを備えている機械体系内のハードウェア・バスに似ている．ソフトウェア・バスは，現在使用中のツールと KA の情報を持つ小規模の専門家システムである．ソフトウェア・バスは，ツールの扱い方を教える型板をツールごとに持っている．Unix のツールは，流れ (stream) に基づいており，逐次的な方法で扱われている．ソフトウェア・バスでは，ツールの振舞いの制御とツールからの出力を受けとるためにソフトウェア・バスとツールとの間にパイプを用意している．KA ツールはもっと柔軟で，ソフトウェア・バスが KA ツールの規則や知識ベースを直接アクセスして KA ツールを制御できる．場合によっては，ツールは利用者からの問い合せに応えられるような会話型のものでもよい．この場合，ソフトウェア・バスは，利用者インタフェース・サブシステムとやり取りできなければならない．

ツールの支援のほかに，ソフトウェア・バスはいろいろなツールからの要求に応じてコード・データベースへの質問や更新を行う．ソフトウェア・バスにとって，コード・データベースは単にバスにつながっている機器にすぎず，他のものと同様に型板によって扱うことができる．ソフトウェア・バスは，スケジューリング・アルゴリズムを持っており，稼動中のツール間の調整を行う．ツールには稼動時間の長さに基づく優先順位，扱っている問題に対する重要度，システムに占める地位（たとえば，コード・データベースはシステム上でのその重要性によって高い優先順位を持つ）などが付与される．ソフトウェア・バスは抽象時計上で稼動し，時計の周期ごとに調整を行う．もし利用者が高い優先順位でツールを起動する（たとえばコンパイル）と，スケジューリング・アルゴリズムによって妨げられることなく完遂される．

ソフトウェア・バスは，ソフトウェア設計プロセスに強力な分散型の問題解決方法を与

える．ツールはバス上にあってデーモン（条件が満たされれば動く）であったり，コンサルタント（依頼がされれば動く）であったりし，設計の進展に合わせて設計者に入力や忠告を与える．これは設計者を肩越しに観察していて，必要なときに働く専門家の集団を設計者に与えることになる．このシステムが成熟するにつれ，この専門家は設計の協力者としてより有用なものとなるだろう．

6. ツールと KA

ソフトウェア・バスに，KA とツールが同時にいくら接続されていてもよい．なお，典型的な KA（まだ実現はされていないが）のイメージは，次のようなものである．

1) 構造設計者……開発中のソフトウェア・システムのブロック・レベルでの構造の設計や仕様化を支援する．一つの箱をいくつかの部分箱に分割し，それらの関係の仕様を作るということを繰り返すことによって，利用者は開発中のソフトウェア・システムの制御構造を設計する．構造設計者は，適宜利用者を補助しながら従属関係，制約，設計階層を追跡する．
2) コード設計者……構造設計で定義した機能ブロックを満たすコードの開発を補助する．コード設計者は，単に構文を検査したり，利用者が生成したコードの一貫性を維持したりするものかもしれないし，あるいはまた設計要求に合う再利用可能コードを見つけることを補助するかもしれない．コードの仕様は，自然語に近い擬似コードから十分にテストされた所与のプログラミング言語による原始コードまでいろいろありうる．理用者が抽象的な仕様から原始コードへ段階的に洗練するのを補助する．
3) 表示設計者……利用者が，開発中のシステム出力を選択し，形式を決め出力データの効果と視覚性を最大にするのを手伝う．
4) 実行管理者……開発中のシステムの実行を制御し，そのシステムの虫とり機能にインタフェースを付与する．実行管理者は，実行環境を絶えず注視しており，構造とコードの仕様によって与えられたいろいろな抽象水準での実行を追跡できる．知識ベースによるプログラム実行を扱ったものに Krohnfeldt[15]の論文がある．

7. 現　　状

現在，本ラピッド・プロトタイピング・システムの設計は，いくつかの局面に分け，それぞれ併行して進めているが，できるだけ既存のツールやユーティリティを利用しようとしている．とくに，編集プログラム，LISP 言語のインタプリート，複数言語の支援，データやファイルのブラウジングには，LISP 機械の環境を利用している．LISP 機械に用意されているウインドウとメニューの機能をシステムの他の部分の開発に利用している．また，flavors パッケージは，オブジェクト指向プログラミングの機能を用意してくれている．使っている専門家システムのツールは，GIST (General Interactive Shell Tools) プロジェクト[16]で作成されたもので，前向きおよび後向き推論用のルール・インタプリタ，フレーム・システム，汎用図形インタフェース，知識ベース作成用のフレーム編集プログラムなどが入っている．アイデアの処理系は，現在開発中である[17]．

現時点で，三つの KA が実現されている．MicroScope[5] は，ルール・ベースの原始コード分析ツールである．これは，相互参照の機能と原始コード水準でのプログラムの詳細分析を与える．Expector[15]はルール・ベースの検査ツールである．これはルール・ベースによるインタプリタの制御下でプログラムを実行し，その実行を詳しく監視することがで

きる．事象 (event) 駆動型のプログラム Probes[18] もまた，ルール・ベースによる検査ツールである．MicroScope と Expector は，LISP とCコードの上で作動する．プログラム Probes は，LISP コード上だけでしか動かない．以上のツールは，Utah 大学計算機科学科の PASS (Portable Artificial Intelligence Support Systems) 研究グループ（本プロジェクトの協同メンバでもある）によって開発された．

MicroScope は原始コードを階層的，オブジェクト指向的手法で表現するので，コード・データベースのモデルを与える．Expector は，プログラムの時制と振舞いに関する情報を表現するためのモデルを提供する．

利用者インタフェースとそのプログラム設計プロセス自身は，アイデア処理系の開発によって推進されつつある．アイデア処理系は，今のところオブジェクト間のネットワーク関係を LISP 機械の画面上に表示している状態である．これらのオブジェクトとオブジェクト間の関係を編集でき，アイデアのネットワークの保存や復元もできる．ルール・システムを目下アイデア処理系に組み込んでいるところであり，アイデア処理活動（ブレイン・ストーミング，設計，洗練，検査）を制御する専門家システムは間もなくできると思う．

まだ実現されていないものは，ソフトウェア・バスのプロトコルである．現在，いくつかの専門家ツールをシステムに統合中であるが，もしいろいろなツールが完全に統合化されるためにはソフトウェア・バスが必要となる．なお，いくつかの UNIX ツールと他の LISP 機械のツールもソフトウェア開発環境に取り入れたいと思っている．

8. おわりに

ラピッド・プロトタイピング・プログラミングの環境を概観した．この環境は，既存のプログラム開発と検査のツールを一つの枠組の中にまとめたものである．このシステムを実現するために，多くの専門家モジュールを提案し，システムの重要な部分を LISP 機械の環境の中で実現した．将来の仕事に次のようなものがある．

1) ソフトウェア・バスの開発
2) ソフトウェア・バスのプロトコルを利用する新ツールと KA をシステムに統合すること
3) 図形と利用者インタフェースの問題点の調査
4) コードの設計と洗練のための専門家システムの開発
5) 高水準のアイデアの検証のための無矛盾性維持機構の付加
6) 設計仕様の各水準間をまたがる（たとえばアイデアから原始コードまで）インタフェースの開発

（ソフトウェア生産技術一部 板倉 教 訳）

参考文献 [1] C. Rich, H. Shrobe, "Initial Report on a LISP Programmer's Apprentice", *IEEE Transactions on Software Engineering*, IEEE, New York, NY, Nov., 1978, pp. 456～467.

[2] I. Goldstein, D. Bobrow, "A Layered Approach to Software Design", Tech. report CSL-80-5, Xerox Palo Alto Research Center, Dec., 1980.

[3] R.C. Waters, "The Programmer's Apprentice: Knowledge Based Program Editing", *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 1, Jan. 1982.

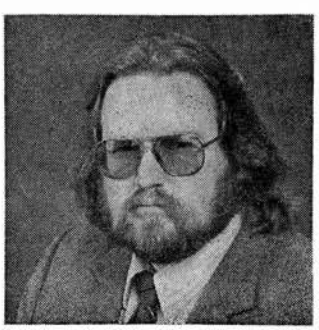
[4] D.R., Barstow, Knowledge-Based Program Construction, North-Holland, New York, NY, Programming Language Series, Vol. 6, 1979.

[5] J.J. Krohnfeldt, R.R. Kessler, "MicroScope: Rule-based Analysis of Programming

Environments", PASS Group OpNote 85-04, University of Utah, Apr., 1985.
[6] J. Krohnfeldt, "MicroScope: Rule-based Analysis of Programming Environments", Master's thesis, University of Utah, Sept., 1985.
[7] US Dept of Defense, "Reference Manual for the ADA Language", Report ANSI/MIL-STD-1815A, United States Dept of Defense, 1983.
[8] A. Goldberg, D. Robson, Smalltalk-80: the Language and its Implementation, Addison-Wesley, Reading, MA, 1983.
[9] A. Goldberg, The Influence of an Object-oriented Language on the Programming Environment, McGraw-Hill, New York, NY, 1984, pp. 141〜174.
[10] F. Halasz, "Xerox Notecards", from Arpanet AIList.
[11] D. Weinreb, D. Moon, Symbolics, Inc., Lisp Machine Manual, fourth ed., Cambridge, MA, 1982.
[12] K. Marrin, "Lisp Software Development Environments Increase Programmer Productivity", EDN, Aug., 1984, p. 89.
[13] K. Marrin, "Software Tools, Tailored Architectures Extend Lisp and Smalltalk Capabilities". EDN, Oct., 1984, p. 53.
[14] K. Marrin, "Lisp Development Tools Help Programmers Slash CAE Software Production Cycles", EDN, Nov., 1984, p. 129.
[15] J. J. Krohnfeldt, R. R. Kessler, "The Expector: Analyzing the Dynamic Behavior of Lisp Programs", PASS Group OpNote 85-05, University of Utah, Jun., 1985.
[16] M. S. Pontecorvo, "GIST: General Interactive Shell Tools", Working Paper, Sperry CTC, Jun., 1984.
[17] M. S. Pontecorvo, "Idea Processing-Concepts, Extensions, and Application", Working Paper, Sperry CTC, Apr., 1986.
[18] H. C. Carr, R. R. Kessler, "Event-driven Program Probes with State", OpNote 86-02, University of Utah PASS Group, Feb., 1986.
[19] D. Bobrow, M. Stefik, Xerox Corporation, The Loops Manual, 1983.
[20] H. Leiberman, "Seeing What Your Programs Are Doing", AI Memo 656, Massachusetts Institute of Technology, Feb., 1982.
[21] B. Myers, "Displaying Data Structures for Interactive Debugging", Technical Report, Xerox PARC, Jun., 1982.
[22] T. Winograd, "Breaking the Complexity Barrier (again)", Proceedings of the ACM SIGPLAN-SIGIR Interface Meeting on Programming Languages-Information Retrieval, Association for Computing Machinery, Gaithersburg, MD, Nov., 1973, pp. 13〜30.
[23] G. Fischer, B. Heinz-Deiter, "The Nature of Design Processes and How Computer Systems Can Support Them", Proceedings of the European Conference on Integrated Interactive Computing Systems, ECICS 82, P. Degano, E. Sandewall, ed., North-Holland, Stresa, Italy, Sept., 1983, pp. 73〜86.
[24] H. Wertz, "An Integrated, Interactive and Incremental Programming Environment for the Development of Complex Systems", Proceedings of the European Conference on Integrated Interactive Computing Systems, ECICS 82, P. Degano, E. Sandewall, ed., North-Holland, Stresa, Italy, Sept., 1982, pp. 235〜250.
[25] H. Lieberman, "Designing Interactive Systems from the User's Viewpoint", Proceedings of the European Conference on Integrated Interactive Computing Systems, ECICS 82, P. Degano, E. Sandewall, ed. North-Holland, Stresa, Italy, Sept., 1982, pp. 45〜59.
[26] Z. Manna, R. Waldinger, Studies in Automatic Programming Logic, North-Holland, New York, NY, Artificial Intelligence Series, Vol. 4, 1977.
[27] E. Sandewall, "Programming in an Interactive Environment: the LISP Experience", *ACM Computing Surveys*, Vol. 10, No. 1, Mar., 1978, pp. 35〜71.

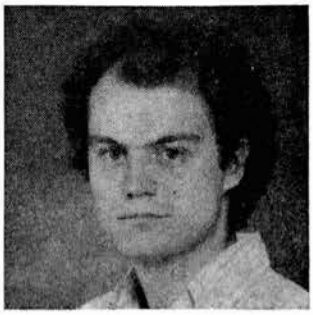
執筆者紹介 Michael S. Pontecorvo

Sperry (現在 Unisys) Communications Corporate Technology Center の Expert Communication Group に所属するシステム・プログラマ．知識ベース・システム・アーキテクチャの設計，知識獲得と機械学習の研究に従事．1984年 Utah 大学で計算機科学の B. S. を取得．既存の手法と知識ベースの手法を併用した，作曲や視覚芸術を支援するソフトウェア・システムの設計の研究に12年間取り組んでいる．



Jed Krohnfeld

1979年に心理学の B. S., 1981年に計算機科学の B. S. を共に Utah 大学で取得. 現在は計算機科学の M. S. コースに在学中. Utah 大学の AI Support System Group のメンバでソフトウェア開発と VLSI 設計の知識ベース・ツールの開発が主な仕事. Salt Lake の Sperry (現在 Unisys) Communications Corporate Technology Center のコンサルタント. AAAI, Cognitive Science Society, ACM のメンバである.



論文 BIBLIO——再使用可能コード・ライブラリ・システム

BIBLIO——A Reusable Code Library System

K. L. Bruso

要 約 一般に，ソフトウェアの開発では，まずコードの持つべき機能を定義，設計し，次に何もないところからコードを作り始める．ところが開発しようとしているコードが，実はテスト済みで正しく動くことが証明されている別のプロダクトの中に，すでに存在していることが多い．このことから明らかなように，より効果的にコードを共有する何らかの方法が開発される必要がある．

コードの共有に関する従来の傾向は，エグゼクティブ・リクエスト (ER 命令) やシステム・ライブラリ (SYSLIB) といった，比較的機械に近い低レベルのルーチンについてコードとドキュメントの共通化を進め，アクセスする手段を提供するというものであった．今後は，この傾向を単一コマンドのレベルから高級言語によるコード・モジュールまで広げ，ソフトウェア・システム構築向きの，より大きく強力なコード・ブロックを提供していかなければならない．

本稿では，複数の開発部門間で共有可能な高級言語のコード・モジュール・ライブラリを開発する技法，およびこれらのコード・モジュールをアクセスするためのシステム (PRIMUS* をベースとする) について報告する．

Abstract In our software development process we define and design code functions and then start coding each function from scratch. In many cases, the code we are creating already exists in another product where it has been tested and proven to work correctly. Clearly, a more effective method of sharing code must be developed.

The trend in software has always been to provide access to code and documentation for common low level routines——Executive Requests and SYSLIB routines for example. We must now continue this trend from single commands into high level language code modules to provide larger, more powerful building blocks for software systems.

This presentation describes a methodology to develop a library of these high level language code modules that can be shared among development organizations, and a PRIMUS-based system which provides access to them.

1. はじめに

長年ソフトウェア産業に従事している人ならば，一つの恐るべき問題が，年を追うごとに大きくなっていくことに気づくはずである．すなわち，コンピュータ・アプリケーションに対する要求がかつてないほど成長し，それを満たすソフトウェアのニーズが，プログラマの生産力をはるかに越えているという事実である．

プログラムの新規開発に対する飽くことを知らない要求に加えて，既存のソフトウェア・パッケージを維持するための巨大な保守作業が積み残されている．ソフトウェア業界全体では，320万人のプログラマが存在し，1983年だけでも65億ラインのコードを生産したとの報告がある[1]．もしソフトウェア産業がこの勢いで成長し続けるならば，1990年には760万人のプログラマが，年間153億ラインのコードを作ることになろう[1]．現在までの成長の傾向がこのようなものであるとすると，これらのプログラマは現在一体何をし，

* PRIMUS は Unisys 社内用ソフトウェアで，問題報告書や修正コードの蓄積・検索・管理用に作られたプログラム開発環境であり，データベース部分とそれをアクセスする使用者インタフェースからなる．

また開発したコードはどう使われているのか，ということについて疑問がわいてくる．

調査によれば，1983年に作られたコードのうち，個々のアプリケーション専用に作られたコードは全体の15パーセント以下にすぎない[1]．残りの85パーセントは共通なコードであり，使用者の入力を調べたり，報告書を作成したり，2進数をASCIIに変換したりするものであった．そして，この共通なコードが，ソフトウェアの開発プロジェクト間で再利用される可能性を持っている．

実際にいくつかのUnisys社のソフトウェア開発プロジェクトを調べてみると，30〜40パーセントのコードが共通であり，単に使用者インタフェースを提供するものや使用者の入力を検証するためのものであった．また，不必要なコードを支援する費用もばかにならない．実際，プログラマのうち，約60〜80パーセントが，既存のコードの保守に携わっていると言われている[2]．

開発待ちのコーディング・プロジェクトの数を減らし，重複するコードの保守に費す時間を減らそうと考えるならば，ソフトウェア・システムを作る時に，より大きなコード・ブロックを使用することを学ばねばならない．

このために，さまざまなアプローチが提唱されてきた．超高級言語，アプリケーション・ジェネレータ，非手続言語による仕様定義に基づいたコード・ジェネレータ，さらにはプログラム・ライブラリなどがその例である．ここでは，プログラム・ライブラリについて議論する．その理由は，最もわかりやすいアプローチであり，初期費用が比較的低廉かつ，ただちに役立つからである．

2. 概　　要

社内のある研究チームは過去1年，再利用コードの可能性について調査をしてきた．コード・ライブラリという課題に対するわれわれの感触は，T. DeMarcoの言葉“再利用可能コードは，今後10年間における生産性向上策のホープである．他の技術については，その間実用化されないであろう[3]”．で良く言い表されている．

Unisys社でも，再利用可能コード・ライブラリの開発が，過去何回か試みられてきた．これらのシステムの失敗の理由は次章に述べられるが，ここで提案したい解決策は，業務として再利用可能コード・ライブラリ・システムを開発する予算を持つ正式な組織を発足させ，ツールとリソースを供給することである．

3. 再使用コード・ライブラリに対して消極的な理由

いくつかのコード・ライブラリが過去Unisys社で試みられ，ある程度の成功を収めている．成功の一例はSYSLIBである．SYSLIBの保守は予算化されており，高い使用実績を有しているために成功した．過去には，ツール・ボックス・マニュアルという名のドキュメントを用意し，コード・ライブラリの統一を図るような試みもあったが，高級言語向きのこの種のコードの収集は，いまだかってSYSLIB並みの成功のレベルまで行きついていない．

コードの収集が失敗する理由は，次の4種に大別される．

1) この種のコードは，表面上正しく動く限り，プロダクトとしてサポートされにくい．すなわち，そのコードの作成者や保守担当者に対し，SUR（問題報告書）を提出し不具合個所を修正してもらうことができない．また，コードの収集が，必要な時に，その場限りのやり方で行われてきた．その上，自分のコードの中でこれらのコー

ドを使用すると，ただちにそのコードの保守の責任を負わされることになりかねない心配がある．

2) コードは，プロジェクトや開発部門が異なると簡単に利用できない．同じスーパーバイザやマネージャの下で働く人々は，そのプロジェクトの中では，コードを再利用すべく互いに働きかけるが，いったんこの限られた環境の外に出ると，共有できるはずのコードに関する知識を持ち合わせていないのが現実である．はなはだしい場合には，自分のコードを秘密にしたり，他の人に使われることを好まないプログラマがいる[4]．

3) “ここで作られたものでないという症候群”．われわれプログラマは，自分が作ったコードを後生大事にする傾向を持つ．これは，コードを作るということが創造的な仕事だからである．また，プログラマは他人のコードを信用しないし，また自分の好み通りに作られていないがゆえに好まない，という傾向を持つ．

4) 再利用可能コードを用いることは，あまり気の進むことではない．再利用可能コードをリロケータブルの形で利用することは，自滅的なことであるかのように思われる．なぜなら，プログラマの生産性の標準的尺度であるところの“作成コード・ライン数”を，自ら減らすことになるからである．

4. 提　　案

われわれは，BIBLIO と呼ばれる再利用可能コード・ライブラリを提案する．BIBLIO は，専門スタッフと予算を持ったプロジェクトとなるはずである．ライブラリの内容は継続的に改良され，開発された部所だけでなく全部門のプログラマの利用に供される．

再利用可能コード・ライブラリを効果的に管理するために，コード・ライブラリアン（ライブラリ管理者）が置かれる．ライブラリ管理者となるには，少くとも6年以上のプログラミングの経験が必要であろう．ライブラリ管理者の仕事は，ライブラリに関する諸手続，コードの受け入れ準備，コード検索のためのツール，流布や保守の方法，それにプログラマに対する教育システム等を確立することになろう．

5. コード受け入れ基準

ライブラリ管理者は，このシステムに含まれているすべてのコードについて最高の権威者であり，責任者でなければならない．ライブラリに受け入れるための基準として，以下のものが考慮される必要がある．

1) ドキュメンテーション……保守が容易にできるよう内部ドキュメントが存在するか，そのコードをどうやって使用するかを記述した外部ドキュメントが存在するか．

2) 復元性……そのコードが，エラー状態や，使用者の誤った使い方に対し，回復可能となっているか．

3) 再利用の能力……これについては，ライブラリの内容とコードが満たすべきニーズのユニークさ等，いく分主観的な決定を行う．たとえば，モジュールの実行時の大きさや実行速度が検討される．

ライブラリ管理者は，コードを他のプログラムに組み込みやすくするために，修正を加えることもある．

①入出力
- *SDF および PCIOS ファイルのリード/ライト，端末，コンソールへのアクセスを行う高級言語によるルーチン
- *ファイル・ハンドリング（ファイル名チェック，ファイルのロール・イン，ER CSF$ エラー解析）
- *オープン，クローズ，ゲット，プット，リード/ライト・ルーチン
- *デバック用トレース・ルーチン

②言語間のインタフェース
- *PLUS ↔ MASM，PLUS ↔ FORTRAN，PLUS ↔ COBOL，およびその逆

③エラー処理
- *コンテンジェンシ・ハンドラ
- *MSCON エラー・ハンドラで，プロジェクト固有なエラー・メッセージを印書する
- *ローカル言語メッセージ・システム・インタフェース

④ デバッギング・エイド
- *PLUS スタック拡張ルーチン
- *PLUS スタック・ジャンプ・トレース
- *SNAP$（書式付きのダンプ・ルーチン）
- *プロシージャ呼び出しのトレース
- *書式付きのメモリ・ダンプ

⑤メモリ・マネジメント
- *メモリ・ページング・ルーチン
- *ダイナミック・メモリ・ハンドリング（ER MCORE$，ER LCORE$）

⑥インライン（INLINES）
- *SYSLIB ルーチン，および以下の ER 命令用の PLUS インライン・コード：READ$，AREAD$，PRINT$，APRINT$，PRNTA$，CSF$，FITEM$，SYMB$，COM$，TREAD$

⑦文字列の操作
- *小文字を大文字にする
- *文字列の長さを調べる
- *ASCII から 2 進数への変換，およびその逆
- *ASCII から FIELDATA への変換，およびその逆

⑧ユーザ/システム情報の検索
- *バッチ，デマンド，デッドライン・バッチのどのモードで動いているか
- *ユーザ識別名，ラン識別名，生成ラン識別名，アカウント番号，プロジェクト識別名，修飾子，条件ワード等
- *PCT 情報
- *MFD 情報

⑨データ・ロック機構
- *Test and set キューイング
- *Read and lock

⑩言語スキャナ

⑪数学/統計

⑫多重アクティビティ

⑬プログラム・セグメンテーション

⑭パフォーマンス解析
- *FLAP インタフェース・ルーチン
- *FLOP インタフェース・ルーチン

⑮エラー・リカバリ

⑯通信
- *端末タイプを識別するためのテスト
- *リアルタイム・アプリケーション

⑰DPS 画面処理
- *コマンド行とエラー行の標準形式

図 1 再利用可能コードの対象分野

Fig. 1 Targeted areas for reusable code

6. 再利用コード・ライブラリの内容

多くの人々は，再利用可能コード・ライブラリが単に既存プログラムから一部のコードを抜き取り，寄せ集めるだけでできるものと思っている．しかし，この方法は通常成功しない．それは，新しい自動車をがらくた置場で見つけた部品で組み立てるようなものである[6]．個々の部品の品質が優れていても，それらが相互に正しく機能するためには，中間的なアダプタが必要となる．ライブラリ管理者は，ライブラリ中のすべてのコードに対し，標準のインタフェースを保証する責任を負うことになる．

コードの再利用の可能性についての研究者は，再利用可能コード・ライブラリに700個程度のモジュールがあれば，ある一つのプログラム中の共通エレメントの約60パーセントが供給できると見積っている[5]．この見積りは，調査が行われる環境によって変わりうる．かりに100Kライン，あるいはそれ以上のコードを開発するプロジェクトをみた場合，かなりの線まで，おそらく51K ラインくらいは再利用可能コード・モジュールでまかなえると考えられる（60%×85%×100Kライン）．

ライブラリの内容は，次の三つのカテゴリに分けられる．

1) 黄金コード……高い信頼性を持つリロケータブル・ルーチンで，単にプログラム中にコレクションするだけですぐ使用できる．
2) コード・テンプレート……黄金コード同様に，正しく動くことが保証されているコード・モジュールであるが，他のプログラムに直接組み込むことができないもの．テンプレート（型板）という名で示されるとおり，他のコード・モジュールを作る際のガイドとなるべきコード・モジュールで，自分のプログラムに組み込む前に手直しが必要である．
3) 再使用可能設計……非手続言語で記述されており，実コードに変換しうるコード・アーキテクチャやアルゴリズムを記述したもの．高水準の設計用言語により，あるアルゴリズム，たとえばある特定のアプリケーション向きの分類アルゴリズムなどを作ることができる．作られたアルゴリズムを，FORTRAN や COBOL などの実コードに変換するツールが用意されることになる．

Unisys 社のある研究チームは，いろいろな言語，たとえば 1100 OS 開発用高級言語 PLUS，メタ・アセンブラ MASM，会話型画面定義システム DPS，汎用構文解析ソフトウェア GSA，会話型処理支援システム CTS，シンボリック・ストリーム・ジェネレータ SSG 等について，再利用可能コード・ライブラリを作るべく調査を行った．ソフトウェアの開発部門が主管する主なプロダクトは，これら言語をフルに使用して作られているからである．

図1は，再利用可能コードを収集する際に，とくに対象として定めた領域を列挙するものである．この調査により，約200のルーチンが選ばれライブラリに収録されることになった．このほか，他のルーチンをライブラリに組み込みたいという提案は，定常的に受け付けられている．

7. ツ　ー　ル

いろいろなツールを調査した結果，ライブラリ・システムを扱うのに適した PRIMUS が選ばれた．PRIMUS は，キーワード検索機能，操作の簡単な挿入・更新機能，理解しやすい使用者インタフェース，可変長レコード形式等，多くの利点を持っている．われわれは，PRIMUS へのインタフェースと全画面メニューを持つソフトウェアを開発し，こ

れを BIBLIO と名付けた．BIBLIO は，@ BIBLIO と入力することにより簡単に呼び出せる．呼び出しによって，図2のメニューが表示される．

図2のメニューで示されている四つの項目の意味は，次のとおりである．

① 選択肢1は，PRIMUS コード・ライブラリ・アプリケーションに入る（後述）．

② 選択肢2は，ライブラリ・システムの使用方法についてガイドを提供する．

③ 選択肢3は，コード・ライブラリ・ドキュメント（図4，5参照）で使用されているすべてのキーワードを表示する．キーワード・リストは，ドキュメントで使用されているキーワードをもれなく反映するよう，常に更新されている．

④ 選択肢4は，システム上にあるユティリティのインデックスを表示する．コード・ライブラリと似ているが，コード・モジュールというよりユティリティと言うべきルーチンを説明する．

ここでは，例として1を選択し，PRIMUS コード・ライブラリ・アプリケーションに入る．すると，画面上に図3が表示され，候補となるべきコード・モジュールをライブラ

```
Sperry Code Library System
    1) Code library interface
    2) Tutorial
    3) Keyword list
    4) Utilities interface
choice?>1 ⤸
```

図 2 BIBLIO へのメニュー・インタフェース

Fig. 2 Menu interface to BIBLIO

```
PRIMUS 3R2 SL74R1 05/10/85 09:48:02
>Document type CODELIB ⤸
  Ready
>Select keyword=inline ⤸
  56 CODELIBs selected
>Report ⤸
```

図 3 PRIMUS へのインタフェース

Fig. 3 PRIMUS interface

```
05/10/85                    Sperry Code Library                    09:46:23
Document Number: 0123                          Date Inserted: 850412
                                               Date Updated:
Title: ER PRINT Inline
Language: PLUS
Description:
    This INLINE interfaces with the EXEC requests PRINT$ and PRNTA$ depending
    on the radix of the parameters passed to it.

    Calling Sequence:
    INLINE er_print IMPORTED ('er print');
    This inline requires at least one parameter and may accept up to three parameters.

    Parameter 1 is the character string to be printed.
                The declared radix of this parameter will determine if PRINT$ or PRNTA$
                is issued.

    Parameter 2 is the spacing count Default: 1

    Parameter 3 is the number of characters to print.
                Default: length of parameter 1.
Enter View Directive (TRANSMIT-GO+1)>0 )
```

図 4　コード・ライブラリドキュメント

Fig. 4　Sample code library document

```
    Examples:
        er_print('This is a literal');
        er_print(name_of_message,2)
        er_print('la de da,'1,2);

    Collection Considerations:

    Restrictions:

    Enhancements:

    Elements: er-print

    Keywords: PRINT$  PRNTA$        ER
              INLINE  EXEC-REQUEST

    Source: M. Rahmani

    Unit Test:

    Enter View Directive (TRANSMIT-GO+1)>exit )
```

図 5　コード・ライブラリ・ドキュメント（続）

Fig. 5　Sample code library document (continued)

リ・システムから選び出す準備ができる．

ここでは，“inline” というキーワードを持つすべてのドキュメントを探してみる (Select Keyword＝inline を入力)．

PRIMUS は，キーワード “inline” を持つドキュメントが，56 個存在していることを表示する．そこで，Report コマンドを用い，これらの内容を調べることにする．

図 4 と 5 は，Report コマンドにより画面上に表示されたドキュメントの例である．画面上の各項目の目的と意味は，次のとおりである．

① Document Number……ドキュメントが入力されたとき，PRIMUS により自動的に割り振られる一連番号．

② Date Inserted……ドキュメントが作られた日．

③ Date Updated……最後にドキュメントが更新された日．

④ Title……1 行で表した標題．たくさんのドキュメントから素早く最適ドキュメントを探すのに役立つ．

⑤ Language……コードが書かれている言語．PLUS，FORTRAN，SSG，CTS 他．

⑥ Description……コードの目的（何をするか），その目的をどう実現するかを詳しく説明する．

⑦ Calling Sequence……該コード・モジュールをどのように呼び出し，また，その時にどのような条件があるかを説明する．呼び出し方法がいくつかある場合には，それぞれについて説明する．呼び出し方法の具体例も示される．

⑧ Collection Considerations……アブソルート・プログラムに正しくコレクションするための条件について説明する．たとえば，バンクの性質（動的・静的等）やDMS ルーチン，あるいは他のライブラリ・モジュールと同じバンクに置く必要がある，などの注意事項を指示する．

⑨ Restrictions……機能について，既知の制限があれば記述する．

⑩ Enhancements……コードは継続的に保守されているので，必然的に機能拡張が加えられることがある．その場合に説明を加える．

⑪ Elements……そのコード・モジュールを構成しているエレメントの名前．コード・モジュールは，inline のように一つのエレメントでできていることもあり，また複数のエレメントとプロシージャからできていることもある．

⑫ Keywords……ライブラリから特定のコードを検索する時に用いられる．キーワードは，ハイフン（-）を使い，たとえば EXEC-REQUEST のように，“キー・フレーズ”として使用することもできる．@ BIBLIO で表示されるメニューにおいて，KEYWORD（選択肢 3）を選ぶと，ライブラリ・システムで使われているすべてのキーワードが表示される．

⑬ Source……出典（作者）

⑭ Unit Test……潜在的ユーザに対し，そのコードが記述されたとおり動くことを保証するために，単体テストの方法とその結果について記述する．また，そのコードを使用しているプロジェクト名も記述する．

8. リリース，流布および保守

コード・ライブラリは，他のプロダクトと同様に保守されるため，使用者は，問題報告書や改造要求書を提出することができる．コードは，それを使用しているプロジェクトではなく，ライブラリ管理者の責任下に置かれる．SYSLIB 同様，定期的に社内にリリースされ，前レベルとの互換性が保たれるよう努力が払われる．

9. プログラマ教育

既存のコードの収集が広く普及しなかったのは，その効用について一般の知識が不足していたためである．ライブラリ管理者は，ライブラリ・システムを効果的に使用できるよう，プログラマ教育に全力を注ぐ必要がある．同時に，ライブラリに加えうる再利用可能コードを作るにはどうしたらよいか，あらゆる機会と手段をとらえて教育を実施する必要がある．たとえば，昼食をはさんで行われるセッション，終業後の教育コース，ビデオ教材を利用したコース，新規採用者に対する教育コース，プログラマに対するコミュニケーション用出版物等を通じて行うこともできよう．

10. 利　　益

再利用可能コードの適用により，数多くの利益が生ずる．第1に，各プロジェクトは自分専用の少数のコードのみを作り保守すればよいことになる．このことは，また全体コストを削減しながら，高品質のコードが短期間に作れるという利益につながる．第2に，より強力なソフトウェア・モジュールを使うことにより，ソフトウェア・システムのプロトタイピングがより現実化する．

1984年に Unisys 社のある部門の30のシステム開発関連のプロジェクトは，一つのプロジェクト当たり26000ラインの新規コードを開発している．もし，このコードの40パーセントが，ライブラリに用意された再利用可能コード・モジュールで作ることができたとしたならば，そのライン数は10400ラインになる．

プログラマが1日に15ライン作るとすれば，10400ラインはおよそ2.7人年分の作業量である．プログラマのコストを，かりに年間75,000ドルとすれば，このプロジェクトは，1年で2.7×75,000ドル，すなわち202,500ドル節約できたことになる．一つのプロジェクトでこれだけ節約できるのであるから，会社全体では莫大なコスト削減になるはずである．また，保守費用の削減も見込まれる．仮りに，30のプロジェクトで，コード・ライブラリを利用したとすれば，コーディングに費すコストだけでも，202,500×30＝610万ドルの節約となる．ライブラリを充実したり，保守するにはコストがかかる．このために3名のプログラマが必要だと仮定し，そのコストを差し引いても，費用の節約は500万ドル以上になる（なお，この3名の内訳は，コードの収集・開発の要員1名，コードのテスト・保守の要員1名，ツールの保守と使用者教育の要員1名となっている）．

11. お　わ　り　に

専門プログラマは，①高品質ソフトウェアをすぐにでも欲しいというユーザからの要求，②高品質を保ちつつ複雑なコードを大量に作るには能力の限界がある，という相反する二つの事実の間に立たされている．われわれは，未開発資源，すなわち再利用可能コードを実用に移すことで，品質，投資効果，それに納期の面で大きな利益を得ることができるはずである．再利用可能コード・ライブラリ・システムを成功に導くには，プログラム要員をライブラリの開発・収集・保守，さらにはそれを効果的に利用するための教育に専従させねばならない．ライブラリ・システムが機能すれば，プログラミングの生産性およびコストの面で大きな前進が得られるはずである．

（ソフトウェア品質管理部　品田　敏孝 訳）

参考文献 [1] T. Jones, "Reusability in Programming: A Survey of the State of the Art" *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, Sept. 1984, p. 488.

[2] Y. Matsumoto, "Some Experiences in Promoting Reusable Software: Presentation in Higher Abstract Levels", *IEEE Transactions on Software Engineering* Vol. SE-10, No. 3 Sept. 1984, p. 502.

[3] T. DeMarco, T. Lister, at the class "Controlling Software Projects", Technology Transfer Institute, Apr., 1-5, 1985.

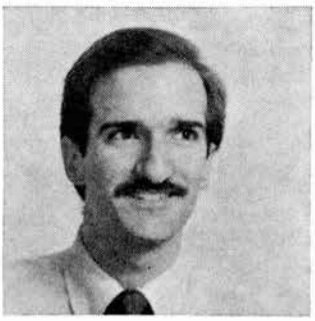
[4] L. Bernstein, C. Yuhas, "Blood From Turnips?", *DATAMATION*. Jan., 15, 1985, pp. 108～112.

[5] T. Jones, "Laying the Groundwork With Reusable Code", *Computer World*, Vol. 18, No. 26 A, Jun., 27, 1984, pp. 12～16.

執筆者紹介 Kelsey L. Bruso

1981年に Iowa 州の Luther College を卒業，音楽とコンピュータ科学の分野で学士号を持つ．

Sperry 社（現在 Unisys 社）でシステム・プログラマとして4年間の経験を有する．当初，要求定義用プロセッサ RDP の開発および保守を担当．現在はプログラム・モジュール・ライブラリの研究開発を担当すると共に，システム生成・管理用プログラム COMUS の機能拡張に携わっている．



論文 カナ漢字変換システム Micro JASTY の開発

Development of Kana-to-kangi Transformation System Micro JASTY

吉田正行

要約 日本ユニバックのデスク・ステーション DS7 のカナ漢字変換システム Micro JASTY を開発したので報告する．本稿では，Micro JASTY の基となっているシリーズ 1100 のカナ漢字変換システム JASTY の機能概要，シリーズ 1100 から DS7 への移植，Micro JASTY 使用方法と処理概要について記述する．

Abstract This paper reports the implementation of Micro JASTY which is a Kana-to-kanji transformation system specifically for UNIVAC desk station DS7.

First of all, JASTY, Kana-to-kanji transformation system for series 1100, from which Micro JASTY is derived, is described. Then the paper discusses the migration of JASTY from 1100 to DS7. Finally, its uses and processing features are explained.

1. はじめに

シリーズ 1100 上でカナ漢字変換システム JASTY[4] を開発した頃は，汎用機では“べた書き入力”が行えても，パーソナル・コンピュータやワープロ上では文節変換が普通であった．その後，メモリやディスケットなどのハードウェアの発達により，パーソナル・コンピュータでも“べた書き入力”による連文節変換が普及してきた．カナ漢字変換はワープロの入力として使われるだけでなく，自然言語処理や機械翻訳の入口に当たり重要な技術である．また，現在機種により異なったカナ漢字変換が使用されているが，日本ユニバックとして統一した機能と操作環境を提供したい．そこで，JASTY と同じ機能を持つカナ漢字変換システムを当社のワーク・ステーション DS7 上で，Micro JASTY（以降 DS7 上のカナ漢字変換システムを Micro JASTY と称する）として開発することになった．この開発プロジェクトは，61年12月に完了しパフォーマンスおよび変換率とも当初の目標を達成できたので，ここに Micro JASTY の開発作業と機能の概要を報告する．

2. JASTY

2.1 概　　要

Micro JASTY の開発のベースとなった JASTY は，二つのプログラムから構成されており，一つは端末装置の画面を介して使用者が入力したカタカナ文字列，またはローマ字列を会話型でカナ漢字変換し，漢字カナ交じり文のシンボリック・エレメントを作成するプログラム (JASTY) である．他の一つは，カナ漢字変換機能の必要な使用者プログラムから呼ばれる各種命令を処理するルーチンからなるライブラリ (JASLIB) である．JASTY と JASLIB の構成をそれぞれ図1および図2に示す．

2.2 変換方式

変換方式は吉田・日高(九州大学)，吉村ら[1],[2],[3](福岡大学) が提唱する“べた書き入力”の自動分かち書きアルゴリズム「文節数最小法」を基本としており，この他に表1に

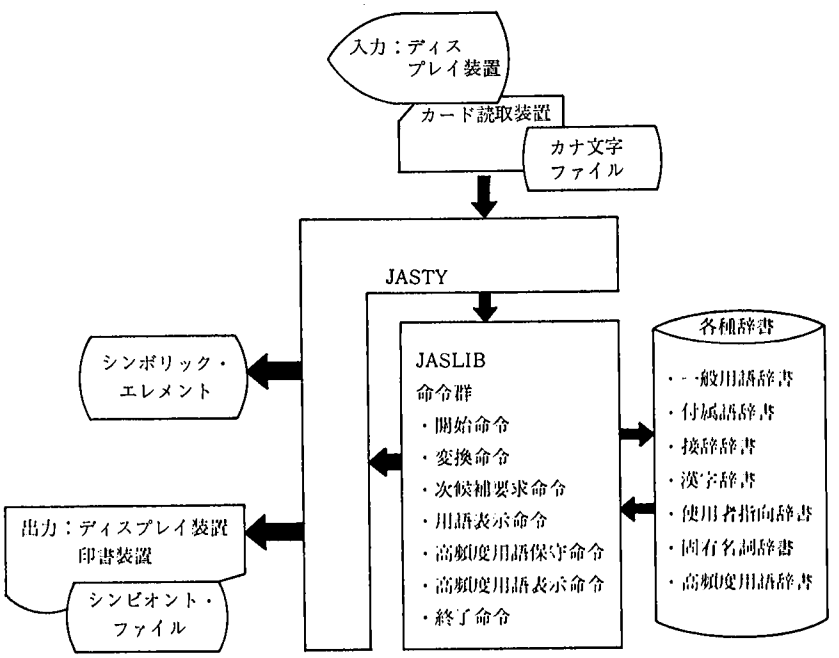


図 1 JASTY の構成

Fig. 1 Structure of JASTY

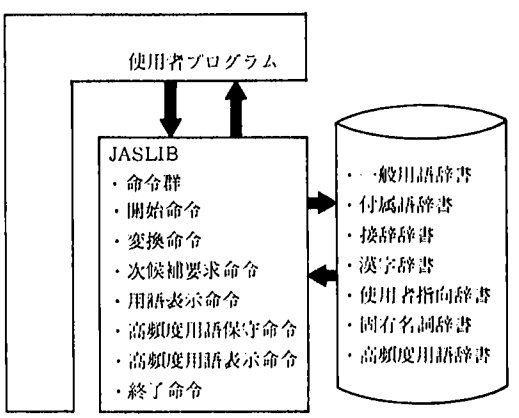


図 2 JASLIB の構成

Fig. 2 Structure of JASLIB

示す 16 種の各種変換指定入力ができる.

文節数最小法とは,“べた書き入力”された文字列を文節構造規則（単語の連鎖として文節を規定する規則）を用いて単語・文節の認定を行うと，認定された結果の中には意味的に継がりのない単語のら列も含まれているが，一般的に文節数が小さい解析結果に正しいものが多いという経験則に基づく手法である.

2.3 辞　　書

JASTY では，カナ漢字変換用辞書として表 2 に示す七つの辞書を持っている.

一般用語辞書は，カナ漢字変換システムにとって中心となる辞書であり使用頻度も高い. また，用語数も 86,420 語と膨大であるためディスク上に常駐しており，必要な都度

表 1 変換指定一覧 (JASTY)

Table 1 Summary of commands for transformation

名称	形式	機能
無変換指定	A／文字列／	英字，数字，特殊記号を漢字コードへ変換する．
ひらがな指定	H／文字列／	指定された文字列をひらがなへ変換する．
カタカナ指定	K／文字列／	指定された文字列をカタカナへ変換する．
用語指定	W／見出し語[. n]／	指定された見出し語を一般用語辞書中の対応する漢字へ変換する．
使用者追加用語指定	U／見出し語／	指定された見出し語を使用者指向辞書，および高頻度用語辞書中の対応する漢字へ変換する．
姓指定	S／姓見出し語[. n]／	指定された見出し語を固有名詞辞書中の対応する漢字へ変換する．
名指定	M／名見出し語[. n]／	指定された見出し語を固有名詞辞書中の対応する漢字へ変換する．
人名指定	G／姓[. n] 名[. n]／	指定された見出し語を固有名詞辞書中の対応する漢字へ変換する．
企業名指定	F／企業見出し語[. n]／	指定された見出し語を固有名詞辞書中の対応する漢字へ変換する．
地名指定	T／地名見出し語[. n]／	指定された見出し語を固有名詞辞書中の対応する漢字へ変換する．
音訓読み指定	Y／読み1，読み2／ Y／読み[. n]	指定された読みを漢字辞書中の対応する漢字へ変換する．
JIS 8単位符号指定	Q／文字列／	指定された文字列を8単位符号のまま残す．
区点番号指定	J／区点番号……／	指定された区点番号を対応する漢字へ変換する．
拡張区点番号指定	E／区点番号……／	指定された区点番号を対応する漢字へ変換する．
16進数表現指定	X／16進数……／	指定された16進数を対応する漢字へ変換する．
略号指定	R／略号／	指定された略号を使用者指向辞書，および高頻度用語辞書中の対応する漢字へ変換する．

表 2 辞書の種類

Table 2 The dictionaries for transformation from Kana to Kanji

辞書名	容量	収容語彙数	備考
一般用語辞書	769 TRK	86,420 語	
付属語辞書	2 TRK	311 語 4,975 個	付属語 接続テーブル
高頻度用語辞書	4 TRK		JASLIB の作業領域の大きさによって最小 100 語から最大 500 語まで収容できる
接辞辞書	4 TRK	514 語	
漢字辞書	35 TRK	6,349 字 3,734 個	JIS 第1／第2水準 読み数（音，訓）
固有名詞辞書	187 TRK	20,045 語	姓，名，企業名，地名
使用者指向辞書			使用者により異なる．

メモリへ読み込んで使用する．したがって，一般用語辞書の記憶方式と検索方式の最適化が重要である．このため，最少の検索時間で必要とするすべての用語を抽出できるように階層化した2進木のデータ構造とした．

付属語辞書は助詞・助動詞を持つ付属語テーブル，自立語と付属語，および付属語と付属語の接続可否情報を持つ接続テーブルから構成されており，一般用語辞書同様，カナ漢

字変換システムにとって中心となる辞書である．この辞書は用語数が少く小容量であり，カナ漢字変換処理中は主記憶装置上に常駐している．

高頻度用語辞書は，一般用語辞書の検索と同時に検索される辞書である．また，カナ漢字変換実行中にこの辞書の用語の登録や削除も行われるため，七つの辞書の中で最も使用頻度は高い．したがって，本辞書はカナ漢字変換の初期処理でディスク上のファイルから主記憶装置上へ読み込まれ，カナ漢字変換実行中は主記憶装置上に常駐している．

接辞辞書，漢字辞書，固有名詞辞書および使用者指向辞書は，カナ漢字変換システムにとって必須の辞書ではなく，特定の処理の時にのみ使用される使用頻度の低い辞書であり，その概要は次のとおり．

1) 接辞辞書：用語指定入力と数詞文節処理用辞書
2) 漢字辞書：音訓読み指定入力処理用辞書
3) 固有名詞辞書：姓，名，企業名，地名，人名指定による入力処理用辞書
4) 使用者指向辞書：使用者追加用語指定入力処理用辞書

これらの辞書は，索引のみを主記憶装置上に常駐させているが，辞書本体は必要の都度ディスク上のファイルから読み込んで使用する．

3. Micro JASTY

3.1 シリーズ 1100 から DS 7 への移植

JASTY から Micro JASTY へ移植を行うに当たって，今までの JASTY を振り返って長所は残し，不十分な点は改善した．

また，DS 7 のハードウェアからくる制限と，DS 7 の OS である C-CP/M からくる制限を考慮し，以下の基本方針を立てた．

1) JASTY の最大の特徴である変換方式（文節数最小法）を継承する．
2) 使いやすさの向上のため，変換指定入力をできるだけ少なくする．
3) C-CP/M の特徴である四つのプロセスのコンカレント処理をサポートする．
4) メモリ・サイズは，スモール・モデル（命令語領域，データ領域の各々が 65 KB 以内）とする．
5) パフォーマンスは，変換キー押下から第 1 候補出力までが 1 秒程度を目指す．

以上の方針で移行作業を開始したが，幸いなことにシリーズ 1100 の開発言語 PLUS (Programming Language for UNIVAC System) と DS 7 の開発言語 C とが類似した言語であったため，シリーズ 1100 の内部仕様書やコーディングが活用できた．このため，文節数最小法の継承は何の問題もなくスムーズに行えた．また，シリーズ 1100 上ではデマンドによる使用を前提としているため，命令語領域はコモンバンクとして稼動する形で作成していたので，C-CP/M 下でのコンカレント・プロセスのサポートはもともと考慮されていた．以上のことより，今回の移植作業はメモリ・サイズ，パフォーマンスおよび使いやすさの向上の 3 点に注意を払って作業を行った．

主な作業項目を以下に述べる．

3.2 改 造 項 目

3.2.1 辞 書 の 統 合

JASTY では，辞書は全体で 7 辞書に分かれ，かつ容量も 1,000 トラックを越す大きなものであった．しかし，Micro JASTY は外部記憶装置として 1 MB（メガバイト）のディスケット装置が 2 台，または 1 MB のディスケット装置 1 台と 10 MB のハード・ディ

スク装置1台とが考えられ，その容量が JASTY と比較して小さい．このため，辞書は三つに統合し，収録用語数も大幅に絞り込んだ．

1) 一般用語辞書……JASTY の一般用語辞書，接辞辞書，漢字辞書，および固有名詞辞書を一つの辞書に統合し，収録用語を約4万3,000用語に絞り込んだ．統合により一つの辞書に入れたこれらの用語は，各々別の品詞コードを割り当てることにより区別している．
2) 付属語辞書……内容的には，JASTY の付属語辞書と同じ辞書である．ただし，外部記憶装置上には持たずにプログラム上に定数として持っている．
3) 高頻度用語辞書……JASTY と同じである．最大収録用語数を増やすため，レコードを可変長とした．

3.2.2 辞書構造の変更

JASTY の一般用語辞書では，シリーズ 1100 上での検索効率を考えて固定のレコード長で階層化した2進木データ構造としていた．2進木構造では，データ内容を可変長で持つように変更しても各ノードを結ぶポインタ領域が必要となるため，辞書ファイルの容量は膨大となる．この方式は，DS7 の外部記憶装置の容量を考えると良い方法とは言えない．

そこで Micro JASTY では容量を減らすため次のような辞書構造に変更した．

1) ブロックごとに索引を持つ．
2) レコードは可変長とし，余分な部分は取り除く．
3) 前のレコードの読みと重複する部分の読みは省略する．

上記3項目の変更を行ったが，この他に Micro JASTY の辞書では他のファイルと一般用語辞書を区別するため，辞書 ID を持たせている．辞書構造を図3に示す．

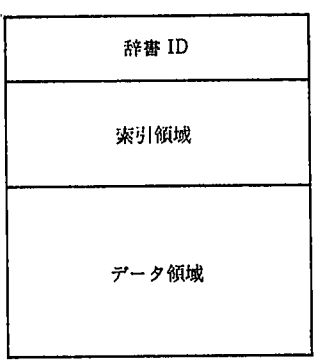


図3 辞 書 構 造

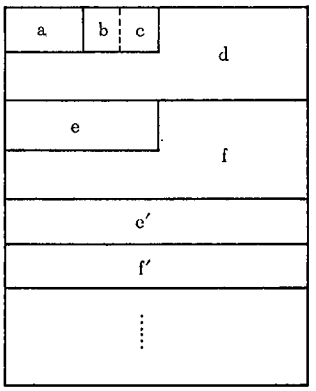
Fig. 3 Structure of independent-words dictionary

索引領域は，読みと“データ領域を結び付けるセクタ番号”の対になったものであり，変換処理中は主記憶装置上に常駐している．

データ領域は辞書容量の 99 パーセントを占有する部分である．1バイトのレコード長の違いによって，4万用語を持つ辞書では 40 KB の違いとなる．このため，データ領域内のレコード形式は1バイトでも短くなるように図4の形式をしている．

3.2.3 辞書の先読み

JASTY でのカナ漢字変換を考えると，カナ文字列の入力は端末側で行われ，辞書の検索はホスト側で行われる．したがって端末側で入力した文字列は，端末の送信キーが押下された時点でホスト側へ一括送信され，辞書の検索へと続く．これに対し，Micro JASTY



a：同音語データ全体の長さ(バイト数)
b：一つ前の用語と読みが異なる部分の長さ
c：一つ前の用語と読みが同じ部分の長さ
d：一つ前の用語と異なる部分の読み
e：品詞(最大3品詞)
f：漢字

図 4 レコード形式
Fig. 4 Layout of record

ではカナ文字列入力部分（キーボード）と辞書検索部分が同一マシン上に存在する．すなわち，カナ文字列を1文字入力するごとに変換ルーチンに先渡しが可能な環境である．この環境を利用して，カナ文字が1文字入力されるたびに必要な辞書検索を行い，DS 7 で最も時間のかかる外部記憶装置との I/O 時間をキーボード入力時間に吸収させることが可能となった．キー入力時間と辞書検索時間を調べてみると，以下のとおりである．

1) キー入力時間[5]

最高のプロ 0.08秒/1キー
平均的プロ 0.20秒/1キー
平均的アマ 0.28秒/1キー

2) 辞書検索時間……DS 7 側で 24 文 692 文字の入力データで辞書検索の測定を行った際の検索時間は，次のとおりであった．

辞書をハード・ディスク上に持った場合

合計処理時間 170秒
1文平均 7秒
1文字平均 0.25秒

辞書を GRAM（グラフィック・メモリ）上に持った場合

合計処理時間 82秒
1文平均 3.4秒
1文字平均 0.12秒

上記より辞書を GRAM 上に持った場合は，平均的プロの入力速度に対応できること．また，辞書をハード・ディスク上に持った場合は，平均的アマの入力速度に対応できることが判明した（図5）．

3.2.4 指定入力の削減

“べた書き入力”が読みのまま自由に入力することを基本にしているのに対し，変換指

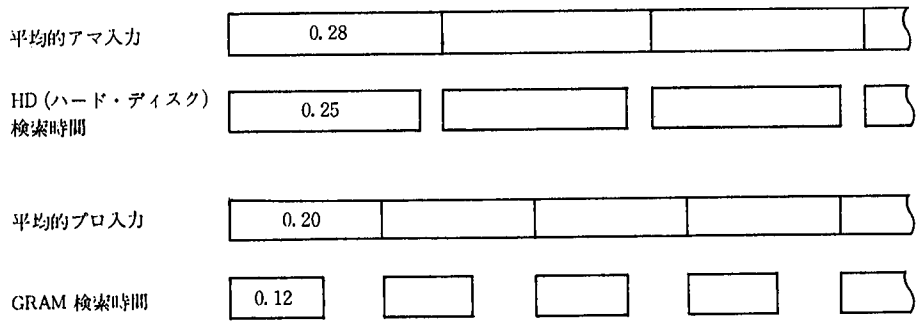


図 5 入力時間と検索時間

Fig. 5 Input and retrieval time

表 3 変換指定一覧 (Micro JASTY)

Table 3 Summary of commands for transformation

名　　称	指示	機　　能
無変換指定	A	英字・数字・特殊記号を漢字コードへ変換する.
ひらがな指定	H	指定された文字列をひらがなへ変換する.
カタカナ指定	K	指定された文字列をカタカナへ変換する.
区点番号指定	J	指定された区点番号を対応する漢字へ変換する.
16進数表現指定	X	指定された16進数を対応する漢字へ変換する.
略号指定	R	指定された略号を高頻度用語辞書中の対応する漢字へ変換する.
郵便番号指定	T	指定された数字を住所に変換する.

定入力は単語ごとに変換種別を指定し，読み以外のキー入力が必要となる．このため，Micro JASTY では不必要な指定入力は大幅に削除し，表 3 に示す 7 種とした．

指定入力が 16 種から 7 種へと大幅に少なくなったため，Micro JASTY は JASTY より，さらに“べた書き入力”が可能となり，入力操作が容易になった．

4. Micro JASTY の概要

4.1 Micro JASTY の使い方

Micro JASTY を使用する場合，次のような使い方が考えられる．

1) フロント・エンド・プロセッサ……キーボードと C-CP/M の間に置かれ，キーボードからの入力を受け取り，カナ漢字変換した結果を C-CP/M へ渡す．アプリケーション・プログラム (AP) にとっては，キーボードから漢字が直接入力されたように見える（図 6）．

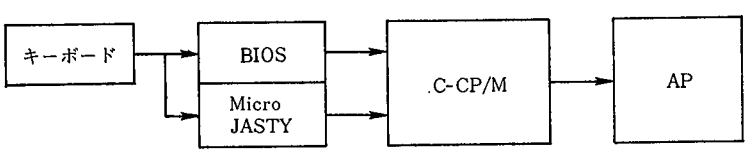


図 6 フロント・エンド・プロセッサとしての使用

Fig. 6 Use of Micro JASTY as front end processor

2) サブルーチン……AP が Micro JASTY をサブルーチンとして呼び出す．この方式

ではキーボードから入力されたカナ文字列はそのまま AP へ渡り，AP と Micro JASTY との間でカナ文字列と漢字カナ交じり文のやり取りを行う（図7）.

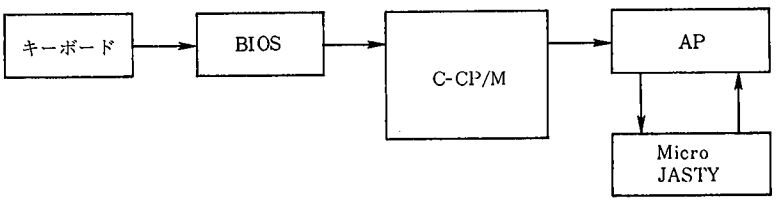


図 7　サブルーチンとしての使用

Fig. 7　Use of Micro JASTY as subroutine

Micro JASTY はコンカレント処理の考慮をしているので，C-CP/M の四つのプロセスから自由に利用が可能である（図8）.

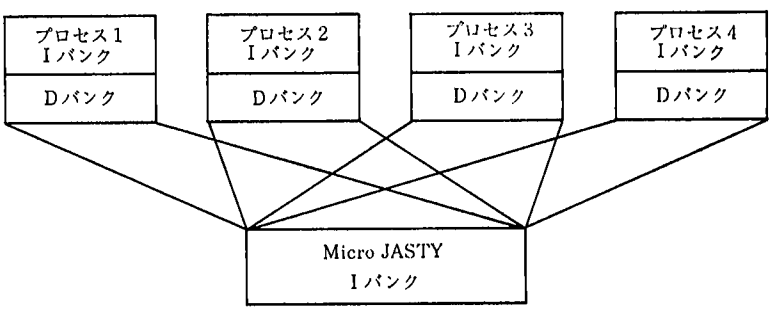
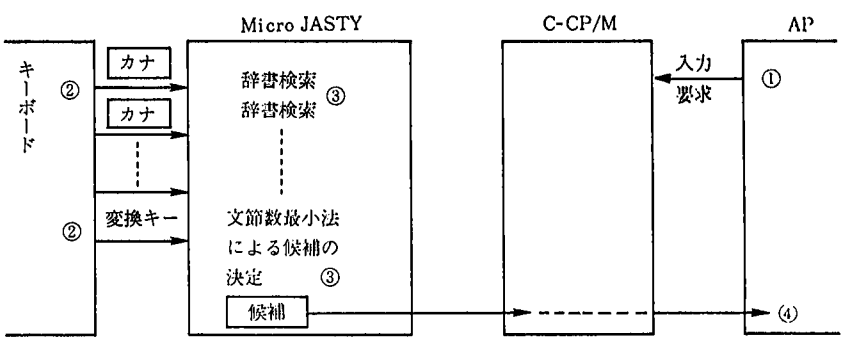


図 8　コンカレント・プロセスでの使用

Fig. 8　Use of Micro JASTY in concurrent processing



① AP より入力要求．この時点よりキーボード入力が可能となる．

② キーボードよりカナまたはローマ字入力．
辞書検索を行い，必要な用語は主記憶装置上に貯える．
変換キーが入力されるまで②を繰り返す．

③ ②で検索した用語の接続検定（先行する自立語もしくは付属語，と後続する付属語とが結び付くか否かを調べる）を行い，文節数最小法を利用して候補を見つける．

④ 候補を C-CP/M を経由して AP へ渡す．

図 9　変換処理の流れ

Fig. 9　The flowchart of transformation process

4.2 処理の流れ

Micro JASTY をフロント・エンド・プロセッサとして使用する場合を例にとり，カナ文字が入力され漢字カナ交じり文に変換され，変換結果が AP に渡るまでの処理を図9に説明する．

5. おわりに

DS7 のカナ漢字変換機能としては，当初われわれが予定していた変換率およびパフォーマンス共に満足するものができた．開発者は長い間シリーズ 1100 を担当しており，パーソナル・コンピュータのソフトウェア開発は始めての経験であった．われわれなりにパーソナル・コンピュータでの使いやすさ等を勉強し，Micro JASTY の開発に取り組んできた．しかし，使用者から見ると不満足な点が発生すると思われる．今後発生した不満足な点は順次改良し，さらに使いやすい Micro JASTY に育て上げたい．

なお，カナ漢字変換方式を設計するに当たって，多大のご指導を頂いた九州大学工学部の吉田教授，日高助教授，九州芸術工科大学の稲永助教授，および福岡大学工学部の吉村助教授に深く感謝の意を表します．

参考文献

［1］ 吉村賢治，日高　達，吉田　将，“表方式を用いた文節構造分析アルゴリズムとその効率について”，情報処理，計算言語学研究会資料，25-6，1980.

［2］ 日高　達，吉田　将，“効率的日本語単語機械辞書”，情報処理学会第 24 回全国大会論文集，5G-7，1982.

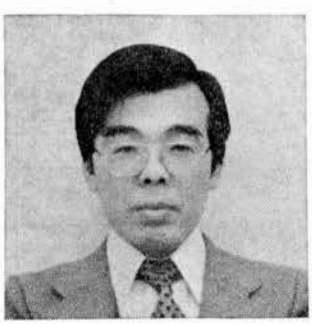
［3］ 吉村賢治，日高　達，吉田　将，“日本語文の形態素解析における最長一致法と文節数最小法について”，自然言語処理，40-7，1982.

［4］ 三ツ矢裕一，吉田正行，小山憲一，“シリーズ 1100 カナ漢字変換システムの開発”，技報，日本ユニバック(株)，No. 7，1984.

［5］ 木村　泉，粕川正充，“日本語ワープロ向け新打鍵レベル模型の検証と応用”，情報処理，日本語文書処理研究会報告，No. 6，1986.

執筆者紹介　吉　田　正　行 (Masayuki Yoshida)

昭和 14 年鹿児島大学理学部卒業，同年日本ユニバック(株)入社．シリーズ 90 でのフィールド・サポート，リアル・タイム・パッケージの開発に従事，56年日本語ソフトウェア開発部に転属，日本語文書処理システムの開発，主に日本語文入力を中心とするプログラム開発を担当．



論文

意味的なつながりを考慮した接尾語辞書の作成

A Suffix Dictionary Utilizing the Meanings of the Stem of Derivatives as the Connection Rule to Proper Suffixes

新谷隆之

要約 カナ漢字変換をはじめとするカナ文字文の機械処理システムにおいては，未登録語の処理は同形異義語の処理と並んで重要な研究テーマである．

どのように辞書の見出し語を増やしても，派生語のすべてをカバーすることは不可能であり，接頭語辞書と接尾語辞書の量的・質的充実を図る以外に未登録派生語の処理方法は考えられない．

日本語においては，文節間のみならず単語を構成する成分の間においても係り受けの存在が認められる．

本稿では，接尾語の中でも圧倒的に多い名詞の接尾語を中心として，受けの成分である接尾語に対して，係りの語をその漢字表記の意味で分類し接続の条件として与えた接尾語辞書の作成手順について述べる．

Abstract Along with the homonym processing, it is crutial to study how to handle unknown words like derivatives in machine processing of Japanese sentences including Kana-Kanji translation. Even massive addition of headwords does not ensure the full coverage of derivatives. Nothing but an improvement of prefix/suffix dictionary both in quality and quantity could solve the problem.

Applying "kakariuke" relation to the intrastructure of derivatives, this paper describes the way to construct a suffix dictionary which utilizes the meanings of the stem of derivatives as the connection rule to proper suffixes.

1. はじめに

近年，仮名漢字変換をはじめとする日本語処理システムは急速な勢いで普及しており，入力方式も単漢字変換から文節変換を経て，今や連文節変換，文章一括変換へと進歩してきた．

当面は，市販のワープロ，パソコンに見られるように，大容量の辞書によらないシステムが発展すると思われるが，辞書の内容が十分でなく文節間や複合語内の論理的意味的つながりを考慮しているとはいえず，とくに，派生語を含んだ文節の変換精度の悪さが目立つ．

将来的にはメモリーをはじめハードウェアの低価格化が進み，数十万の見出し語を有する辞書がこれらのシステムでも搭載可能になると思われ，このためにも本格的な辞書の整備は急務である．

未登録語としての派生語の処理のためには，その派生語を直接自立語辞書に登録するのが最も簡単な解決策であり効果も大きい．しかし派生語はその数が膨大である．しかも基本的な単語としての地位が確立しておらず，単語というよりはむしろ言語表現の一形態と考えられるような語が少なくない．このため派生語をすべて登録しようとするのは無理であり無駄である．

派生語を構成する末尾の語は，その直前の単語（群）の接尾語として機能していることが多く，両者の間には一種の係り受け関係が成立している．

そこで，この係り受け関係を分類整理してあらかじめ辞書に登録しておけば，同形異義の単語群と接尾語群の中から正しい組み合わせを選ぶのに役立ち，派生語の漢字変換の精度の向上を図ることができる．

本稿では，まずここでいう派生語を定義し，未登録語となりやすい派生語のタイプを示した後，派生語を含んだ10万7千余語の自立語辞書から上記のような係り受け関係の存在する派生語名詞を機械的に抽出し，それから目的の接尾語辞書を作成した手順について述べる．また，係りの語の品詞／意味分類別にどのような接尾語が付きやすいかの統計も取ったので併せて報告する．

2. 用語の定義と派生語の構造

国文法用語の定義は解釈の違いから諸説があり，定まっているわけではない．

ここでは，まず本稿で用いる派生語および接尾語の定義を行い，次に派生語の構造について考える．

2.1 用語の定義

国文法では，意味の面からも形態の面からも独立していて言葉の基本単位となるものを**自立語**という．自立語の例としては，名詞，代名詞，動詞，形容詞，副詞などがあげられる．

一方独立して一語を形成せず，造語成分として作用するものを**接辞**といい，いわゆる接頭語・接尾語がこれに相当する．

この接辞と自立語からなる二次語を**派生語**といい，たとえば「ご指導」，「美しさ」をあげることができる．これに対して，自立語（または自立語に準ずるもの，たとえば形容詞の語幹）を組み合わせてできた二次語を**複合語**という．

「社会運動」，「望み薄」，「飛び跳ねる」などがこれに当たる．

本稿では派生語の定義を拡大して，上に述べた複合語のうち，分かち書きされる可能性がなく，また発音する時には，ひとまとめに発音される複合語も派生語という．そして，派生語の末尾の自立語または接辞をその**派生語の接尾語**といい，派生語のうちで名詞であるものを**派生語名詞**という．

接尾語の付いた派生語では「判断力」のように自立語「判断」が接尾語「力」を修飾している場合が少なくない．これは，いわば単語内の係り受けである．そこで，接尾語に係かるという意味でこのような修飾部を**係り語**という．

派生語「合格者名」では接尾語「名」に対して「合格者」が係り語である．ところが，「合格者」の他にも「失格者」，「応募者」など多数の自立語が「名」と結び付いて派生語名詞となる．これらの派生語に共通している末尾の語「者」はまた一つの接尾語であり，かつ，接尾語「名」に係かる**真の係り語**であるということができる．

「合格者」，「失格者」，「応募者」などをすべて接尾語「名」の係り語として登録せずに，真の係り語「者」（名詞の意味分類としては人間のコードを与える）のみを接尾語辞書の見出し語「名」に登録することによって派生語内の係り受け関係が記述でき，かつ派生語処理用辞書群のトータルのサイズも小さくできる．

2.2 派生語の構造

次に本稿で扱う接尾語の付いた派生語を語構成の面から考えて，未登録語となりやすい

主なものを列挙すると次のようになる.

1) 狭義の接尾語が付いたもの
　感情的，金次第，若さ，先生達
2) サ変動詞語幹＋名詞→名詞
　担当者，終了後，点滅器，連絡船
3) 名詞＋名詞→名詞
　郵便局，西洋館，人事部，
4) 名詞＋動詞連用形→名詞，サ変動詞
　水洗い，山登り，刑務所帰り，北向き
5) 動詞連用形＋名詞→名詞
　貸し料，降り口，気取り屋，上げ幅
6) 動詞連用形＋動詞→動詞
　飲み続ける，歩き回る，言い兼ねる
7) 動詞連用形＋形容詞→形容詞
　歩き易い，聞き苦しい，聞き取り難い，

このうち3)，4)，5)の接尾語が本稿でいう接尾語辞書に収録される．なお，ある特定の品詞に常に付く接尾語である6)と7)，また，1)と2)の大部分の接尾語は，われわれの名詞の意味分類には直接関係しないため，述った辞書構成がなされる．

3. 接尾語辞書に必要な情報

接尾語辞書作成に当たっては辞書のデータ構造を考えるのは当然のこととして，その他に

1) どのような語をどれくらい辞書に登録しておけばよいか（自立語辞書に派生語に含まれた形で入れておいた方がよいものもある）
2) 他の機械辞書との関わり方（使用の優先順位など）をどうするか

を考慮しなければならない．

ここで，1)については，2.2節で述べた接尾語のうち，ある特定の品詞に常に付くものは，その品詞や活用形ごとに接尾語をまとめた辞書群を作る．また，特定の語にしか付かないものは接尾語辞書の見出し語とはせず，派生語そのものを自立語辞書に登録する．その他，使用頻度の高いと思われる派生語は，その接尾語が接尾語辞書の見出し語であろうとなかろうと自立語辞書の見出し語として登録する．

個々の接尾語については，それが付く自立語の品詞や活用形，またそれが付くことによってできる派生語の品詞はどうなるかということの他に，

・意味的にどのような範疇の語に付くことが多いのか
・その接尾語が付くことによってできる派生語の意味はどうなるか

ということなどを考慮しなければならない．このためには，自立語辞書の各見出し語の意味分類がなされている必要がある．

われわれの自立語辞書では，名詞についてはそれを35種類に分類し，さらに各名詞を表記する漢字の意味による細分類を行った情報が利用可能であるので，それを用いることにした．

4. 自立語辞書

接尾語辞書作成に当たって使用した辞書は九州芸術工科大学で作成した自立語辞書(KID-J86)で，1986年8月現在，登録語数107，522語，そのうち名詞が93，381語であり，旧版の KID-J 82 を大幅に増補改定したものである．JIS の第1水準以外に第2水準の漢字も用いて漢字表記の充実を図っており，多数の派生語を含んでいる．

自立語辞書は1レコード128バイトの順編成ファイルであり，カナ見出しの降順に配列されている．

各見出し語には，カナ見出しと，漢字表記，漢字表記に対する切れ目という欄を設け，漢字に読み仮名を与える働きをさせている．

用言については，その語幹（上一段，下一段動詞にあっては活用語尾の不変化部分を含む）のみをカナ見出しに与えている．

表1は，自立語辞書で用いている名詞の分類コードとその意味である．

表1　名詞の意味分類コード

Table 1　Meaning category codes of nouns

コード	意　味	コード	意　味
1	他の分類のいずれにも属さないもの	J	生物，法律（～法）と表記されるもの
2	時，方向など，副詞的用法のあるもの	K	貨幣，金銭，給料，財産，経済関係の語
3	形容動詞的に使われるもの	L	建物の一部分や部品
4	人間，神，仏，霊魂，妖精など	M	複雑な機構の機械類
5	動物名，動物，細菌	N	気体，雲，霞，自然現象の語
6	植物名，植物，植物の一部分	O	音楽，曲，歌，芸能関係の語
7	鉱物名，鉱物，液体，自然物など無生物	P	時間，期間
8	飲食物，料理など	Q	衣服，織物，装身具の一部分
9	人間や動物の体，体の一部分	R	スポーツや遊びの道具，用具
A	衣服，織物，装身具，履き物など	S	スポーツ，遊び
B	家，ビル，橋，堤など建造物	T	時刻，時期
C	道具，日用品，簡単な機構の器具類	U	人間が自然物に手を加えてできたもの
D	書籍，評論，文章，詩歌など	V	部屋，室など建物の内部の場所
E	乗り物，交通機関	W	世界や活動分野など抽象的な場所
F	病気，怪我，傷	X	団体，組織，人の集まり，展覧会など
G	弾く楽器，楽器一般	Y	光景，様子，状態，形式
H	吹く楽器	Z	具体的な場所
I	打楽器		

5. 接尾語辞書の作成

接尾語辞書は，そのプロトタイプが，4章で述べた自立語辞書から自動生成される．最終的には，やはり辞書の常として人手による確認とチェックを経なければならない．

5.1 接尾語辞書の自動作成

名詞の意味分類を用いた派生語名詞処理用の接尾語辞書を作成するために，次の条件を適用して，自立語辞書からデータとしての派生語を抽出した．

1) 名詞であること
2) 当て字でないこと
3) 形容動詞的に使われる名詞ではないこと
4) 末尾の語が目的格になっていないこと
5) 全体の漢字数が3文字以上であること
6) 係りの語の漢字数が2文字以上であること
7) 末尾の語の漢字数が2文字以下であること

つぎに，抽出された派生語の係りの語それぞれについて，品詞／意味分類情報を付加するため自立語辞書を検索した．抽出された派生語が接頭語＋自立語の組み合わせであった場合，自立語辞書に接頭語は登録されていないので，係りの語は見つからず，そのような派生語は，この段階で除かれた．

ただし，係りの語が用言の場合，自立語辞書は用言の語幹のみを持つので，係りの語が自立語辞書に見つからないこともあり得る．したがって，自立語辞書に同一の漢字コードが見つかり，かつ仮名見出しが異なる場合には，動詞なら連用形，形容詞なら連体形の活用語尾を自立語仮名見出しに付けて，係りの語の仮名見出しと等しければ，その自立語の品詞情報を係りの語の品詞情報とした．

また，自立語辞書に見つかった係りの語が同じく派生語である場合は，その派生語の接尾語部分を元の接尾語の真の係り語と見なし，抽出した派生語内の係りの語を真の係りの語の情報で置き換えた．

つぎに，このようにして収集した派生語を，接尾語および係り語をソート・キーとして分類した．

最後に，接尾語ごとに，それを含む派生語の出現頻度と，係り語の品詞／意味分類コードの出現頻度を取り，各接尾語に対する複数個の同一係り語は，出現頻度を付して一つのレコードにまとめた．

作成した派生語名詞処理用の接尾語辞書は，接尾語ごとに，

1) その接尾語自体の辞書情報を持つレコード
2) 係り語のレコード（一つ以上）
3) 係り語の品詞／意味分類の出現頻度を示すレコード

からなる．

図1は，派生語名詞処理用の接尾語辞書のフォーマットである．

タイプ1（接尾語）レコード

見出しID	カナ見出し	見出し切れ目	漢字コード														
CNSFX1	接尾語カナ見出し	接尾語カナ見出し切れ目	接尾語漢字コード	小文字数	同形順位		名詞分類1	代表漢字数	名詞分類2	固有名詞	助数詞	人称代名詞	指示代名詞		レコード番号

タイプ2（係り語）レコード

見出しID	カナ見出し	見出し切れ目	漢字コード																		
CNSFX2	係り語カナ見出し	係り語カナ見出し切れ目	係り語漢字コード	小文字数	同形順位	サ変動詞	その他動詞	形容詞	形容動詞	副詞	連体詞	名詞分類1	名詞分類2	名詞分類3	名詞分類4	代表漢字数	名詞分類5	固有名詞	助数詞	人称代名詞	指示代名詞	係り語出現頻度	レコード番号

タイプ3（品詞／意味分類）レコード

見出しID	カナ見出し	漢字コード			品詞出現頻度					名詞出現頻度						
										普通名詞意味分類						
CNSFX3	接尾語カナ見出し	接尾語漢字コード	小文字数	接尾語出現頻度	サ変動詞	他の動詞	形容詞	形容動詞	副詞	連体詞	A～Z, 0～9	固有名詞	助数詞	人称代名詞	指示代名詞		レコード番号

図1 接尾語辞書フォーマット

Fig. 1 Suffix dictionary format

作成結果をまとめると表2のようになる.

表 2　派生語名詞処理用接尾語辞書作成結果
Table 2　Constrcution results of suffix dictionary for derivatives

辞　書	レコード項目	レコード数
自立語	名詞	93381
	接尾語付き派生語名詞	31164
接尾語	接尾語	2090
	平均異なり係り語	12.31

5.2　品詞別接尾語辞書の作成

2.2節で述べたように，接尾語の中には名詞に対して，「次第」,「だらけ」のようにある特定の品詞の語に普遍的に付く，あるいは付くことが非常に多いものがある．また，動詞の連用形に付いて複合動詞を作る動詞など，名詞の意味情報が使えないものがある．それらについては，品詞別に接尾語辞書を作成した．表3に，係り語の品詞・活用形別に収集した接尾語見出し語数を示す.

表 3　品詞別接尾語辞書
Table 3　Suffix dictionaries for specific part of speech

係り語の品詞と活用形	見出し語数
名詞	1770
動詞連用形	1270
サ変動詞語幹	300
形容動詞語幹	50
形容動詞的に使われる名詞	43
形容詞語幹	36

6.　実験結果と考察

ここでは自立語辞書から自動作成された名詞の接尾語辞書を中心に，その特性と問題点について述べる.

表4は，抽出された接尾語をその出現頻度順に並べたものである．個々の接尾語の係り語は，意味的なコードで分類したくらいでは，重なりが多く，その特徴が捉えにくい.

ところで，名詞は普通，漢字表記され，漢字は意味を担っている．このことに注目し，すべての名詞をその語構成（語末の漢字が意味の主体をなすかどうか）と，語末から数えて何文字の漢字が意味の主体をなすかで分類した．その結果，漢字情報によって，係り語に対して，さらに詳しい分類を行うことができた.

表5は，係り語の語末の漢字情報を基にして，出現頻度の高い係り語と接尾語の組み合わせを並べたものである．さらに表6および表7には接尾語「室」と「費」について，それに係る語の品詞と意味分類を出現頻度順に示した.

たとえば，接尾語「室」は，人を表す「長」（名詞の分類コード4）に連接して，自立語辞書の中で40回現われたことを意味する．「長」と名の付くような人は，専用の部屋をもっていることが多いことを表している.

また，動詞性の強いサ変動詞の語幹に接続する語として「費」があるが，これは「造」に連接して13回現われている．物を造るのには金が必要なことを表しているといえそう

表 4 出現頻度の高い接尾語

Table 4 Suffixes of frequent use

No.	出現頻度	接尾語	係り語品詞／意味分類
1	979	者	サ変／3，4，D，E
2	480	性	サ変，形動／7，Z
3	366	品	動詞，形動／3，Z
4	352	機	動詞，固名／X，Z
5	345	室	サ変，4
6	344	法	サ変，形動／X，Z
7	333	部	サ変／O，S，Z，9
8	332	地	動詞／Z
9	313	費	サ変／X
10	281	日	動詞／T

表 5 出現頻度の高い係り語と接尾語の組み合わせ

Table 5 Combinations of kakari-word and suffix of frequent use

No.	接尾語	係り語	例
1	費	送，造，務	輸送費，建造費
2	名	員，者，長	社員名，合格者名
3	品	送，蔵，製	郵送品，愛蔵品
4	者	学，論	科学者，筋論者
5	室	長	学長室

表 6 接尾語「室」への係り語の品詞／意味分類

Table 6 Parts-of-speech/meaning categories of kakari-word connecting with the suffix "室"

No.	頻度	係り語	係り語品詞／意味	例
1	40	長	名詞／4	院長室
2	10	事	名詞／4	理事室
3	8	員	名詞／4	職員室
4	7	務	名詞／1	事務室
5	6	官	名詞／4	教官室
6	6	議	名詞／1	会議室
7	6	等	助数詞	一等室
8	6	理	サ変動詞	管理室
9	5	児	名詞／4	乳児室
10	4	習	サ変動詞	実習室

表 7 接尾語「費」への係り語の品詞／意味分類

Table 7 Parts-of-speech/meaning categories of kakari-word connecting with the suffix "費"

No.	頻度	係り語	係り語品詞／意味	例
1	13	造	サ変動詞	改造費
2	13	送	サ変動詞	移送費
3	13	務	名詞／1	医務費
4	11	料	名詞／8，C	食料費
5	10	備	サ変動詞	軍備費
6	8	築	サ変動詞	建築費
7	7	入	サ変動詞	加入費
8	7	査	サ変動詞	検査費
9	6	助	サ変動詞	援助費
10	6	理	サ変動詞	修理費

である．この他「費」は，「送」，「務」，「料」，「備」，「築」などに連接することが多いが，このことは係り語と接尾語の漢字の意味的なつながりを反映している．

そこで，接尾語辞書のうち，漢字情報が利用できる派生語名詞処理用の接尾語辞書では，各接尾語に対して，それの係り語の語末の漢字表記と意味分類コードを与えて，きめの細かい接尾語処理ができるようにした．

結局，接尾語辞書は，係り語の品詞情報のみを用いる品詞別接尾語辞書と，さらに係り受け情報を用いる派生語名詞処理用辞書の二本立てとなった．

前者については，すでに人手によるチェックを済ませている．内容は，係り語の性質が文法情報のみで決まるものなどからなり，後者の接尾語辞書を補うものである．

後者については，自動作成された接尾語辞書をそのまま使うことは問題があり，自立語辞書との兼合いを考えて，現在チェック作業を行っている．たとえば，「曜」と「日」の関係のように，接尾語辞書の見出し語「日」に対する係り語として「曜」を登録するより，「〜曜日」という語は，せいぜい11の限定された語しかないため，むしろ派生語そのものを見出し語として，自立語辞書に登録した方がよい場合もあるからである．

7. おわりに

自然言語の機械処理には辞書が重要な働きを示すことは当然であるが，なかでも仮名漢字変換などカナ文字文の機械処理では，同形異義語と未登録語の処理が中心課題となり，

その処理のためには予想以上に膨大な容量の辞書が必要となる.

ここで，例として未登録語の中で大きな割合を占める漢語系の派生語名詞について考えてみよう.

これは，大まかにいえば，三文字漢字語の処理をどうするかということになる.

二文字漢字語は，それが同じ格調，同じレベルの文中では，そのまま他の語に置き換えることが困難なことにより，単語としては不動の地位を占めている．これはまた，市販の国語辞書を見ても，二文字漢字語については，日常使用される語は網羅されており，同じ規模同士の辞書であれば，見出し語がほぼ一致していることからもいうことができる.

一方，三文字漢字語は，一般に二文字漢字語からできた派生語である．つまり，確定した二文字漢字に一文字漢字を添えれば，すぐにできあがる簡便性を持っている．このため，造語成分としての漢字の意味を知っている人間なら簡単に新たな語を作り出せるということになる.

このようなわけで三文字漢字語は，その潜在的な数は二文字漢字語の比ではなく，それをカバーしようという機械辞書は，また語構成の仕組みを取り入れた柔軟性を持ったものが要求される.

そこで，われわれのアプローチは，基本的には可能な限り派生語は自立語として取り込むべきだと考え，使用頻度の高い派生語は逐次辞書に登録している.

そして，それができない場合には，辞書に派生語の合成の仕方を記述しておくことにより，変換対象の語を分析して，そのうち適切なものを選ぶという方法を取っている.

本稿では取り上げなかったが，接頭語が付いた語も派生語であり，これの処理は避けて通れない問題である．文法的に処理できる一部の接頭語以外は，本稿で述べたような簡単な係り受けで解決できるものは少なく，新たな観点からの研究が待たれる.

最後に，日頃ご指導ご鞭撻賜わった吉田将教授(九州大学)，稲永紘之講師（九州芸術工科大学）に深甚の謝意を捧げる.

参考文献
[1] 稲永紘之，小西彬允，“カナ文字文のための機械辞書の構成について”，電子通信学会技報，AL 76-39，1976.
[2] 松村明 編，日本文法大辞典，明治書院，1971.
[3] 稲永紘之，吉田 将，“日本語処理のための機械辞書”，情報処理，23-2，1982.
[4] 稲永紘之，橋本和博，吉田 将，“係り受け辞書のカナ漢字変換への応用”，情報処理学会第30回全国大会，1985.
[5] 稲永紘之，橋本和博，“漢字の意味を利用した係り受け辞書によるカナ漢字変換システムについて”，日本ユニバック技報，No. 10，1986.
[6] 稲永紘之，新谷隆之，“意味的なつながりを考慮した接尾語辞書の作成について”，情報処理，NL 57-3，1986.

執筆者履歴 新 谷 隆 之 (Takayuki Shintani)
昭和26年生，49年大阪大学工学部産業機械工学科卒業，同年4月日本ユニバック(株)入社，現在商品開発本部ソフトウェア三部所属，日本語処理に従事．情報処理学会会員.

論文 DS 7 におけるコンカレント CP/M

Concurrent CP/M for the DS 7

阿部 比呂志

要 約 1985年7月に出荷開始された多機能デスク・ステーション DS 7 は，日本ユニバックの端末 UTS 50 の後継機であり，端末としての機能の他にパーソナル・コンピュータとしての機能を併せ持つように設計されたプロダクトである．

DS 7 は，パーソナル・コンピュータとしての汎用性を確保するために Digital Research 社のコンカレント CP/M* を採用している．

本稿では，コンカレント CP/M を DS 7 に移植した際の技術的手法，および新たに追加した機能等について報告する．

Abstract DS 7 is a strategic product designed to provide not only intelligent terminal capability and also personal computer capability. In order to realize its general purposes, we selected Concurrent CP/M released by Digital Research Inc. as the operating system for DS 7. This paper reports some technical considerations for the transplantation of Concurrent CP/M on the DS 7, and its new functions implemeneted.

1. はじめに

日本ユニバックが，1985年5月に多機能デスク・ステーションとして発表した DS 7 では，その OS としてマルチ・タスクとマルチ・ウインドウ機能を有するコンカレント CP/M を採用している．国内でも，このコンカレント CP/M を採用しているメーカは少なくない．

本稿では，コンカレント CP/M を DS 7 に移植する上でマルチ・ウインドウに対して付加したポップ・アップ・ウインドウとマルチ・インジケータ・ライン，ユーザ・メモリの 1 MB までの拡張，さらには DS 7 独特の機能として付加したマルチ・メディア・サポートについて報告する．

2. コンカレント CP/M の概要

本章では，コンカレント CP/M 上で動作する DS 7 独特の機能を紹介する準備として，コンカレント CP/M そのものの概要を述べる．

2.1 DS 7 のコンカレント CP/M

DS 7 が，その CPU として搭載しているのは，Intel 社の 16 ビット・マイクロ・プロセッサ 80186 である．80186 は，8086 をベースにし，処理速度の向上，タイマやダイレクト・メモリ・アクセス等の周辺回路の内蔵化を実現した 8086 の拡張版である．

パーソナル・コンピュータ向けの汎用 OS のベンダの一つである Digital Research 社が，8086 系に対してリリースした最初の OS が CP/M-86 である．同社は，これにマルチ・タスク機能を持たせたものをコンカレント CP/M-86 Version 1.0 として 1982年3月にリリースし，さらに 1983年11月には，DS 7 が採用したコンカレント CP/M-86 Version

* コンカレント CP/M は，Digital Research 社の登録商標である．

3.1 をリリースしている．この Version 3.1 では，マルチ・ウインドウ機能が新たに追加された．

この他にも，8086 系の汎用 OS としては，Microsoft 社がリリースした MS-DOS 86 Version 2.0 などが数多くのメーカで採用されている．しかし，DS7 の開発が始まった 1984 年 6 月の時点で，マルチ・タスクとマルチ・ウインドウ機能を合わせ持っていたのは，コンカレント CP/M-86 Version 3.1 のみであった．この点が，DS7 を採用した最大の理由である．

なお，本稿では，コンカレント CP/M-86 Version 3.1 を単にコンカレント CP/M と略記する．

2.2 コンカレント CP/M の内部構造

コンカレント CP/M の内部構造は，図 1 に示すように，いくつかのモジュールで構成されている．基本的には，ハードウェア依存部の拡張入出力システム (XIOS) と，物理的なハードウェア構造からは独立したハードウェア非依存部のスーパバイザ (SUP) の二つの部分からなる．

コンカレント CP/M の特徴であるマルチ・タスク機能を実現するのが，スーパバイザ内のリアルタイム・モニタ (RTM) で，最大 255 個の並列タスクのスケジューリングとディスパッチング，タスク間の同期と通信，排他制御のためのセマフォ管理など，タスクに関するあらゆる管理機能を統括する．

マルチ・ウインドウ機能に関与するモジュールは，仮想コンソール管理部 (VCM) と，ウインドウ管理部 (WM) である．これらの役割については後述する．

一方，XIOS は，CRT，キーボード，プリンタ，ディスケットなどを直接駆動する I/O ハンドラの集合である．解像度の異なる CRT や，キー配列の異なるキーボードなど，メーカによって差異のあるハードウェアに合わせて XIOS は作られる．

ハードウェア非依存部のスーパバイザは，Digital Research 社のようなベンダがリリー

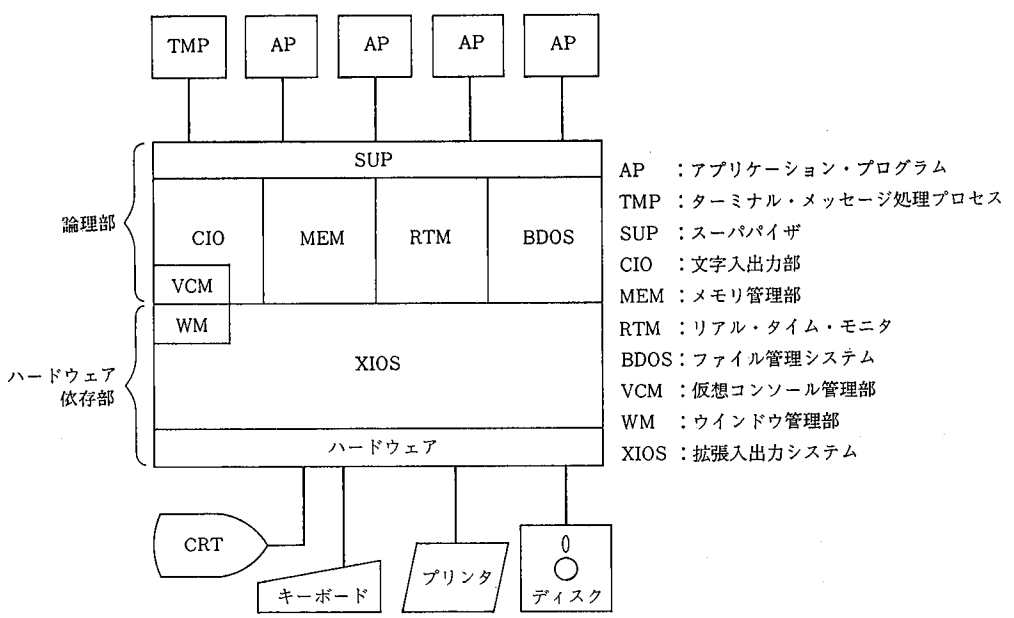


図 1 コンカレント CP/M の内部構造

Fig. 1 Block diagram of Concurrent CP/M

スし，ハードウェア依存部の XIOS は当社のようなインプリメンタが開発する．一般的に，この XIOS の開発のような作業を汎用 OS の移植といっている．

3. マルチ・ウインドウ機能の拡張

本章では，コンカレント CP/M が持つマルチ・ウインドウの上に他社のコンカレント CP/M にはない DS 7 独自の機能として付加したポップ・アップ・ウインドウとマルチ・インジケータ・ラインについて記述する．

3.1 マルチ・ウインドウ機能

コンカレント CP/M のマルチ・ウインドウ表示は，図 2 のような方法で実現されている．

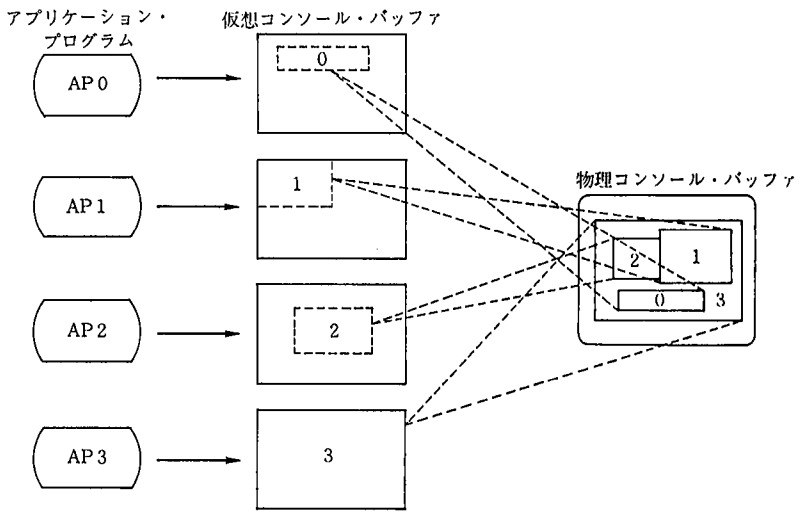


図 2　マルチ・ウインドウ表示

Fig. 2　Multi-window display

DS 7 の CRT コンソール（仮想コンソールと区別するために，以下では物理コンソールと呼ぶ）上に表示されるのは，物理コンソール・バッファの内容であるが，マルチ・ウインドウ表示のために，メモリ上の別の位置に物理コンソール・バッファと同一フォーマットの仮想コンソール・バッファを用意している．標準のコンカレント CP/M では，この仮想コンソール・バッファを 4 個持っている．

アプリケーション・プログラム（AP）には，それぞれ一つの仮想コンソール・バッファが割り当てられ，プログラムからの文字の出力は，この仮想コンソール・バッファに書き込まれる．

そして，XIOS 中のウインドウ管理部が，各仮想コンソールの指定部分を切り出し，物理コンソール・バッファの指定位置に転送することによって，マルチ・ウインドウ表示が可能になる．

次に，仮想コンソール・バッファへの文字の出力が，どのようにして物理コンソール上に表示されるかを図 3 を用いて説明する．なお，説明を単純にするため，ここではウインドウの数（つまり，仮想コンソール・バッファの数）を 3 個とする．

仮想コンソール・バッファ上の文字が，物理コンソール上に表示される過程では，次の四つのロジックを通る．

・ウインドウ切り出し
・表示位置調整
・マッピング・メモリ
・プライオリティ・テーブル

3.1.1 ウインドウ切り出し

仮想コンソール上の指定された範囲の矩形部分が，物理コンソール上にウインドウとして表示されるわけであるが，このロジックは表示対象の文字が，この矩形の範囲内か，範囲外であるかを判定するフィルタである．範囲外である場合には，以降のロジックには進まない．

図3の例では，仮想コンソール0上の文字「ア」が，ここでふるい落とされる．

3.1.2 表示位置調整

仮想コンソール上の矩形部分を，物理コンソール上の任意の位置に移動して表示できる

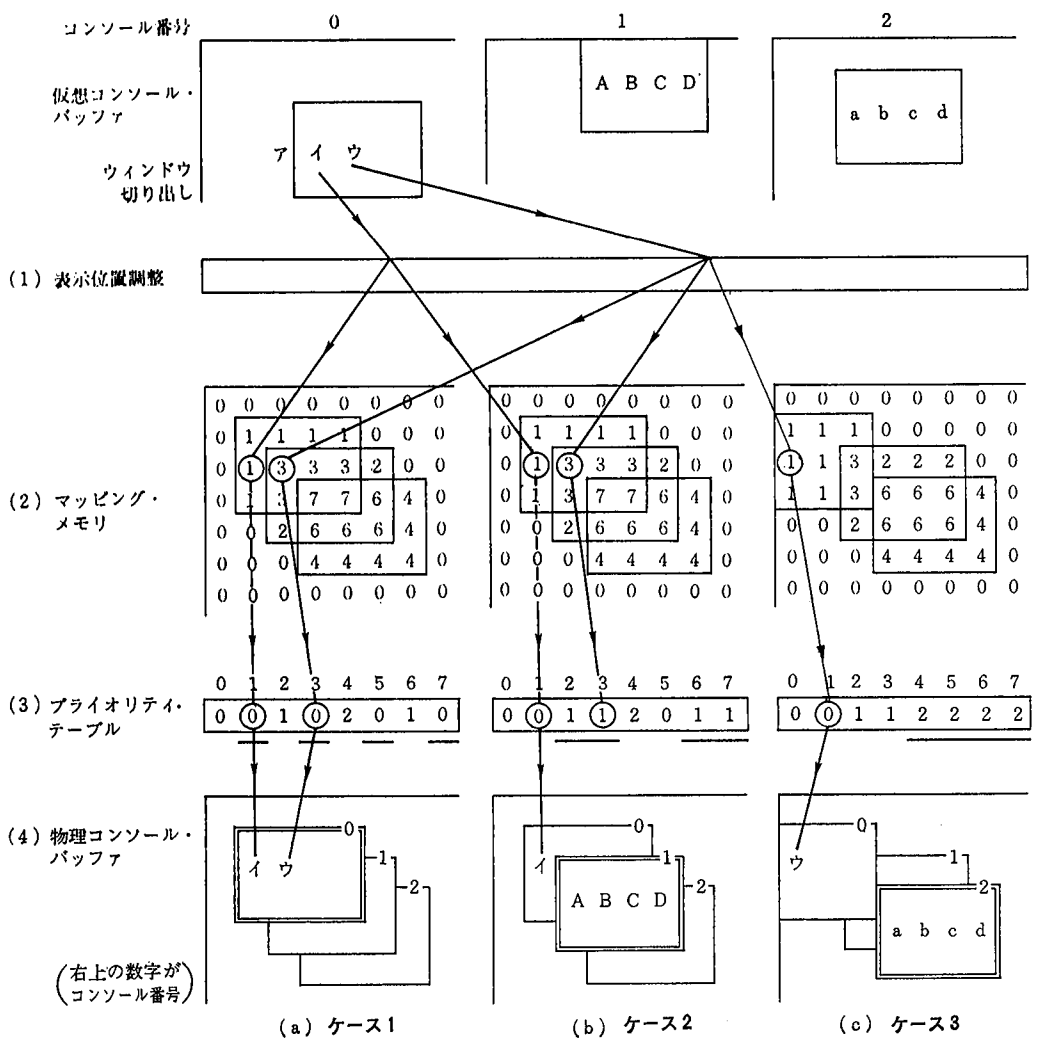


図 3 ウインドウ制御のロジック

Fig. 3 Mechanism of window control

のも，このウインドウ制御の特徴である．極端に言えば，仮想コンソール上では左上隅にある矩形部分を，物理コンソール上の右下隅に表示することが可能である．

こういった移動を制御するのが，表示位置調整のロジックであり，物理コンソール上の矩形部分が，仮想コンソール上の矩形部分よりも小さい場合（つまり，仮想コンソール上の文字が物理コンソールの端より，はみ出るような場合）には，余分な部分をカットするフィルタとして働く．

図3の例では，ケース3における仮想コンソール0上の文字「イ」が，ここでふるい落とされる．

3.1.3 マッピング・メモリ

マッピング・メモリは，物理コンソールに表示できる文字数と同じバイト数を持ち，同じようなロー／カラム構成を持つバッファであり，それぞれのバイトがウインドウの重なり具合を示す．

バイト中のビット0がコンソール0，ビット1がコンソール1といったように割り当てられ，各コンソールの矩形部分内に対応するバイト中のビットが，“1”にセットされ，矩形部分外が“0”にリセットされる．

図3の例では，物理コンソール上でのウインドウの重なり具合に合わせ，実際のマッピング・メモリにセットされる値を示した．ケース3の場合には，コンソール0のウインドウが移動しているため，マッピング・メモリの内容が他のケースとは異なった値となっている．

3.1.4 プライオリティ・テーブル

ウインドウ制御ロジックの最終段のフィルタが，このプライオリティ・テーブルである．

このテーブルは，マッピング・メモリの各バイトが取り得る値の数（図3の例では，ウインドウの数が3個であるから $2^3=8$）だけのエントリを持ち，ウインドウの重なりの優先度が登録される．

制御ロジックは，マッピング・メモリの内容をインデックスとして，このテーブルのエントリを取り出し，その値と出力しようとするコンソールの番号が一致した場合のみ，物理コンソール上に，文字を出力する．

図3の例では，ケース2における仮想コンソール0上の文字「ウ」が，ここでふるい落とされる．

このテーブルの更新は，物理コンソール上でのトップ・オブ・スクリーン・ウインドウ（一番上に表示されるウインドウで，図3では二重枠で囲んで示す）が切り換わるたびに，テーブル中の対応する部分（図3のプライオリティ・テーブルの下のアンダーライン部分）に，トップのコンソール番号が登録されることによって行われる．

3.2 表示の高速化

前項で説明したように，一つの文字が物理コンソール上，つまり画面上に表示されるまで，複雑なロジックを経由しなければならず，かなりの CPU タイムを必要としている．

これを少しでも回避するために，ウインドウ管理部は，ある特殊なケースでは制御ロジックを経由しない方法をとっている．

特殊ケースとは，トップ・オブ・スクリーン・ウインドウが 24×80 の大きさ，つまりフル・スクリーンのサイズで，ウインドウとして切り出されていない場合（見方によっては，ウインドウを切り出している状態が特殊ケースといえるかもしれないが）である．こ

の時の文字の出力は，仮想コンソール・バッファには書かれずに，直接物理コンソールに書かれる方法をとり，CPU タイムを軽減し表示の高速化を図っている．

なお，この場合には，仮想コンソール・バッファと物理コンソール・バッファの内容とが一致しなくなるが，ウインドウに変化があった時（コンソールを切り換えた時など）に，物理コンソールから仮想コンソールへ逆向きに転送して一致させている．

3.3 ポップ・アップ・ウインドウ機能

一般に，パソコンでは実行させようとするプログラムや，プログラムに与えるパラメータの指定などに関して，キーボードからプログラム名やパラメータ名の文字列を入力する代わりに，プログラム名やパラメータ名などの一覧表を表示し，その中から必要なものをワンタッチ（最小限のキー入力）で選択させるメニュー選択方式をとっている例が多い．この方式でメニューを重ねられるものは，ポップ・アップ・メニューあるいはプル・ダウン・メニューなどと呼ばれている．

DS7 上にも，この方式を取り入れようとコンカレント CP/M のマルチ・ウインドウ機能の中に，独自の手法によるポップ・アップ・ウインドウ機能を付加した．

これは，図4に示すように，通常の4個のウインドウの他に5個目のウインドウを追加し，トップ・オブ・スクリーン・ウインドウに対するプログラム選択などのメニュー表示を行うものである．また，このポップ・アップ・ウインドウは，常にトップ・オブ・スクリーン・ウインドウよりも上に表示されるようになっている．

ポップ・アップ・ウインドウ機能の実現のために，次のような変更をウインドウ管理部に施した．

3.3.1 仮想コンソール・バッファの追加

5個目のウインドウのサポートのため，4個であった仮想コンソール・バッファを1個追加し，ポップ・アップ・ウインドウへのメニュー文字出力用のバッファとした．

3.3.2 プライオリティ・ロジックの変更

ポップ・アップ・ウインドウを常に一番上に表示させるため，プライオリティ・テーブルの後ろ半分の32エントリ（総エントリ数は，ウインドウの数が5個であるから $2^5=64$）

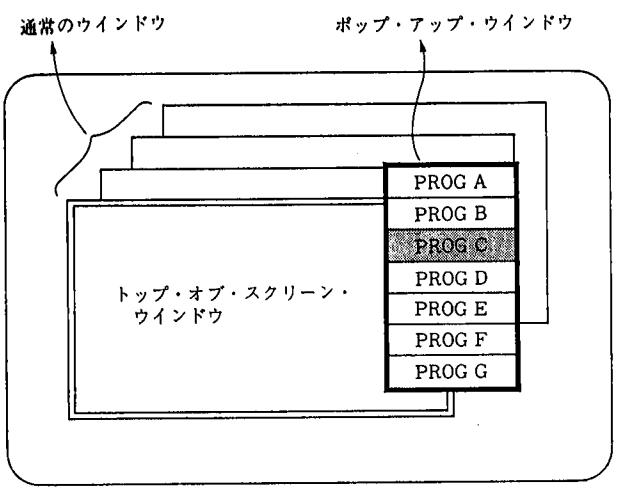


図 4 ポップ・アップ・ウインドウ

Fig. 4 Pop-up window

を，いつでもポップ・アップ用のコンソール番号である“4”に固定し，通常のコンソール出力が，ポップ・アップ・ウインドウ上の表示を壊さないようにした．

このようにしてインプリメントされたポップ・アップ・ウインドウ機能であるが，物理コンソール上に一つのメニューしか表示できないため，メイン・メニューとサブ・メニューの同時表示といった，複数のポップ・アップ・ウインドウの表示には適さない．しかし，一つのメニュー表示ですむような場合には，強力なオペレータ・インタフェース用ツールとなっている．

3.4 マルチ・インジケータ・ライン機能

DS 7 の物理コンソールは，80 カラム×25 ラインの表示が可能であり，そのうちの上から 24 ラインを通常のキー入力用エリアとして使用し，マルチ・ウインドウの表示も，この 80 カラム×24 ラインの範囲内で行われている．

一方，最下行である 25 ライン目は，トップ・オブ・スクリーンのコンソール番号，プログラム名や時刻の表示といったステータス表示用のシステム・ステータス・ラインとして使用されている．

しかし，この 25 ライン目をシステム・ステータス・ラインの表示専用としてだけでなく，UTS 50 エミュレータ・プログラムでは，UTS 50 と同じようなロー／カラムといったアプリケーション・プログラム固有のメッセージ，他のコンソールの利用状況，さらに

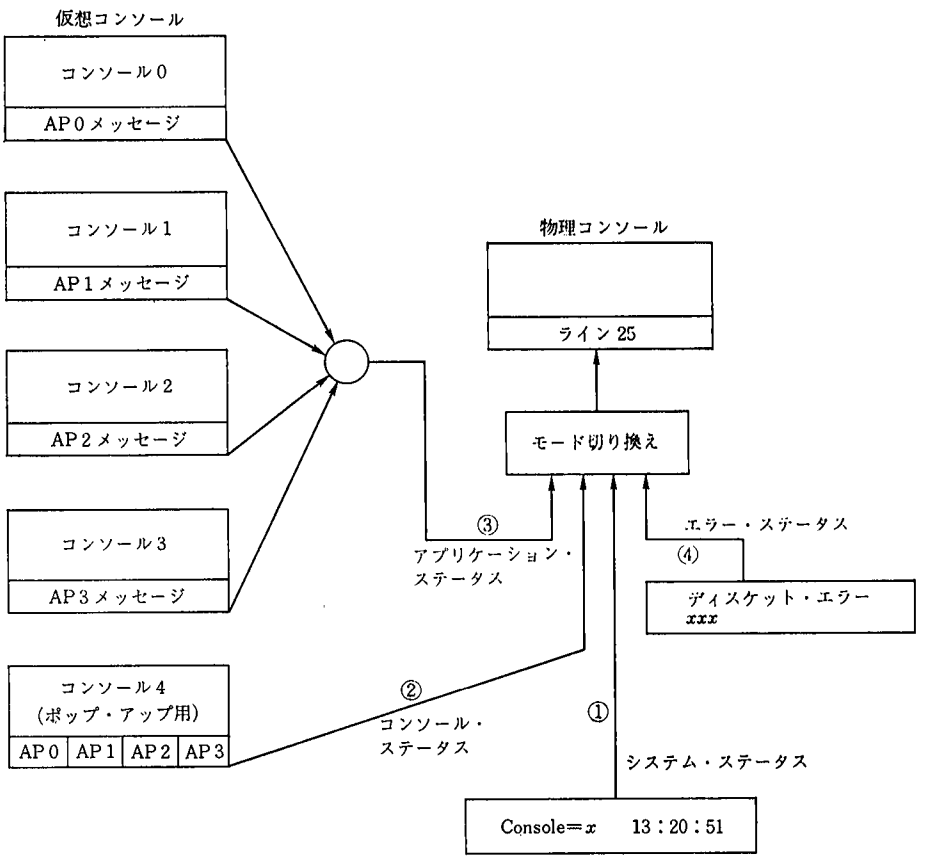


図 5　マルチ・インジケータ・ライン

Fig. 5　Multi-indicator line

はディスケットやプリンタのエラー・メッセージなど種々の表示に使用したい，という要求が生まれた．

この要求を満足させるために，他社のコンカレント CP/M にはないマルチ・インジケータ・ライン機能を追加することになった．

これは，図5に示すように，物理スクリーン上の25ライン目にモード切り換えにより，以下の4種類のメッセージを表示できるようにしたものである．

① システム・ステータス……従来のシステム・ステータスに，キーボードのカタカナ・シフト状態とキャラクタ／グラフ表示モード状態を付加したメッセージ．

② コンソール・ステータス……1ラインを4分割し，4個のコンソールすべての利用状況を，一目で見ることができるメッセージ．

③ アプリケーション・ステータス……アプリケーション・プログラム固有のメッセージで，トップ・オブ・スクリーンのコンソールからのメッセージ．

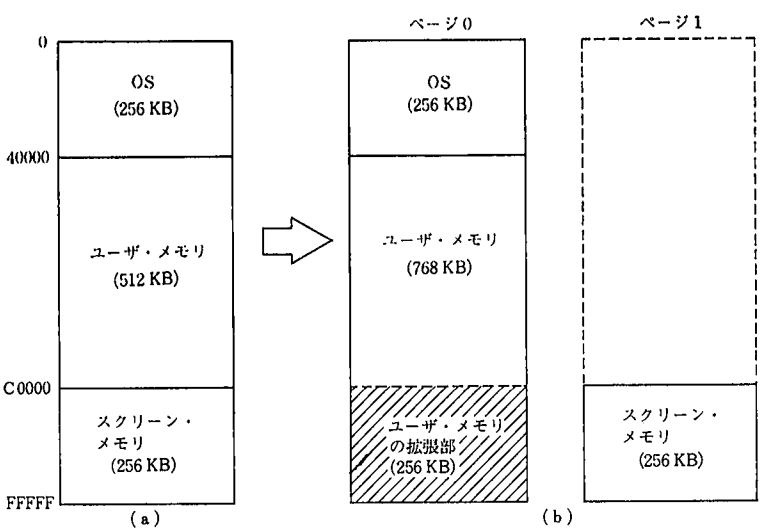
④ エラー・ステータス……ディスケットやプリンタのエラー発生時に，オペレータの対応を求めるメッセージ．

モード切り換えは，①から③については，プログラムあるいはキーボード入力によって行われるが，④のエラー・ステータスは，即刻オペレータの介入を許すために，XIOS が他のメッセージに上書きをして表示し，あとで元のモードに戻すといった方法をとった．

このマルチ・インジケータ・ライン機能の追加によって，他社のコンカレント CP/M と比較しても，より柔軟で，より親切なオペレータ・インタフェースとなったと言えよう．

4. ユーザ・メモリの拡張

DS7 の CPU である 80186 は，1メガ・バイトまでのメモリ空間をサポートできることから，当初図6の(a)のようなメモリ・マッピングで使用していた．しかし，アプリケーション・プログラムのロードされるユーザ・メモリが512キロ・バイトだけだと，大き



(a) 旧メモリ・マッピング　　(b) 新メモリ・マッピング

図 6 メモリ・マッピング

Fig. 6 Memory mapping

なプログラムが2本くらい走ると，他のプログラムがロードできなくなり，せっかくのマルチ・タスク機能が生かされなくなる．このため，ユーザ・メモリをもっと拡張したいという要求が生まれた．

そこで，ハードウェアが持つページ・スイッチング機構* を利用し，スクリーン・メモリの部分をページ1の空間に追い出し，ページ0の空いた部分をユーザ・メモリとして使用し，合計 768 KB のユーザ・メモリにした（図6の(b)）．

これで，ユーザ・メモリの拡張の要求は満たしたが，プログラムの実行において以下のような問題が発生した．

1) ユーザ・スタックの問題
2) ディスケットのリード／ライト・バッファの問題

4.1 ユーザ・スタックの問題

プログラムの実行中に，タイマなどのハードウェアの割り込みが生じると，プログラムの実行が中断され，タイマ割り込みの処理ルーチンに制御が移る．このとき，プログラムの中断点のアドレスが，スタック・ポインタで示されるプログラム中のスタック領域に書かれる．

ここで，プログラムがスクリーン・メモリのアドレスと重なる位置にロードされた場合を考えてみる．プログラムからの要求により，XIOS がスクリーン・メモリをアクセスする時には，ページ0からページ1に切り換えるが，ページ1になっている最中に割り込みが生じると，中断点のアドレスがページ0にあるユーザ・メモリのスタックにではなく，ページ1のスクリーン・メモリ上に書かれてしまい，画面表示が壊されるという問題が発生する（図7）．

この問題を解決するために，XIOS がスクリーン・メモリをアクセスしている間（つまりページ1にしている間）には，割り込みをディセーブルするようにした．しかし，コミュニケーション・ラインからの割り込みなどの急を要するルーチンのサービスのために，一度に多量のバイトをブロック転送するようなスクリーン・アクセスができず，数バイトずつの細切れの転送をし，そのたびに割り込みをイネーブルしなければならなくなった．

これを根本的に解決するには，割り込みが生じた時にハードウェアが自動的にページ0に戻すといったことが必要となってくる．

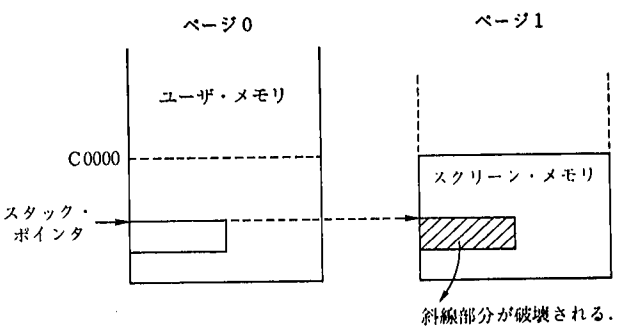


図 7　スタック位置によるスクリーン破壊

Fig. 7　Screen destruction at some stack position

* ページ・スイッチング機構とは，ハードウェアのメモリ制御用レジスタの内容を変えて，CPU がアクセスするメモリ空間を切り換える機構である．

4.2 ディスケットのリード／ライト・バッファの問題

ディスケット，あるいはディスクのリード／ライトによるデータ転送に関する CPU の負荷を軽減させるために，DS7 のハードウェアは，ダイレクト・メモリ・アクセス機構 (DMA) を持っている．

この DMA とは，ディスケットとリード／ライト・バッファとの間のデータ転送を，CPU の処理とは無関係に，コマンドを与えるだけで自動的に行う便利な機構である．

しかし，ここでも 4.1 項と同じような問題が生じてくる．それは，リード／ライト・バッファがスクリーン・メモリのアドレスと重なる位置にある場合に，DMA によるデータ転送がスクリーン・メモリとの間で行われ，画面表示が破壊されるという問題である（図 8）．

この問題の解決策としては，図 9 に示すように，ページの影響を受けない OS の領域にローカル・バッファを設け，このバッファに DMA 転送した後に，目的のリード／ライト・バッファへ CPU によるブロック転送を行うようにした．

ローカル・バッファの採用に当たり，そのバッファ・サイズを決定するためにディスケット・アクセスにかかる時間を測定したところ，表 1 のような値が得られた．

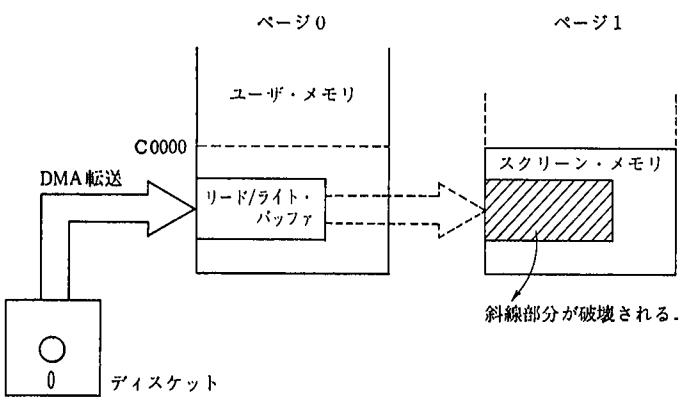


図 8 DMA 転送によるスクリーン破壊

Fig. 8 Screen destruction caused by DMA transfer

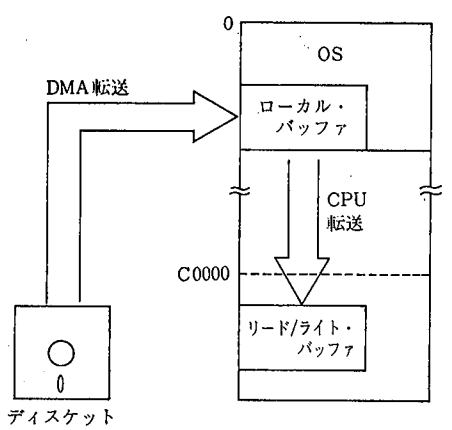


図 9 ローカル・バッファの採用

Fig. 9 Implementation of local buffer

表 1　ディスケットの複写にかかる時間

Table 1　Elapsed time for diskette copy

ローカル・バッファを使用しない場合	ローカル・バッファを使用した場合のバッファ・サイズ		
	8 Kバイト	4 Kバイト	2 Kバイト
2分30秒	2分36秒	3分54秒	6分30秒

注）ローカル・バッファを使用しない場合とは，リード／ライト・バッファのアドレスがスクリーン・メモリに重ならない場合のことである．

この結果から，ローカル・バッファのサイズは，8 KB にすることにした．

5.　マルチ・メディアのサポート

一般にパーソナル・コンピュータは，プログラムやデータを保存するためのメディアとして，ディスケットを使用している．しかし，同じ OS を採用している機種同士でも，ディスケットの記録容量やフォーマットが異なり，機種間のデータ交換が容易でない．

DS7 では，この問題に対処すべく内蔵の 5 インチ FDD に関して，UTS 50 のタイプ Hフォーマット（記録容量 1 MB）および CP/M フォーマット（同 1 MB），また UP 10 E/70 の CP/Mフォーマット（同 1.2 MB），さらには多くの機種が採用している MS-DOS フォーマット*（同 650 KB）の 4 種類のディスケット・メディアをサポートする機能を組み入れた．このマルチ・メディアのサポート機能を説明する．

たいていのディスケットは，トラック 0 のフォーマットにより，ディスケット全体のフォーマットが判別できることから，XIOS の中に表 2 のような特性テーブルを用意し，各ディスケットのトラック 0 のフォーマットを表すいくつかの要素を，特性テーブルにあらかじめ登録しておく．

アクセスしようとするディスケットが，最初にセレクトされた時に，特性テーブルの 1 番目のエントリの要素を使用してトラック 0 をリードしてみる．そこで，エラーが生じた場合（数回のリトライ動作を含む）には，2 番目のエントリを用いるといったトライ・アンド・エラー方式で，適合するフォーマットであるかどうかを判断するわけである．

なお，特性テーブルのエントリ順は，使用頻度の高いフォーマット順に並べ，アクセス効率の低下を防いでいる．

表 2　5インチ・ディスケットの特性テーブル

Table 2　Characteristics of 5 inches diskettes

エントリ順 ＼ フォーマット ／ 要素	1	2	3	4
	CP/M (1.2MB)	CP/M (1.0MB)	タイプH (1.0MB)	MS-DOS (650KB)
記録方式	MFM	MFM	FM	MFM
バイト数／セクタ	1,024	256	128	512
セクタ数／トラック	8	26	26	8
全トラック数	154	154	154	160
モーター・スピード〔rpm〕	360	360	360	300

MFM : Modified Frequency Modulation
FM : Frequency Modulation

* MS-DOS のバージョン 1.XX のフォーマットであり，バージョン 2.XX のフォーマットはサポートされない．

この機能の実現によって DS7 と他機種間とのデータ交換が，より広範囲になったのは言うまでもない．

6. おわりに

他社のコンカレント CP/M より優れたものということを目標にして進めてきた DS7 のコンカレント CP/M 移植作業は，基本部分のインプリメントの他に，前章で説明したような新機能の実現により，ほぼ満足のいく結果が得られた．

今後のパーソナル・コンピュータは，32ビットのマイクロ・プロセッサの導入，強力なコミュニケーション機能の実現や，大容量メモリ空間をカバーする仮想記憶の考えを持った汎用 OS の出現などにより，従来の大型機のアーキテクチャに近づいてゆく傾向がある．今回の開発で得られた知識や手法が，次期プロダクトにも反映できるように努力してゆきたい．

参考文献 ［1］ 吉本万寿夫，“マルチウインドウ表示機能を内蔵するマルチタスク OS”，日経バイト，Apr. 1984, pp. 11～18.
［2］ 阿部比呂志，“Multi-window on UTS 50”，TECHNICAL MEMORANDUM，日本ユニバック，TM-1176, 1984.

執筆者紹介 阿 部 比呂志 (Hiroshi Abe)
昭和50年国立宮城工業高専電気工学科卒業．同年日本ユニバック(株)入社．テクニカル・サポート担当エンジニアを経て，54年よりマイクロ・プロダクトの開発に従事．現在，マイクロ・ハードウェア開発部ワークステーション開発一室に所属．

Lisp マシンのウインドウ・システム

The Window System on Lisp Machine

大田一久
K. Ohta

1. はじめに

近年，コンピュータ・システムのユーザ・インタフェースの形態が変化してきている．この変化を支えているのは，ワークステーションと呼ばれる新しい小規模のコンピュータおよびそのソフトウェアである．本稿では，このようなユーザ・インタフェースの進歩を支える技術を紹介することを目的に，ワークステーションとして Lisp マシン（日本ユニバックの KS-301）を取り上げる．そして，そのユーザ・インタフェースをサポートするソフトウェアであるウインドウ・システムについて解説する．

1.1 ユーザ・インタフェースの動向とその背景

最近のシステムのユーザ・インタフェースを以前のシステムのそれと比較すると，次のような点が変化してきている．

まず，従来，文字テキストで行われてきたユーザとの会話が，アイコンのような図形を交えた，より視覚的なものになってきたことである．

二つめは，このため図形の表示や入出力を柔軟に行うことのできるデバイスの進歩が挙げられる．すなわち，コンソールのハードウェアが文字専用のディスプレイとキーボードの組み合わせから，図形と文字を統一的に扱うビットマップ・ディスプレイにマウスを加えた構成に変わってきたことである．

三つめとして，ユーザに対する積極的な支援の機能である．すなわち，ユーザの入力の誤りを指摘することから，一歩進んでユーザに誤りをおこさせないように補助する方向になってきている．この一つの例がメニュー形式の入力であり，正しくないものは入力できないようにしている．このため，入力量を考慮して暗号のような省略形を用いる必要もない．これに関連して，オンラインでユーザの必要とする操作上の情報を与える仕組みも発達してきている．以前からユーザの要求に応じて説明を与えるものはあったが，最近はメニューの中の選択肢にカーソルを位置づけると，その選択肢の説明が画面上に表示されるものなどが現われている．

もう一つ指摘すると，文字テキストで行単位に情報を交換する形式の従来のユーザ・インタフェースから，より多くの情報を大域的に提供するような画面単位の情報の交換へと変化しており，ユーザとシステムの間のコミュニケーションの幅が広くなってきている．

このようなユーザ・インタフェースを実装した例として，次のものを挙げることができる．

1) Xerox D シリーズ Smalltalk[1]…Smalltalk はオブジェクト指向プログラミング言語として有名であるが，単なるプログラミング言語を越えてプログラミング環境として新時代を切り開いたシステムである．とくにマウス，ビットマップ・ディスプレイによるアイコン，メニューを多用したユーザ・インタフェースは，その後のシステムに大きな影響を与えた．また同じハードウェアには，Lisp の環境を実装した Interlisp-D がある．

2) Apple Macintosh[2]……パソコンで最新のユーザ・インタフェース技術を駆使したマシンであり，このような技術が必ずしも高価なものではないことを実証した．数多くのアプリケーション・ソフトウェアが，同様のユーザ・インタフェースで提供されている点は注目に値する．

3) MIT 系 Lisp マシン[3]……MIT での研究成果に基づいて商品化されたワークステーションで，現在 Symbolics, LMI, TI の 3 社が製造している．日本ユニバックが販売している KS-301 は，このうち TI 社の製品を OEM しているものである．Xerox D シリーズ+Smalltalk とねらいは似ているが，より実用指向ということができる．本稿では，この系統の Lisp マシンのユーザ・インタフェースのサブシステムであるウインドウ・システムを取り上げる．

4) Unix 系*ワークステーション……Unix を搭載した汎用ワークステーションが数多く市場に出てきているが，それらの中にもマルチ・ウインドウ・システムを実装してユーザ・インタフェースの改善を図っているものが現れている．その中で代表的なものが，Sun Microsystem 社の SUN ワークステーション上のウインドウ・システムである SunView および News，あるいは MIT を中心に開発され DEC, SUN

* Unix は，米国 AT&T 社の Bell 研究所で開発されたオペレーティング・システムで，AT&T 社によってライセンスされている．

などのワークステーションで動作する X-window[4] である．上記1）～3）が，ある意味では特殊な環境で動作するのに対して，これらは移植性と汎用性を持ち設計も新しいことからウインドウのモデル等がより整理されており，次の世代のウインドウ・システムであるとも言える．

このような動向の背景としては，次の点を挙げることができる．

1) デバイスの進歩……先にも述べたようにビットマップ・ディスプレイ，マウスといったデバイスによって図形の入出力などが容易になり，また操作性が著しく向上することになった．

2) 資源のコストの低下……メモリや CPU の速度の向上と価格の低下によって，ユーザ・インタフェースのために計算機の能力をより多く割り当てることが可能になったことである．すなわち画面の制御のために，あるいはマウスの動きを追跡するために CPU を使い，オンラインでユーザに与える情報をメモリ上あるいはディスク上に持つことができるようになった．

3) ネットワークの出現……ユーザ・インタフェースのために計算機資源が必要であることはもちろんであるが，さらに複雑な制御に対しても応答性が必要である．大規模 TSS 環境ではこの点に限界があった．このためにはマシンをパーソナル化することが一つの回答であるが，システムとユーザのコミュニケーションの幅を広くすることができる，反面 TSS のメリットであるファイルの共有や電子メイル等の機能が失われてしまう．この点を補うためにネットワークが利用されるようになり，パーソナル・マシンであるワークステーションの実用性が高くなった．

この三つの点は，すなわちワークステーションの特徴であり，ワークステーションの出現がユーザ・インタフェースの進歩の大きな要因であると言ってよい．

1.2 Lisp マシン

ここで MIT 系 Lisp マシンの歴史，目的および特徴について簡単に紹介しておく．Lisp そのものの歴史は1950年代にさかのぼり，多くの処理系が汎用機の上に作られてきた．とくに MIT では DEC-20 の上の Maclisp が使われていたが，大規模 TSS での応答性の問題やメモリ空間の制約などから，その限界を越える専用マシンが計画された．

このマシンの目的は次の三つであった．一つは当然のことながら Lisp を効率よく実行することである．Maclisp 以前にも Lisp を効率よく実行するためのテクニックが数多く提案され使われてきたが，Lisp の特徴である関数呼び出しの多用，大量のメモリ消費，実行時の型判定等は汎用機の上では効率化に反する要件であった．これらの要件を満たすアーキテクチャを Lisp 専用に開発することが第1の目的であった．

二つめはプログラム開発実験環境である．Lisp は主に人工知能の実験プログラムの開発のために使われていたが，システムの仕様があらかじめ詳細に決まっていることはまれで，インクリメンタルに開発と拡張が行われていた．このような開発形態をサポートするために，コード，テスト，デバッグのサイクルを短縮することが要求された．ここには当然操作性の向上も含まれ，ユーザ・インタフェースの改善に重きが置かれた．

最後の一つは，研究者たちの統合ワーク・ステーションとすることである．TSS 環境でも，プログラムの開発の他に論文の作成のための文書清書システム，研究者同士のコミュニケーションのための電子メイル・システム等の機能が使われていたが，これらの機能を引き継ぎ拡張し，つまりその前に座ればすべての仕事が済むようにすることであった．

このような目的に基づいて設計・制作されたマシンは，次のような特徴を持っている．

まず Lisp 向きのアーキテクチャであるが，具体的には関数呼び出しのコード生成を容易にするスタック指向アーキテクチャ，メモリ管理，データ型の判定を支援するタグ付きメモリ，そして大きな仮想記憶空間である．つぎに充実したプログラミング支援ツールである．Lisp では動的リンクが標準で言語そのものがインクリメンタルに開発しやすいようにできているが，さらにエディタ，コンパイラ，デバッガ，トレーサなどが揃っている．これらのツールは，本稿で取り上げるウインドウ・システムを駆使した操作性と，一覧性の高いユーザ・インタフェースを持っている．また，このウインドウ・システムや他のソフトウェアをサポートするオブジェクト指向プログラミング・サブシステムや，フレーバ・システム (FLAVORS)[5][6] が大きな特徴である．フレーバについては次節で取り上げる．このように Lisp マシンは，他のワークステーション同様，パーソナル・マシン化することで，高いパフォーマンスを得ているが，ネットワークで結合して使用することで，その真価を発揮できるように設計されている．

1.3 フレーバ・システム

Lisp マシンでは，オブジェクト指向プログラミングのための Lisp 言語の拡張として，フレーバ・システムと呼ばれる副言語をサポートしている．フレーバ (Flavor) という言葉は，このサブシステムそのものを指す場合と，この言語での型に相当する概念を表す場合と両方に用いられている．ウインドウ・システムはフレーバ・システムを用いて実現されており，ウインドウ・システムの機能はフレーバ・システムなくしては有りえないと言っても過言ではないであろう*.

オブジェクト指向プログラミングでは，プログラムを書くことは，型を定義してその実現を与えることである．従来のプログラミングでは，型を用いてプログラムを書くのに対して，オブジェクト指向プログラミングではプログラムを書くことで型そのものを実現すると言ってもよいだろう．ここで型とはデータの性質を指し，その型を持つデータに対しての操作およびその結果として性格付けられる．このとき操作に対する結果を得る過程を与えることが型の実現である．オブジェクト指向プログラミングでは，この型をクラスと呼び，型のデータをインスタンスと呼ぶ．フレーバ・システムではとくにフレーバという言葉をクラスの代わりに用いる．また，操作に対する結果を得る手続きをメソッドと呼ぶ．オブジェクト指向のプログラミングでは，システムの仕様をシステムそのものの型の性質と考え，その実現の際に必要な型をさらに実現すると言う形で分割詳細化を行ってゆく．

この種のプログラミングをサポートする言語としては Smalltalk[9] が有名で，その他にもいくつかの言語が提案されている．それらは大きく二つのグループに分けることができる．一つは，Smalltalk のようにまったく新しい言語として設計されたものである．もう一つは，フレーバのように他の言語の拡張として設計されたもので，C に対する Objective-C などがある．前者は稼働するマシンが限られることから後者の方が，より一般性があると考えられる．

フレーバの最大の特徴は組み合わせの機能である．従来もサブルーチン，あるいは手続きという形でプログラムの一部を抽象化し，共用することが行われてきた．サブルーチンはプログラムとしては不完全で，その挙動は外部の環境と引数などに依存している．したがって，意図した結果を得るために使用に当たってのプロトコルが決められているのが普通である．フレーバ・システムの場合も事情は似ており，型の性質の一部を抽象化し，共用することが行われる．この共用される不完全な型をとくにミキシン (Mixin) と呼ぶことがあり，他のフレーバと組み合わせて意図した性質を持つ新しいフレーバを作り出す．組み合わせる場合に他の型の性質に依存する場合があり，組み合わせの方法などのプロトコルがフレーバごとに決められている．

Smalltalk がプロトタイピングを重視しているのに対して，フレーバ・システムはより実用に耐えるシステムを記述することを目的としており，アーキテクチャ上のサポートやコンパイル方法の工夫および最適化が行われている．その一方，セマンティクスが厳密に定義されているとは言いがたいが，オブジェクト指向プログラミング・システムとしては実用の域に達しているものの一つである．

フレーバ・システムの応用例としては，次のものがある．まずその第一はウインドウ・システムで，ウインドウの多くの属性を組み合わせ，管理するのに威力を発揮しており，同時に画面の表示等に要求されるパフォーマンスを確保している．もう一つはネットワーク制御ソフトウェアで，各種のホスト，パス名，通信プロトコルの性質がフレーバとして管理されている．その他にも I/O 制御などのシステム・ソフトウェアにもかなり使われている．

2. ウインドウ・システムの機能

前章でウインドウ・システムと呼ばれているソフトウェアが，Lisp マシンのユーザ・インタフェースをサポートしていることを述べた．本章では，まずウインドウとは何かについて述べ，ウインドウ・システムの三つの大きな役割について解説する．また，メニュー形式のユーザ・インタフェースをサポートする機能について概略を述べる[10].

2.1 ウインドウとは何か

Lisp マシンのウインドウは，コンソールのスクリーンの上の矩形領域である．スクリーン上には，同時に複数個のウインドウが存在することができる．これをマルチ・ウインドウと呼ぶ．多くの場合これらのウインドウは重なり合っていて，ちょうど机の上に紙を重ねて置いた状態に似ている．これをオーバーラッピング・ウインドウと呼ぶ(図1)**.

* 現在，Lisp の標準として CommonLisp[7] が有力であるが，その中にオブジェクト指向プログラミングの機能を導入しようという動きがある．その仕様としては，フレーバではなく Common Loops[8] というシステムの仕様が採用される可能性が強い．CommonLoops は Xerox の提案で，機能としてはフレーバの機能を包含しているが，上位互換性を意味しているわけではない模様である．Lisp マシンは，すでに CommonLisp そのものへの対応を完了しているが，オブジェクト指向プログラミングの機能については，ウインドウ・システム等の既存のソフトウェアの扱いも含めて何等かの対応を迫られることになろう．

** これに対して，スクリーン上をお互いに重なり合うことなく分割して，ウインドウとする方法をタイリング・ウインドウ (tiling window) と呼ぶ．

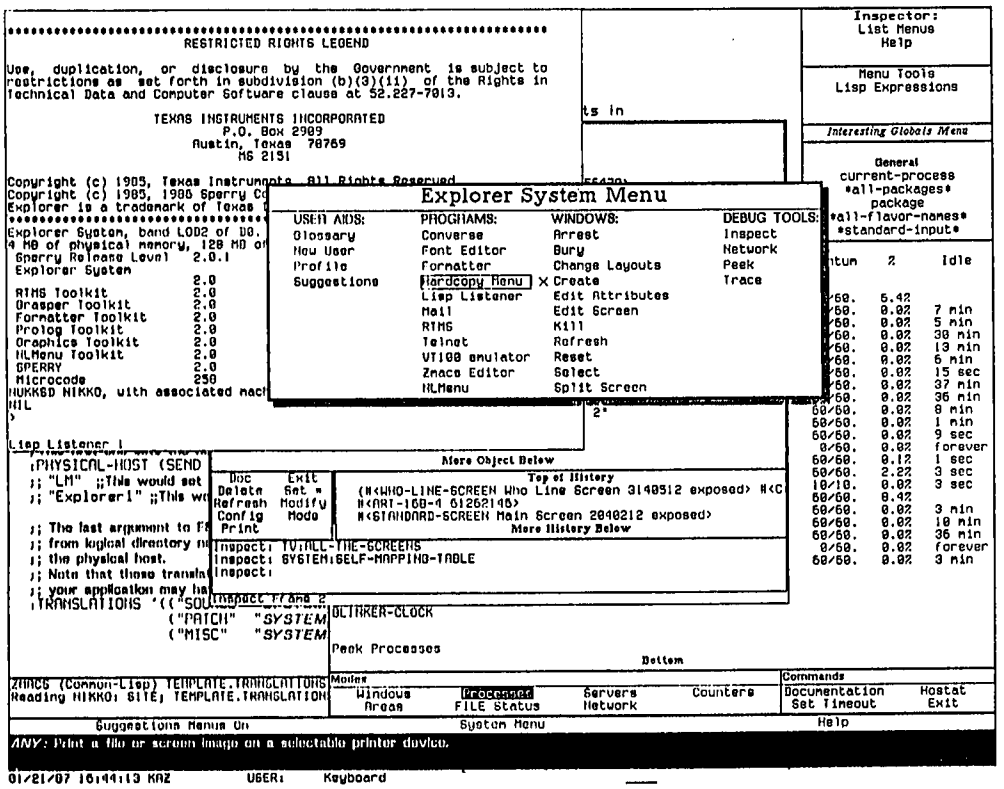


図 1 オーバーラッピング・ウインドウ

Fig. 1 Overlapping windows

これらのウインドウはスクリーン上を移動したり，あるいは領域の大きさを変えることができる．また，スクリーン上で重なり合った状態をコントロールすることができる．すなわち，ウインドウは，①重なりの下の方で他のウインドウに完全に覆われている状態，②他のウインドウと重なっているが一部が見えている状態，③あるいは重なりの一番上にあって領域全体が見えている状態，をとることができる．

またウインドウは，それぞれの領域ごとに独立に図形や文字を描くことができ，キーボードあるいはマウスからの入力を受け取ることができる．したがって，それぞれ仮想コンソールを表していると考えることができ，システム内の複数のプロセスがウインドウをそれぞれ専用の仮想コンソールとして扱うことができる．

以上の基本的な機能の上に，いくつかの高いレベルの機能が用意されている．たとえば表示されている文字や図形にマウスが近づくと，その周囲に枠が現れて感応する機能がある．また，ウインドウ内の表示をマウスを用いてスクロールする機能がある．このほか，複数のウインドウを結合したフレームと呼ばれるウインドウがあり，(この場合，フレームを構成する各ウインドウをペイン (Pane) と呼ぶ) コントロール・パネルのような仮想コンソールを使うことができる (図2)．

2.2 ウインドウ・システムの役割

ウインドウ・システムの基本的な役割は，大きく次の三つである．一つはコンソールの制御である．これによってコンソールのハードウェアを物理的に制御し，何らかの入出力を行うことである．コンソールにはキーボード，ディスプレイ，マウスなどのデバイスが装着されている．これらのデバイスは各種のプログラムから使用されるが，それらのプログラムとのインタフェースの役割を果たす．

二つめはプロセスとユーザのコミュニケーションを管理することである．システムの内部では同時に複数のプロセスが存在し，各々が対応するウインドウを持つことができる．しかし，キーボード，マウスといった入力のためのハードウェア資源は一つづつであり，ある時刻にユーザはただ一つのプロセスと情報のやり取りを行うことができる．言い方を換えれば，それぞれ一つづつしかないコンソールのハードウェア資源を，どのプロセスに対応させるかを

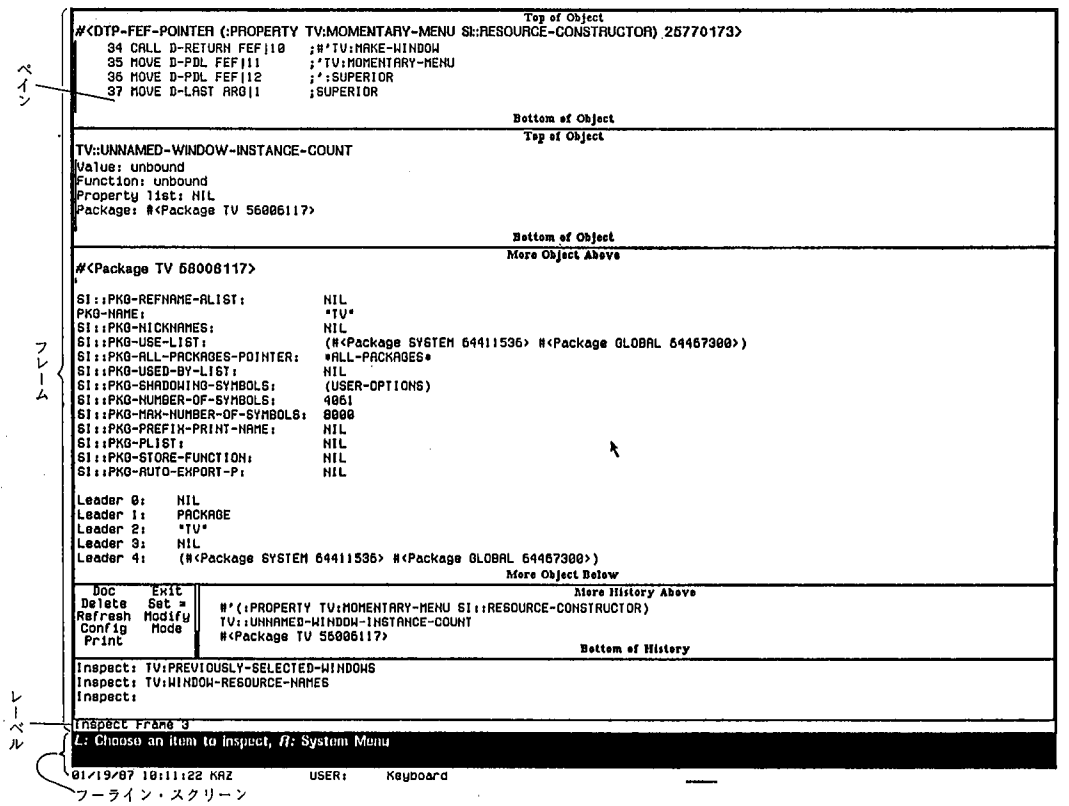


図 2 フ レ ー ム
Fig. 2 Frame

管理することである．これはユーザの要求によることもあり，またプログラムから制御されることもある．

もう一つがウインドウの挙動を制御することである．プログラムは，ウインドウを仮想コンソールとみなして各種の入出力操作をすることができる．さらに，高度の操作性を提供する各種の機能がある．その中には，ウインドウのスクロールやマウスによって表示されたものを指示する機能などが含まれている．

2.3 チョイス・ファシリティ

通常のウインドウは，一般の文字と図形の入出力を目的としているが，限定された入出力を目的としたウインドウがある．これらのウインドウを総称してチョイス・ファシリティと呼んでいる．

これは，ユーザの入力を支援することを目的としており，入力の選択肢あるいは型（具体的には構文上および意味上の検査）を定義し，正しくない入力を排除しようとするものである．これは Lisp マシン・システムのユーザ・インタフェースで大きな役割を果たしている．

この機能は，ウインドウ・システムの基本的な機能の応用例である．実際の例は 4 章で紹介する．

3. ウインドウ・システムの実現

前章で述べたウインドウ・システムの機能について，その実現の概略を述べる．

3.1 ウインドウ・システムの構成

ウインドウ・システムは大きく分けて，次の二つの部分から構成される．一つはいくつかの制御プロセスで，もう一つは各ウインドウのフレーバ・インスタンスである．制御プロセスはコンソールの各デバイスの制御，およびスクリーン・マネージャと呼ばれる，スクリーン上の領域を管理するプロセスである．各ウインドウのフレーバ・インスタンスは，ウインドウに必要なデータ，およびウインドウに対する操作を実行するメソッドを持っている．

3.2 ウインドウの性質

ウインドウ・フレーバは，ウインドウの基本的な性質を持ついくつかのフレーバと，そのほかの付随的な性質を持つフレーバを組み合わせて作られる．この組み合わせを変えることで，希望する性質を持つウインドウ・フレーバを作ることができる．以下

の節でウインドウの基本的な性質について述べる．

3.2.1 スクリーン上の矩形領域の管理

スクリーンに現れるイメージは，ピクセルと呼ばれる点から構成されている．ピクセルは明るさと色の情報を持っており，特殊なメモリ（物理スクリーン配列）の内容と対応している．通常の白黒スクリーンではピクセルの色は黒に固定され，明るさは明暗の2段階であるので，1ビットで1ピクセルを表すことができる．このことから，この種のスクリーン・デバイスをビットマップ・ディスプレイと呼んでいる．ウインドウは，このビットマップの一部を占有することによって実際に見える状態になる．スクリーンもウインドウの一種と見なされており，ウインドウはスクリーンを頂点とする階層構造で管理されている．ウインドウのインスタンスのうち，この階層構造に含まれているものはアクティブなウインドウと呼ばれる（図3）．各ウインドウは，そのイメージをスクリーン配列（screen array）と呼ばれる配列に保持している．階層構造の頂点にあるスクリーンは，物理スクリーン配列をスクリーン配列として持っている．

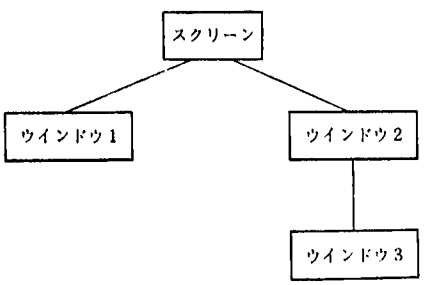


図3 ウインドウの階層
Fig. 3 Window hierarchy

スクリーン配列は，次の三つの状態をとる．

1) スクリーン配列は存在しない．
2) 親のウインドウのスクリーン配列の一部を共有している．
3) 自分で配列を持っている．

スクリーンの直接の子供のウインドウを考えると，2）の状態ではスクリーン配列は物理スクリーン配列の一部を共有することになり，その内容はディスプレイ上に実際に表示される．この状態をエクスポーズされた状態という．3）の状態ではウインドウの内容は保存されてはいるが，実際に表示はされていない．この配列をとくにビット保存配列（bit-save array）と呼ぶ．1）の状態ではウインドウの内容は失われる．

ウインドウが，スクリーンの直接の子供でない場合も，スクリーンに至るすべての親ウインドウが，2）の状態であればその内容はディスプレイ上に実際に表示される．このとき，親のスクリーン配列はビット保存配列である場合もあり，この場合は，エクスポーズされていても内容がスクリーン上に表示されるとは限らない（図4）．

3.2.2 イメージの出力

ウインドウに対するイメージの出力は，ウインド

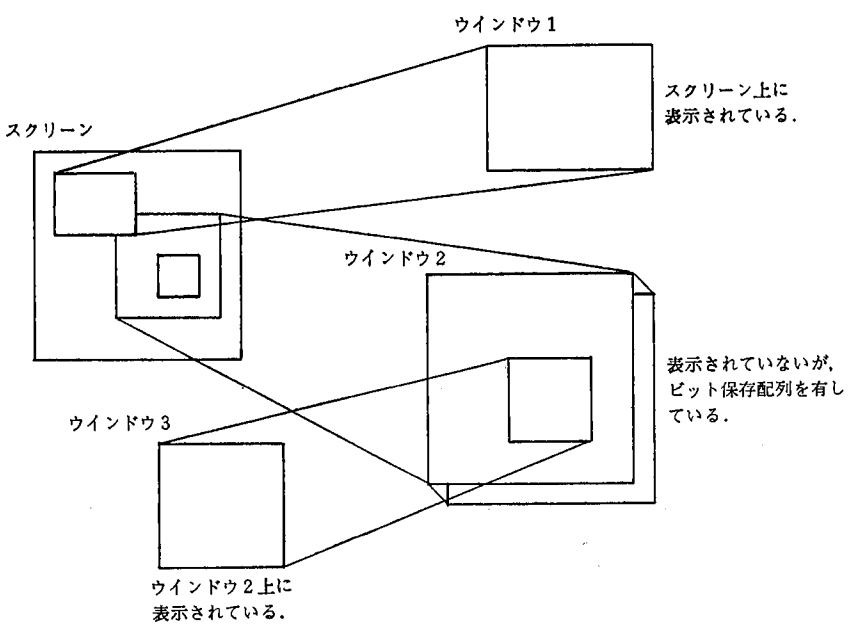


図4 スクリーン配列とウインドウ表示との関係
Fig. 4 Relation between screen array and window exposure

ウのスクリーン配列にピクセルでイメージを描くことによって行われる．したがって，ウインドウのスクリーン配列が存在しない場合は出力することができない．

文字を出力する場合，文字コードで対応するピクセル・イメージの表を引いて，そのイメージをスクリーン配列上に置く．この表をフォントと呼び，これを複数用意して切り換えることによって多様な字体と大きさの文字を，一つのウインドウ内で用いることができる．

図形を出力するのは文字より単純であり，図形をスクリーン配列上にピクセル・イメージとして直接描いてやればよい．

3.2.3 マージン（ウインドウの飾り）

スクリーン上では，複数のウインドウが重なっている場合があり，領域の境界をはっきりさせることが望ましい．このとき，ウインドウの周囲にマージンと呼ばれる飾りの領域を取ることができる．このマージンの中には，ウインドウの枠を示すボーダー，ウインドウのレーベル，さらにウインドウの内容をスクロールするためのスクロール・バーなどが表示される．これらもウインドウのスクリーン配列上にピクセル・イメージとして描かれる．これらのボーダー，レーベル，あるいはマージン内の領域を制御するミキシン・フレーバがそれぞれ用意されている．マージンの内容はウインドウに対するテキストや図形の出力と独立に制御される．

ウインドウの矩形領域からマージンの領域を除いた部分をインサイドと呼ぶ．通常のテキストや図形の出力は，この領域に対して行われる（図5）．

3.2.4 ウインドウの座標系

ウインドウは，それぞれ二つの座標系を持っていて，ウインドウに対する操作はこのいずれかの座標系を用いて行われる．一つはウインドウの矩形領域全体を含む座標系で，矩形領域の左上隅を（0，0）とし，右方向に x，下方向に y をとる．これをウインドウの外座標系と呼ぶ．マージン領域に対する操作や子供ウインドウの相対位置などは，この座標系で表される．もう一つの座標系は，インサイド領域の左上隅を（0，0）として，右方向に x，下方向に y を取る．これは内座標系と呼ばれ，図形の出力の際にはこの座標系が用いられる（図5）．またテキストの出力の際に用いられるカーソルの位置は，この座標系で x と y を文字と行単位に換算して表される．このとき，文字の幅と行の高さはウインドウで用いられるフォントの集合から決められる．

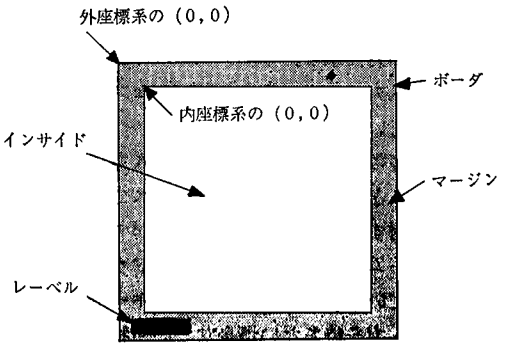


図5 マージンとインサイド
Fig. 5 Window margin and inside

3.2.5 入力バッファ

ウインドウは，多くの場合入力バッファを持っており，キーボードからの入力を受け取ることができる．ある時刻におけるキーボードからの入力は，ある一つのウインドウの入力バッファにはいる．このウインドウを，セレクト（select）されたウインドウと呼ぶ．どのウインドウをセレクトするかはユーザが要求することもでき，プログラムから制御することもできる．セレクトされていないウインドウを使っているプロセスは，入力要求を出しても待状態になる．

プログラムからウインドウの入力バッファにデータを強制的に挿入することもできる．たとえば，マウスのクリックがあった時に，その情報は通常ウインドウの入力バッファに挿入される．このときマ

Explorer System Menu

USER AIDS:	PROGRAMS:	WINDOWS:	DEBUG TOOLS:
Glossary	Converse	Arrest	Debugger
New User	Font Editor	Bury	Inspect
Profile	Formatter	Change Layouts	Network
Suggestions	Hardcopy Menu	Create	Peek
	Lisp Listener	Edit Attributes	Trace
	Mail	Edit Screen	
	RTMS	Kill	
	Telnet	Refresh	
	VT100 emulator	Reset	
	Zmacs Editor	Select	
	NLMenu	Split Screen	

図6 メニューの例
Fig. 6 An example menu

ウスのデバイスを制御するプロセスと，ウインドウに対して入力要求を出したプロセスは別のプロセスであり，入力バッファを介したプロセス同士の通信を利用している．

また，複数のウインドウで一つの入力バッファを共有することができ，一つのプロセスで複数のウインドウに対する入力を監視することができる．とくに，フレームを用いて複数のウインドウを組み合わせて用いている場合に，メニューとキーボードのどちらかの入力を待ちたい場合などに便利である．

3.2.6 マウス・ハンドラ・メソッド

マウスを制御するプロセスは，ウインドウのいくつかのメソッドを呼ぶ．このメソッドではマウスの位置やクリックの有無等を知ることができる．これを用いてマウスがそばにくると，それに感応する表示を実現することができる．これはメニューなどで広く用いられている．また，スクリーンの下部にはフーライン・ドキュメンテーション・ライン (Wholine Documentation Line) と呼ばれる領域があり，マウスの動きに応じて説明を表示させることができる（図2）．

また，マウスを用いてウインドウの内容をスクロールさせるための標準的な操作手順が用意されている．これには，スクロール・バーを用いる方法，マウスをウインドウの辺へ持ってゆくことでスクロールさせる方法などがある．

3.2.7 ブリンカ

カーソルやマウスの位置を示すためのスクリーン上の表示を総称してブリンカと呼ぶ．これはカーソル，マウスの動きに追従して表示される．また点滅させることができ，このことからブリンカと呼ばれる．ブリンカとしては各種の図形や文字を使うことができる．マウスに感応して表示される枠や，対応する括弧が点滅する等の機能はすべてブリンカを用いている．

3.3 スクリーン・マネージャ

スクリーン・マネージャの仕事は，次の二つである．

ウインドウは，スクリーン上にちょうど紙が積み重ねられているように見える．このとき，一番上にあるウインドウが消滅したり，下の方へ移動した場合，新しく一番上になったウインドウを完全に見える状態にしてやる必要がある．これを自動エクスポーズ (auto exposure) と呼び，スクリーン・マネージャが他のウインドウに覆われていないウインドウを捜してエクスポーズする．

もう一つは，部分的に他のウインドウに覆われているウインドウの，見える部分の内容の面倒を見ることである．このようなウインドウは，親ウインドウのスクリーン配列の対応する領域の一部を，覆っているウインドウに占有されているため，エクスポーズされた状態ではない．したがって，ウインドウ自身はイメージを描くことができず，スクリーン・マネージャが物理スクリーン配列の内容やビット保存配列がもしあれば，その内容からイメージを復元する．

4. ウインドウ・システムのアプリケーション

この章では，ウインドウ・システムのアプリケーションの例として，システムの一部に組み込まれ提供されているいくつかのソフトウェアを紹介する．

4.1 チョイス・ファシリティ

この機能は，限定された入力に対してより高い操作性を得ることを目的としている．システムの各種のプログラムのユーザ・インタフェースの一環として用いられている．

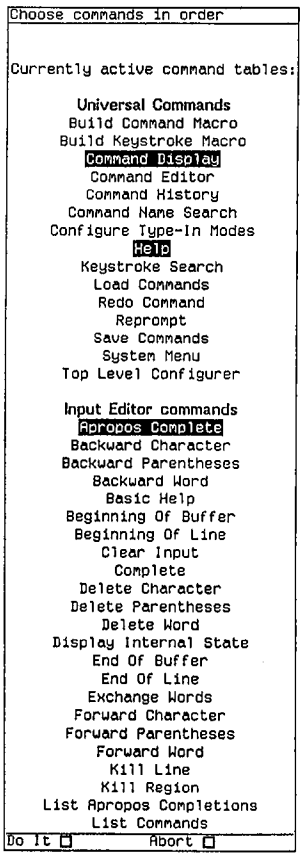
1) メニュー……ウインドウの一種で，複数の選択肢の中から選択させることを目的としている．ただ一つを選択する通常のメニューと，複数選択することが可能なマルチプル・メニューがある．メニューのウインドウを作るためには，各々の選択肢として表示される文字列，選択された場合に返される値，ドキュメンテーションとして用いられる文字列などを与えればよい（図7）．

2) マルチプル・チョイス……いくつかの項目について，同じ選択肢からの選択を適用させたい場合に用いる．項目の集合，およびメニューの場合と同様の選択肢を定義して用いる．選択肢同士の間の依存関係を定義することもできる（図8）．

3) チューズ・バリアブル・バリューズ (Choose-Variable-Values)……いくつかの項目について，それぞれ異なった種類の入力をさせたい場合に用いる．各項目について名前，入力の種類(型)ドキュメンテーション等を定義する．入力の種類としては，選択肢からの選択の外に，数値，パス名，フォント名などシステムで使われている情報に限定した入力も可能である（図9）．

4.2 コントロール・パネル型インタフェース

いくつかのシステム・ユーティリティが，複数のウインドウを組み合わせたフレームを用いて，コントロール・パネルのようなインタフェースを提供している．



(←) 図 7 マルチプル・メニュー
Fig. 7 Multiple menu

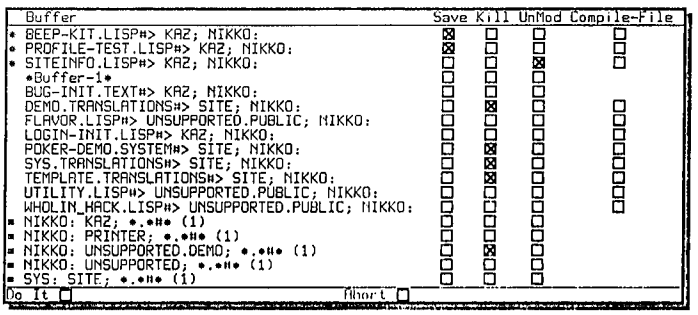


図 8 マルチプル・チョイス
Fig. 8 Multiple choice

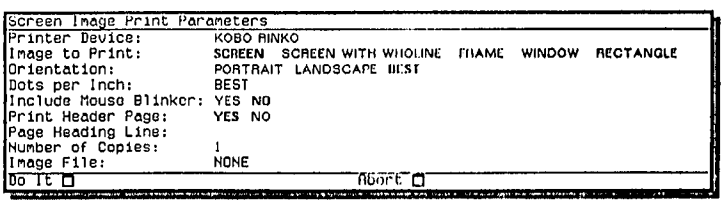
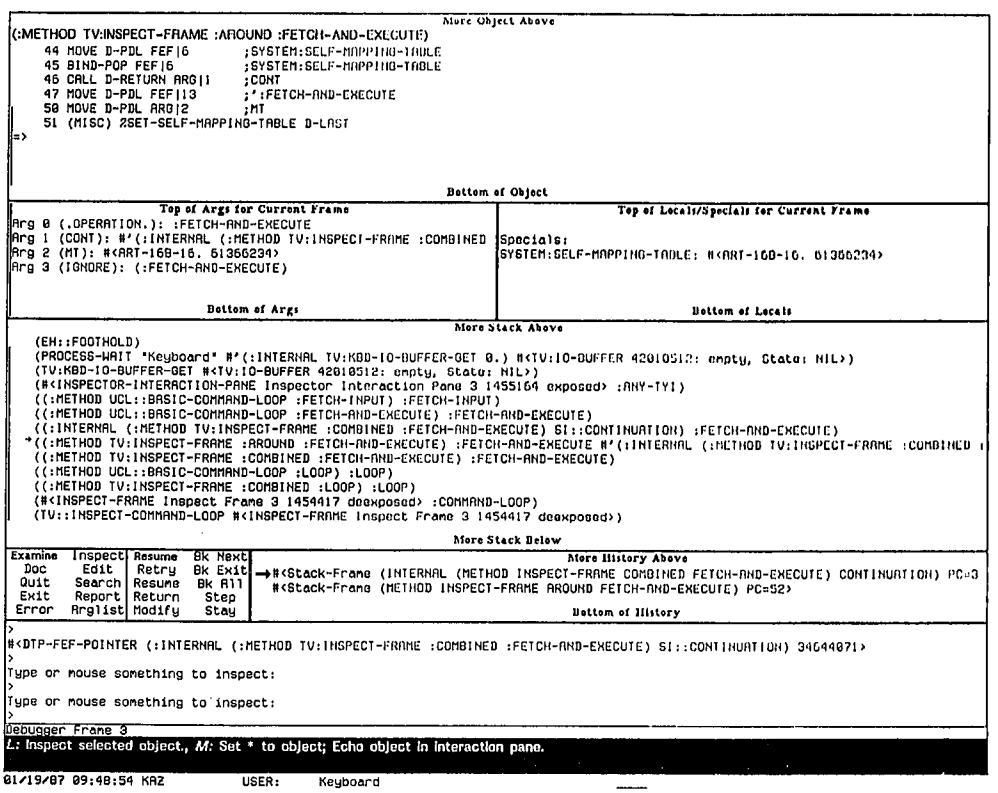


図 9 チューズ・バリアブル・バリューズ
Fig. 9 Choose variable value

(↓) 図 10 デバッガ
Fig. 10 Debugger



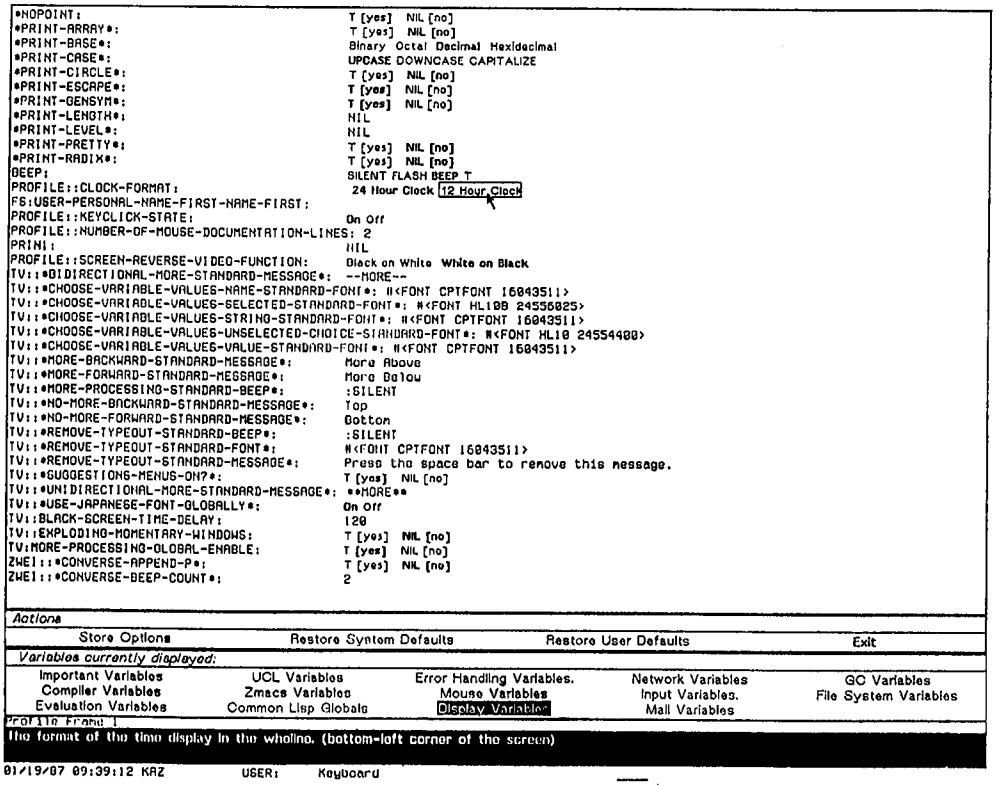


図 11 プロファイル
Fig. 11 Profile

1) デバッガ……プログラムの実行中の状態を視覚的にわかりやすく表示し，さらにその状態を変更することができる．実行中の関数の機械語命令，関数呼び出しの履歴を表すスタックの内容，呼び出しの際の引き数，ローカル変数の内容などが表示される．とくに，引き数とローカル変数の内容は変更することができ，スタックの中で呼び出しを遡って再試行することができる（図 10）．

2) プロファイル……システムのパラメタを管理するためのユーティリティ．各パラメタは Lisp のグローバル変数として定義されており，その値をチューズ・バリアブル・バリューズと同じようにして変更することができる．これらの変数はいくつかのカテゴリに分けられており，そのカテゴリの切り換えや設定されたパラメタの保存などの機能をメニューから指示することができる（図 11）．

5. おわりに

現在，Lisp マシンは人工知能専用と考えられているが，本稿で紹介したウインドウ・システムは，人工知能以外でも，高度の操作性を要求されるアプリケーションに適用することができると考えられる．たとえば，試行錯誤を要求される設計，計画業務，あるいはオフィス・オートメーションの中で，従来の紙と鉛筆に変わる媒体として操作性の高い小型計算機の要求が存在している．ウインドウ・システムは，このような要求に答えるソフトウェアの一つであるといえる．

参考文献

[1] A. Goldberg, Smalltalk-80——The Interactive Programming Environment, Addison Wesley, 1984.
[2] C. Kaehler, et al., Macintosh Plus, Apple Computer, Inc., 1986.（日本語版 アップル・コンピュータ・ジャパン, 1986）
[3] R. D. Greenblatt, et al., "The LISP Machine", Interactive Programming Environments, D. R. Barstow, et al., eds, McGraw-Hill, 1984.
[4] R. W. Scheifler, J. Gettys, The X Window System, MIT, 1986.
[5] H. Cannon, Flavors——A Non-hierarchical approach to Object-oriented programming,

MIT, 1982.
[6] D. A. Moon, Object-Oriented Programming with Flavors, ACM OOPSLA Conference, 1986.
[7] G. L. Steele, Common Lisp the Language, Digital Press, 1984.
[8] D.G. Bobrow, et al., Common Loops-Merging Lisp and Object-Oriented Programming, ACM OOPSLA Conference, 1986.
[9] A. Goldberg, Smalltalk-80-The Language and Its Implementation, Addison Wesley, 1983.
[10] Explorer Window System Reference, Texas Instruments Incorporated, 1985.

(知識システム開発部)

MHS メッセージ通信システム

An Orientation for Message Handling Systems

佐 藤 茂 夫
S. Sato

1. はじめに

オフィス・オートメーション(OA)の普及に伴い，オフィス作業の大きな部分を占める文書の配信・管理のため，各メーカから種々の「電子メール・システム」が提供されている．これらの電子メール・システムの利用が広まるにつれて，システム同士の相互接続による広域の電子メール・システムへの要求が起こってきた．しかしながら，既存の電子メール・システムは各社が独自の方式に基づいて開発したもので，相互に接続するのは不可能である．

このような背景から CCITT（国際電信電話諮問委員会）は，1984年に電子メール・システムを相互に接続するための標準として X. 400 シリーズ勧告“メッセージ通信システム” MHS (Message Handling Systems)[1] を発表した．なお，X. 400 シリーズ勧告は，表1で示す八つの勧告から成り立っている．

表 1 CCITT X. 400 シリーズ勧告
Table 1 Recommendations of CCITT X.400 series

勧告番号	勧告名
X.400	MHS システム・モデルとサービス要素
X.401	MHS 基本サービス要素と付加ユーザ機能
X.408	MHS 符号化情報タイプの変換規則
X.409	MHS プレゼンテーション転送構文と記述法
X.410	MHS リモート・オペレーションと高信頼性転送サーバ
X.411	MHS メッセージ転送レイヤ
X.420	MHS 個人間メッセージ通信ユーザ・エージェント・レイヤ
X.430	MHS テレテックス端末のためのアクセス・プロトコル

本稿は，個人間メッセージ通信（IPM：Interpersonal Messaging）サービス（勧告 X. 420）の利用を前提とした MHS の技術内容と標準化の動向について解説する．

2. MHS のモデル

MHS は OSI（Open Systems Interconnection：開放型システム間相互接続）の概念に従ったアプリケーションであり，以下のようにメッセージ通信のためのモデルを規定している．

2.1 MHS のネットワーク・モデル

MHS のネットワークは図1に示すように，メッセージ転送エージェント（MTA：Message Transfer Agent）とユーザ・エージェント（UA：User Agent）から構成される．

図1のユーザは，端末の前に座った人間やアプリケーション・プログラムなどであり，メッセージの発信者または受信者である．

メッセージ転送システム（MTS：Message Transfer System）は，メッセージの転送（システム間の移送）を受け持つ部分であり，一つまたは複数の MTA から構成される．MTA は互いに協力してメッセージの転送や中継，さらには宛先の UA への配信などを行う．

UA は，ユーザに代わってメッセージ転送システムにより提供されるメッセージ転送サービス（MT サービス）を利用して，ユーザ間のメッセージの中身にかかわる処理を行う．UA は，コンピュータ・アプリケーション・プロセスであり，他の UA と共同してメッセージ通信サービスを提供する．実装に当たっては，UA にはユーザ（人間）がメッセージを作り上げるための編集機能や，メッセージを保存するためのメール・ボックス機能などが必要となるが，MHS 勧告では，これらはローカルな問題としており，規定していない．

1984 年に勧告された MHS では，メッセージ通信の形態として個人と個人の間でのメッセージの送受信を扱った個人間メッセージ通信サービスが規定されている．このサービスを実現する UA を，個人間メッセージ通信ユーザ・エージェント（IPM UA）と呼ぶ．

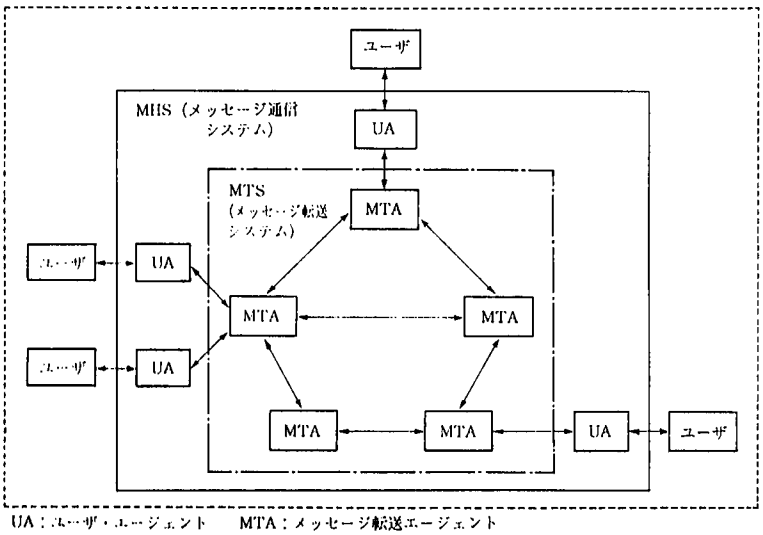


図 1 MHS のネットワーク・モデル
Fig. 1 MHS network model

2.2 メッセージの構造

MHS で扱われるメッセージは，図2に示すとおり，エンベロープとコンテントからなる．

エンベロープはメッセージの転送と中継に必要な制御情報を含んでいて，メッセージ転送システムにより作られ，また解釈される．

コンテントは UA により作られる．個人間メッセージ通信で使われるコンテントの構成は，図3に示されるように通常のビジネス文書に準じた形式であり，ヘディングとボディからなる．ヘディング部分には，その文書の性格・属性に関する情報（たとえば，宛先，写，表題，機密度など）が含まれる．

ボディ部分には，ユーザが送信する文書の本文に相当する情報が入る．本文（ボディ）には，文章（文字データ）の他，図形（ファクシミリ，ビデオテックスなど）や音声などのさまざまな表現形式の情報を混在させることができる．

2.3 MHS のシステム・モデル

MHS は，そのサービスを実現するために固有のプロトコルを定めている．このプロトコルを実行する主体は，図4で示すとおり，OSI 基本参照モデル[2]の応用層に位置づけられている．

MHS の応用層は図5で示すとおり，メッセージ転送サービスを実現する MTA エンティティ*と，個人間メッセージ通信サービスを実現する UA エンティティとの二つの副層から構成される．

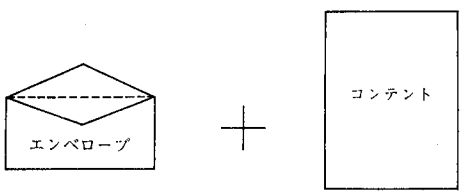


図 2 メッセージの構造
Fig. 2 Message structure

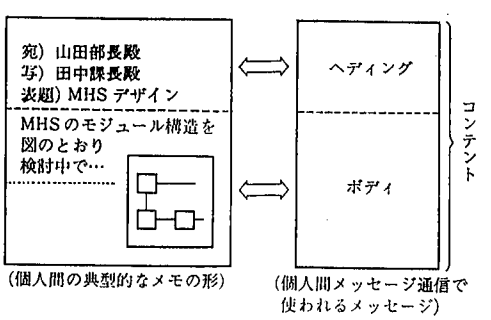


図 3 ビジネス・メモと個人間メッセージ通信のメッセージとの関係
Fig. 3 Relationship between a memo and an IP-message

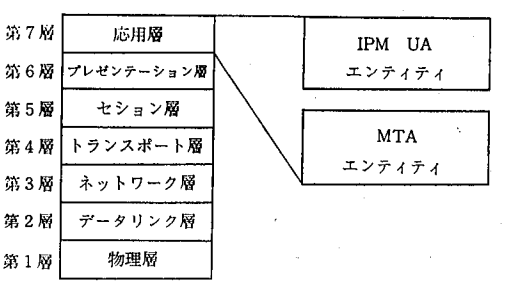


図 4 MHS と OSI 7 階層モデルの関係
Fig. 4 OSI 7 layers model and MHS

* エンティティとは，OSI のある要素の実体を呼ぶ．

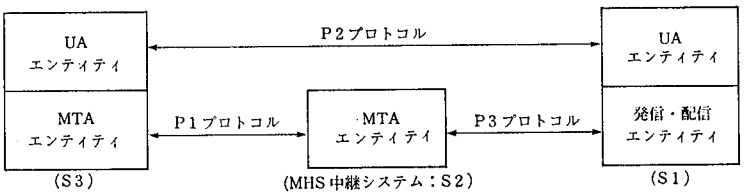


図 5 MHS 応用層の階層
Fig. 5 Layerd structure of MHS application

MHS の各々のシステムは，実装のレベルにより三つの型に分類される．UA の機能だけを持つシステムを S1，MTA の機能だけを持つ中継システムを S2，そして UA と MTA の両方の機能を持つシステムを S3 と呼ぶ．

2.4 OSI の他階層との関係

OSI の応用層に位置づけられる MHS は，下位層の利用方法についても規定している．

プレゼンテーション層との関係では，X.409 により MHS の応用層のデータ（プロトコル・データ・ユニット）の記述形式とコーディング方式の規則を定めている．セション層については，メッセージを確実に転送するために，アクティビティの機能単位の利用を定めている（X.410）．

また，トランスポート層についても利用するクラス等を定めている．

3. メッセージ転送サービスとプロトコル

3.1 メッセージ転送サービス

MTA エンティティが，その上位副層である UA エンティティに提供するサービスをメッセージ転送サービスと呼ぶ．勧告 X.400 では表 2 に示すメッセージ転送サービスの種類を規定している．メッセージ転送サービスは，メッセージの転送に当たっての優先度や配信時刻の指定といった転送の方法に関する各種のサービスを提供する．

表 2 メッセージ転送サービス要素
Table 2 Message transfer service elements

サービス要素	分類
① 基本メッセージ転送サービス要素	
アクセス管理 (Access management)	基本
コンテント・タイプ通知 (Content type indication)	基本
メディア変換表示 (Converted indication)	基本
配送時刻通知 (Delivery time stamp indication)	基本
メッセージ識別 (Message identification)	基本
不達通知 (Non-delivery notification)	基本
メッセージのメディア通知 (Original encoded information types indication)	基本
受信可メディア登録 (Registered encoded information types)	基本
送信時刻通知 (Submission time stamp indication)	基本
② 発信配送サービス要素	
代行受信者への配送許可 (Alternate recipient allowed)	必須
時刻指定配送 (Deferred delivery)	必須
時刻指定配送中止 (Deferred delivery cancellation)	必須
送達通知 (Delivery notification)	必須
同報全宛先通知 (Disclosure of other recipients)	必須
優先度指定 (Grade of delivery selection)	必須
同報 (Multi-destinction delivery)	必須
不達通信の抑止 (Prevention of non-delivery notification)	付加
コンテントの返送 (Return of contents)	付加
③ 変換サービス要素	
メディア変換禁止 (Conversion prohibition)	必須
メディア変換指定 (Explicit conversion)	付加
自動メディア変換 (Implicit conversion)	付加
④ 問い合わせサービス要素	
着信端末機能問い合わせ (Probe)	必須
⑤ 状態通知サービス要素	
代行受信者の登録 (Alternate recipient assignment)	付加
メッセージ一時預かり (Hold for delivery)	付加

表 3 メッセージ転送層サービス・プリミティブ
Table 3 Message transfer layer service primitives

サービス・プリミティブ	機能
LOGON	UA と MTA との間の MT サービスの開始を要求する.
LOGOFF	MT サービスの終了を要求する.
REGISTER	MTA エンティティが保持している UA の属性 (例: UA が扱えるコンテントの最大長) を変更する.
CONTROL	MTA エンティティが保持している UA の属性 (例: UA が扱えるコンテントの最大長) を一時的に変更する.
SUBMIT	UA エンティティが MTA にメッセージの転送を要求する. このとき, 受信者の名前, 優先度, 配信通知の要求などの情報も付けて MTA に要求する.
DELIVER	MTA が宛先の UA エンティティへメッセージを配信する. このとき, 発信者名, 他の受信者名, 配信の時刻などの情報を付けて MTA に要求する.
PROBE	UA エンティティは実際のメッセージの発信に先立ち, 受信者側の UA エンティティの受信能力 (コンテントの種類, 長さ等) を事前に問い合わせる.
CANCEL	時刻指定付きで, すでに発信したメッセージの転送・配信を取り消す.
NOTIFY	先に発信したメッセージの配信の結果, (正しく届いたか, 届かなかったか) や PROBE の結果を UA エンティティへ通知する.
CHANGE PASSWORD	パスワードの変更を行う.

上位の副層の UA は, 表3で示されるメッセージ転送サービス・プリミティブによって, MTA エンティティへ要求等を行う.

3.2 メッセージ転送プロトコル (P1 プロトコル)

メッセージ転送サービスのうち, 多くは, そのメッセージの転送にかかわる複数の MTA の協力によって実現される. こうした共同作業を可能にするために, MTA エンティティ間に規約, すなわちメッセージ転送プロトコルが規定されていて, これを P1 プロトコルと呼ぶ.

MTA エンティティは, 図6に示すように論理的に三つの要素から構成される. アソシエーション管理者の役割は, メッセージを転送するための隣接する二つの MHS システム間に確立させるアソシエーション* の確立・維持・解放の管理を行うことである.

メッセージ・ディスパッチャの役割は, UA エンティティから依頼された発信メッセージや, 他の MTA から到着したメッセージに基づいて, P1 プロトコルを実行することである.

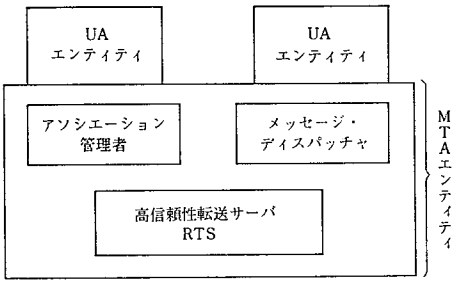


図 6 MTA エンティティのモデル
Fig. 6 Model of an MTA entity

高信頼性転送サーバ (RTS: Reliable Transfer Server) の役割は, セション層のサービスを利用してメッセージを隣接する MTA に確実に転送することである.

P1 プロトコルにより, 隣接 MTA 間で転送される情報をメッセージ・プロトコル・データ単位 (MPDU: Message Protocol Data Unit) と呼ぶ. MPDU には, ユーザ・メッセージ・プロトコル・データ単位 (UMPDU) とサービス・メッセージ・プロトコル・データ単位 (SMPDU) の2種類がある.

UMPDU は, UA エンティティが依頼したメッセージを運ぶためのものであり, エンベロープとコンテントからなる (図2). エンベロープには P1 プロトコルのための制御情報が入り, コンテントには個人間メッセージ通信から渡された情報が入る.

SMPDU は, 送達通知のような MTA 間の状況の伝達に使われる.

3.3 メッセージ・ディスパッチャの機能

メッセージ・ディスパッチャは P1 プロトコルの実行とともに, メッセージの中継, 送達・不達の通知, そしてコンテントの変換を行う.

1) メッセージの中継……複数の宛先へメッセージを送る (同報) 場合には, 複数の MTA エンティティがかかわることになる. 受信 UA エンティティがいくつかの異なる MTA エンティティに属する場合には, そのメッセージはいくつかの異なる経路をたどる. 発信 UA エンティティから発信メッセージを受け取った MTA エンティティは, 各受信者について最適な経路, すなわち, 隣接する最適な MTA エンティティを選択する. 選ばれた隣接する MTA エンティティが複数の場合は, メッセージ (UMPDU) のコピーを作り, それぞれの MTA エンティティに向けて転送する. それを受け取った MTA エンティティは, 受信者が自身のローカルな

* OSI のアプリケーション層のエンティティ (アプリケーション・エンティティ) 間の関係

UA エンティティであるなら，DELIVER サービスによってその UA エンティティにメッセージを渡し，そうでないならば同様にしてメッセージのコピーを作り，隣接する最適な MTA エンティティへ転送，すなわち中継する．

2) 送達・不達通知……MTA 層は，転送を依頼されたメッセージを受信 UA エンティティに正しく渡すことができたか否かを，発信UAエンティティに知らせる．これを送達・不達通知と呼び，SMPDU の一種である送達通知メッセージ・プロトコル・データ単位をメッセージ・ディスパッチャ間で転送することにより実現される．

受信MTAエンティティがメッセージを受信UAエンティティに渡すことができたとき，その MTA エンティティは発信元の MTA エンティティへ送達通知を返送する．また，受信側のMTAエンティティがメッセージを受信UAエンティティに渡すことができなかったとき，あるいは，中継 MTA エンティティが次の MTA エンティティへメッセージを転送することができなかったとき，そのMTAエンティティは発信元のMTAエンティティに不達通知を返送する．

発信元のMTAエンティティは，送達あるいは不達通知を受け取ると，発信 UA エンティティへそれを伝える．

3) コンテントの変換……MHS では異なるメディア（ここでは文字情報，ファクシミリなどの情報の表現方式をいう）によるエンド・ユーザ間のメッセージ交換を可能にするため，メッセージ・ディスパッチャがメディアの変換を行う．メッセージ・ディスパッチャは，それぞれの UA の扱えるメディアに関する情報を持っていて，発信者が変換を禁止指定していない場合には，受信可能なメディアへと変換を行う．

3.4 発信・配信プロトコル（P3 プロトコル）

UA の機能しか持たない S1 型システムが，メッセージを転送・受信するためにはメッセージ転送層のサービスを利用する必要がある．このため，S1 型システムの中の発信・配信エンティティ（SDE: Submission and Delivery Entity）と MTA エンティティとの間に発信・配信プロトコルが規定されていて，これを P3 プロトコルと呼ぶ．

P3 プロトコルでは，表3に示されたメッセージ転送層サービス・プリミティブと同等の機能を遠隔システムから利用するため，発信オペレーション，配信オペレーション等のオペレーションが規定されている．

4. 個人間メッセージ通信サービスとプロトコル

4.1 個人間メッセージ通信サービス

個人間メッセージ通信サービスはメッセージ転送サービスを利用することにより，個人間で通信し合うメッセージの中身にかかわるサービスを提供する．勧告 X.400 では表3に示す個人間メッセージ通信サービスを規定している．表4に見られるように，個人間メッセージ通信サービスにはメッセージ転送サービスが含まれているが，これはエンド・ユーザは個人間メッセージ通信サービスを介して MHS のサービス（メッセージ転送サービスを含めて）を享受するためである．

4.2 個人間メッセージ通信プロトコル（P2 プロトコル）

4.1 節で示された個人間メッセージ通信サービスは，IPM UA エンティティ間で規定されている個人間メッセージ通信プロトコルにより実現される．個人間メッセージ通信プロトコルを P2 プロトコルと呼び，IPM UA エンティティ間で転送される情報をユーザ・エージェント・プロトコル・データ単位（UAPDU: User Agent Protocol Data Unit）と呼ぶ．

UAPDU には，個人間メッセージ UAPDU (IM-UAPDU)と個人間状態通知UAPDU(SR-UAPDU)の二種類がある．IM-UAPDU は，エンド・ユーザ間で交換するメッセージそのものを運ぶためのものであり，SR-UAPDU は UA エンティティ間の状態通知のための情報を運ぶためのものである．IM-UAPDU と SR-UAPDU は共に，転送に当たっては下位のメッセージ転送層によって実現され，それらはユーザ・メッセージ・プロトコル・データ単位 (UMPDU) のコンテントとして運ばれる（図7）．

IM-UAPDU は図7で示されるとおり，ヘディングとボディの二つから構成される．ヘディングには転送するメッセージの属性についての情報（例：

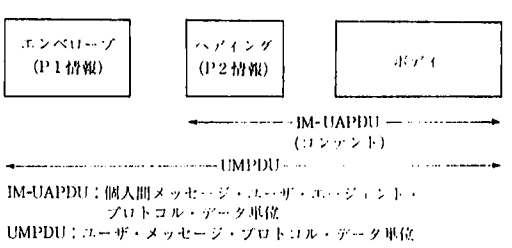


図7 P1 と P2 のプロトコル・データ単位の関係
Fig. 7 Relationship between P1 and P2 protocol data units

表 4 個人間メッセージ通信サービス要素
Table 4 IPM service elements

サービス要素	送信 UA	受信 UA
① 基本個人間メッセージ通信サービス要素		
基本メッセージ転送サービス要素（表2の①に同じ）	（表2参照）	
IP メッセージ識別 (IP-message identification)	基本	基本
メディア通知 (Typed body)	基本	基本
② 発信・配送・変換サービス要素		
表2の②，③に同じ	（表2参照）	
③ ユーザ・エージェント・アクション・サービス要素		
秘密受信者通知 (Blind copy recipient indication)	付加	必須
受信不可通知 (Non-receipt notification)	付加	付加
受信確認通知 (Receipt notification)	付加	付加
自動転送通知 (Auto-forwatded indication)	付加	必須
④ ユーザ・エージェント情報転送サービス要素		
発信者通知 (Originator indication)	必須	必須
発信許可者通知 (Authorizing users indication)	付加	必須
第一受信者・コピー受信者通知 (Primary and copy recipients indication)	必須	必須
有効期限通知 (Expiry date indication)	付加	必須
クロスリファレンス通知 (Cross-referenicing indication)	付加	必須
重要度通知 (Importance indication)	付加	必須
廃棄通知 (Obsoleting indication)	付加	必須
秘密度通知 (Sensitivity indication)	付加	必須
標題通知 (Subject indication)	必須	必須
返信通知 (Replying IP-message indication)	必須	必須
返信要求通知 (Reply request indication)	付加	必須
転送 IP メッセージ通知 (Forwarded IP-message indication)	付加	必須
暗号化通知 (Body part encryption indication)	付加	必須
複合ボディ (Multi-part body)	付加	必須
⑤ 問い合わせサービス要素		
問い合わせ (Query) 表2の④に対応	付加	——
⑥ 状態		
代行受信者の登録 (Alternate recipient assignment) } 表2⑤に対応		
メッセージ一時預り (Hold for delivery) } 表2⑤に対応		

宛先，標題，機密度など）が，また，ボディには，文字データ，ファクシミリ情報などのエンドユーザの送信した情報が入れられる．

5. MHS 標準化動向

5.1 MHS 実装標準化の動き

MHS による相互接続を行う場合，CCITT 勧告は機能範囲が広く，そのすべてを実装することは技術的にもコスト的にも負担が大きすぎる．このため各国では，実装の推進を図るため，使用する機能の範囲，プロトコル・パラメタ値の制約などの詳細な取り決めを行っている．これを実装仕様・実装標準と呼んでいる．

米国では NBS*1 が，ヨーロッパでは CEN/CENELEC*2 が，こうした実装仕様を作っている．日本では TTC*3 が実装仕様を作成し，JUST*4 が勧告を行う．

また，個々の実装された製品の互換性を確認するため，各国では検証システムの提供を計画している．米国では COS*5 が，ヨーロッパでは SPAG* が検証システムの実現を計画している．日本では INTAP*7 が MOTIS（次項で説明）の検証システムの提供を 1988 年に予定している．

5.2 ISO MOTIS との関係

ISO（国際標準化機構）の TC 97/SC 18 では，“文章およびオフィス・システム”について国際標準の作成を行っている．SC 18 は事務文書の表現方法についての標準として ODA[4] (Office Document Architecture) を作成しているが，この事務文書をシステム間で転送する手段として，MOTIS[5] (Message Oriented Text Interchange Systems) を標準草案として規定している．MHS は通信事業者の国際機関である CCITT が勧告したものであ

*1 NBS: National Bureau of Standards, 米国標準局
*2 CEN/CENELEC: European Committee for Standardization/ European Committee for Electrotechnical Standardization, 欧州規格委員会／欧州電子技術規格委員会
*3 TTC: 電信電話技術委員会
*4 JUST: Japanese Unified Standard for Telecommunications
*5 COS: Corporation for Open Systems
*6 SPAG: Standard Promotion and Application Group
*7 INTAP: 情報処理相互運用技術協会

るが，MOTIS はこの MHS をベースに，ISO の構成員である各国のメーカや利用機関の要求を取り入れて拡張したものである．

MOTIS の実装仕様作りは，わが国では INTAP が ODA のそれと同時に行っているが，内容は TTC のものと互換性が保たれるものと予想される．

6. INS プロトコル研究会相互接続実験の概要

NTT が主催する INS プロトコル研究会は，昨年夏，MHS による日本初の相互接続実験[3]を実施した．

同研究会第3専門部会は 1985 年から相互接続を目指し，実験のための MHS の実装仕様の検討を始めた．日本ユニバック，および国産主要メーカ7社と富士ゼロックス（実験参加）9社は，合意された仕様に基づき，それぞれのマシン上に MHS を実装した．

1986 年2月からは，参加各社間でのテストが始まり，同年7月，NTT 横須賀通信研究所において国内関係者および報道関係者に対する公開実験が実施された．そこでは，参加各社間で同報通信や返信などの電子メール特有のデモンストレーションが行われた．

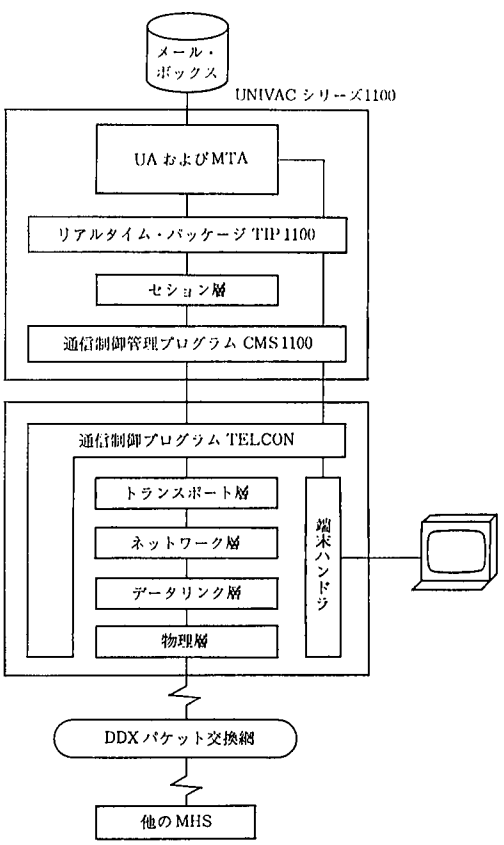


図 8 実験システムのソフトウェア構成図
Fig. 8 Software structure of MHS trial system

日本ユニバックでは，短期間で実験に対応したシステムを構築するため，図8に示すソフトウェア構造を採用し，成功裏に接続実験を終了した．このシステムは実験用に開発したものであり，これ自体を MHS 商品として提供する予定はないが，ここで開発されたトランスポート層，セション層のソフトウェアは，OSI に準拠した他の商品（パソコン通信等）で生かされる予定である．

7. おわりに

MHS は OSI に基づくアプリケーションであることから，国内の各メーカは深い関心を持っており，これから1～2年のうちには実装システムが各社から発表されるものと予想される．

日本ユニバックにおいても，MIIS システム（仮称）の最初のリリースを 1988 年秋に行う予定である．この MIIS システムでは，当社の現在の標準メール・システムである AOA/MAIL 1100 と異機種システムとを接続するためのゲートウエイ機能も用意されることになっている．

また，日本ユニバックでは，IBM 社の SNA* に基づいた文書交換システムに対応したシステムも開発中であり，MHS システムは，これら各種メール・システムとの整合性を維持すると共に，国際標準に基づいたシステムとして提供されることになろう．

参考文献

[1] CCITT Recommendation X. 400-X. 430 Message Handling Systems
[2] ISO, IS 7498, Information processing systems—Open systems interconnection-Basic reference model
[3] 釜江尚彦，“MIIS 電子メール，日本初の相互接続実験に成功するまで”，日経コミュニケーション，1986，10. 20，pp. 115～130
[4] ISO, DIS 8613/1～6, Information Processing—Text and office systems—Office Document Architecture (ODA) and interchange format
[5] ISO, DIS 8505, Information Processing—Text communication—Functional description and service specification for message oriented text interchange systems

（通信システム部）

* SNA: Systems Network Architecture. IBM 社のネットワーク・アーキテクチャ．

石畑 清
疋田輝雄 共著

"Ada プログラミング"

岩波書店，A 5 判，xiv＋288 pp.，1986 年，2,500 円.

Ada の ANSI/MIL 規格が定まってから早5年が経過している．現在，国際的には Ada の ISO 規格化，日本では JIS 規格化の作業も行われている．Ada に関する書物は規格制定以前から数多く出版されてはいるが，それらはいずれも英文のものである．そのうちいくつかは翻訳されているが，日本人の手になるものはほとんどない．本書は日本人によって日本語で書かれた初めての Ada 入門書であり，著者らの Ada 普及の意欲がうかがわれる．

著者の一人である石畑氏は，JIS 規格化作業部会などにも参加し広く Ada にかかわっている，プログラミング言語に深い見識を持つ人である．著者らによれば，本書は「多数のプログラム例を通して，Ada の言語機能とプログラム技法を解説する．まず Ada の全体像をやさしく紹介し，次に基本構文を整理して要領よく解説するとともに，パッケージによる大規模プログラミング，タスクによる実時間プログラミングなどのプログラム技法を説き明かす．Ada の表現力は強力であるが，その基本概念は多くはなく，またむずかしくもない――これは数多くの Ada の実践を通して得た結論である」また「およそプログラミング言語の入門書は，その言語の基本的機能をはっきり示すことを当然の出発点として，さらにそのプログラムの作る可能世界のいくらかを読者に開示してみせなければならないであろう．その言語でどのようなプログラムを作れるかということである」との立場から完結したプログラムを数多く収録している．著者らは Ada を「Pascal＋パッケージ＋タスク」として捉えているが，これはおおむね適切な認識といえるであろう．

本書の全体は次の4部から構成されている．

第1部（1章～4章）……Ada 入門としてプログラム例によりながら Ada の外見を紹介している．

第2部（5章～11章）……基本的な機能としてデータ型，式，文，副プログラムなどが説明される．

第3部（12章～15章）……大規模プログラミングに関連する機能としてパッケージ，分離コンパイル，派生型，多重定義，汎用単位などについて述べられている．

第4部（16章～18章）……実時間処理のための機能についてであり，例外処理とタスクが扱われている．

また，巻末には付録として構文一覧と標準環境 STANDARD が付されている．以下各章について簡単な紹介をする．第1章は，「Ada プログラム：手続きと関数」で平均計算の手続きとベクトルの内積の関数のプログラムを見ながら，手続きや関数がどのような形と機能を持つものかが概観されている．第2章は，「Ada プログラム：パッケージ」で乱数発生のパッケージと表操作のパッケージの例を示しながらのパッケージの概要紹介である．第3章は，「プログラムの構成」でプログラムを書く上で必要となる基本的事項のもっとも大きな単位ともっとも小さな単位の両面からの紹介である．分離コンパイル，識別子や予約語，パッケージ STANDARD の概略が述べられている．第4章は，「入出力」であり入出力の基本的な考え方の説明がある．

第5章は，「型と宣言」で型と宣言の基本的な考え方，列挙型，整数型，実数型の三つの基本的な型について述べられている．第6章は，「式」であり，式の構成法を列挙しその簡単な説明がある．第7章は，「文」で，Pascal との違いに注意しながら，簡単に文についてまとめている．第8章は「副プログラム」である．第9章は，「配列」で配列の宣言と使い方が文字の出現頻度のプログラムと Boyer-Moore の文字列照合アルゴリズムを例として説明されている．第10章は，「レコード」でレコード型を使った実用的なヒープソートの例がある．第11章は，「アクセス型」で他の言語でいうポインタ型といわれるものである．その使用例の説明がリストと木を使って行われている．

第12章「パッケージ」は，第2章をより詳しく説明したものである．第13章「プログラムの構成と分離コンパイル」は，大規模なプログラムを分割していく方法の説明でありその例として相互参照のプログラムが載せられている．第14章は，「名前について」である．Ada は，パッケージや分離コンパイルのほかにも，大規模プログラミングに関する機能をいくつか用意している．型定義における派生型，識別子や演算子などのプログラム・テキスト上での可視性，それに関連しての別名と多重定義の紹

介である．第15章は，「汎用単位」で汎用副プログラムと汎用パッケージの話である．汎用手続きの例としてクイックソート，汎用パッケージの例として線形計算のプログラムがある．

第16章は，「例外処理」の説明である．例外とは，異常事態に名前をつけて，区別できるようにしたものである．その例外の発生と処理方法が述べられている．第17章は，「タスク」である．タスクは，並列処理のための機能である．この章では，Ada の並列処理機能を概観し，並列処理プログラムの基本的パターンが紹介されている．第18章「タスク(続)」は，前章に引き続いてのタスクの説明である．言語仕様についても例題に沿って説明がなされている．並列ソートの例があるがデータ1個につき1個のタスクを割り当てるという非現実的ではあるが，正に Ada のタスク機能の特徴を表しているプログラムはおもしろい．

表現仕様，ハードウェアに直接依存する機能，ファイル，プラグラマなどはほとんど述べられていないが，これは本書が入門書であるという点からすれば了解できる．Ada 用語の日本語訳は，まだ定まったものがないので英語の索引があればもっと親切であったであろう．Pascal のような言語に親しい人にとっては格好の入門書である．もっと詳しく Ada を知ろうとする人は次の文献がよい．

[1] Reference Manual for the Ada Language, ANSI/MIL-STD-1815 A, US Department of Defence, Jan. 1983, 邦訳：情報処理振興事業協会編：最新 Ada 基準文法書，bit 別冊，共立出版，1984.
[2] 米田信夫編：プログラム言語 Ada，bit 別冊，共立出版，1981.
[3] N. Gehani, ADA: An Advanced Introduction, Prentice Hall, 1983, 邦訳：ADA 上級プログラミング，啓学出版，1986.
[4] G. Booch, Software Engineering with ADA, Benjamin, 1983.

（ソフトウェア生産技術一部　山田　勲）

中桐　滋
久田俊明共著

“確率有限要素法入門”
——不確定構造の解析——

培風館，A5判，iv＋309 pp.，1985年，3,900円

近年のコンピュータによる数値解析手法，とくに有限要素法（FEM）の急速な発展は，信頼性の高い構造物を経済的に造り出すことに大きな貢献をしてきた．CAE の名の下，FEM の浸透が深まり，ますます多くの FEM プログラムの使用者が解析に要するコストを意識しながら，設計対象物をできる限り忠実に表現する解析対象物（解析モデル）を造り出す努力を重ねている．しかしその努力の結果である解析モデルが，一つのサンプルに過ぎないという考えが一般的である．これは入力された形状，境界条件，材料の定数を表す数値を確定値とみることに起因する．つまり，これまでの FEM によるモデル化は，入力された数値の持つ種々の不確かさを考慮に入れない確定的なものであった．

本書は構造物の不確定性を考慮した一手法，確率有限要素法（Stochastic Finite Element Method）の入門書である．

第1章「序説」で，確率有限要素法の基礎的概念と手法の骨子が述べられる．

有限要素法および本書で関連する確率の諸事項が簡単に復習され，2個の確率剛性バネ（バネ定数が確率変数）を含む3質点系モデルの上で摂動法により，確率応答変位が計算されることが平易に述べられる．確率応答は期待構造（確定バネ）の解と期待構造からのずれ（摂動）により，表されることが示される．この手続きを理解した読者は，第2章「静解析問題」における基礎方程式（微分方程式）を離散化して得られる剛性行列の種々のパラメータ値（節点の座標値，ヤング率，ポアソン比，熱伝導率，電気容量，電気抵抗など）の確率密度関数を与えて，変位，応力，温度，電圧などの摂動解と統計量（期待値，分散・共分散など）を得るといったように応用範囲を広げることができるはずである．

また，1，2章で著者が強調している以下のことは重要である．

① 摂動法によれば，原則的に支配方程式は期待構造に関するものを1回解けば，そこからの変動率は付随的計算だけで得られる．
② 摂動法が取扱いうる変動量はあくまで“十分に小さい”ことが必要である（確率密度関数の幅が狭いこと）．
③ 摂動次数を上げても，必ずしも精度が上ることにはならない．
④ ②，③に関連して，一般に採用すべき摂動次数と，確率変数に許容される変動幅について結論することは困難である．

なお，本章を読めば，確率有限要素法が非線形構造解析，流体，電磁場，など従来の FEM 領域に広く適用できることがうかがえるが，本書の範囲は構造物の静解析，固有値解析および動解析に限定

している．

第３章「不確定構造の固有値問題」では，読者は２バネ系実固有値解析と１自由度減衰振動系複素固有値解析により，固有値問題の世界に容易に入り込むことができる．

第４章の「動解析」では，時刻歴応答およびスペクトル解析が扱われるが，中心課題が不確定減衰であることは納得がいく．その理由は，これまでの確定モデル化において，減衰係数の設定に不安がつきまとうのが常であったからである．さて，第５章「構造安定性，信頼性問題への応用」では，著者は“新たに開発した逐次摂動法”を主張している．

この手法は，従来の摂動法が持っていた構造信頼性の欠点を克服するために開発された．その欠点というのは，問題となる領域が期待値点から相当隔ったところに位置することが多いにもかかわらず，従来の手法では取扱い得る変動量があくまで十分小さいことを前提条件としているため，摂動解の精度が十分に保証されないことであった．

また，この方法は，方程式の求解が唯一回（期待値点で）だけという摂動法の利点を保持しつつ，展開点の逐次移動により変動範囲を広げようとの試みとも言える．

一般に，本書における確率変数の変動範囲や，高次摂動の採用の良否への言及は，数質点系シンプル・モデルについてであり，使用者が実際の複雑な構造物に対するとき，混乱はまぬがれないと思える．また，使用者の立場からは，数多い候補の中から何を確率変数として選び，その確率分布関数に何を当てるか，その変動範囲はどうか，といったことをモデル化の際，悩まざるを得ないだろう．

さらに，プログラム開発者の立場では，何を確率変数として準備すれば良いかの判断，確率分布の入力の仕方，応答の出力方法など十分考慮を加えなければならない．数値計算上は，幸いにもK行列（摂動０の期待構造）のコレスキー分解は多くのプログラムでなされているので，その結果の大行列を保存しておけば，１次，２次，…の変動率は後処理として組み込み可能であろう．この意味で，少くとも計算処理部分については，NASTRAN など従来のプログラムの変更は困難でないと判断できる．

本書は，一冊の本として確率有限要素法が紹介されたのは最初（と思う）であり，総じて入門書にふさわしいモデルの選択と平易な記述があり，多くの人の注目するところとなるであろう．

また，この手法が広く採り入れられるか否かは，真に設計者が必要とするか否かの問題であり，評者の予想は無意味であろう．

（応用ソフトウェア二部　渡部義維）

ISSN 0289-6257

技　　報

UNIVAC TECHNOLOGY REVIEW

No. 13

発	行	日	昭和62年5月31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

あなたとナレッジ・エンジニアのパートナーシップが、AIの新時代を拓く。

AIの活用により、問題解決の世界は大きく広がります。それは確かな知識や経験を持つあなたと、それを的確にシステム化できるナレッジ・エンジニア(KE)との共同作業により初めて可能となるのです。アプリケーション開発の豊富な経験とAIの専門知識をあわせ持つKEのサポート——いわば専門家とKEの緊密なパートナーシップが決め手です。それぞれがオーダーメイドといわれる専門家(エキスパート)システムの分野だからこそ、お客さま固有のノウハウをひとつひとつていねいにシステム化できるKEが必要なのです。ユニバックのKEはあなたの企業の問題解決を的確にサポートします。ユニバックでは、総合的な問題解決のためにAI技術を駆使し、つねに最良のシステムをお届けするため、ハードウェアとソフトウエアとともに、優秀なKEと充実した教育システムを用意しています。ユーザと最良の信頼関係を結び、問題解決のために最も効果的なシステムを構築するユニバックのKEは、あなたの企業の問題解決に対応した高度な技術を提供するパートナーです。

問題解決に即応する「KS-301」、そして「KEE」。

知識工学専用ステーション「KS-301」、そしてエキスパート・システムの構築と利用のためのソフトウエア「KEE」。いま話題のAI技術を応用した日本ユニバックの「知識システム」構築のためのツールです。AI時代を迎えた新しい情報処理の世界で、総合的な問題解決のために力強く応えます。

ユニバック知識システム

KS-301 / KEE

★KEEは、Intelli Corp社の商標です

「知識と経験」を、貴重な財産にする。鍵をにぎるのは、ナレッジ・エンジニアです。

あなたの会社にも、長年蓄積された「知識と経験」という名の貴重な財産が沢山ありますね?その「知識と経験」が、いまAIという新しい問題解決ツールによって、生きたカタチで活用できる時代が到来したのです。ユニバックのナレッジ・エンジニアが、あなたの会社の貴重な財産の活用をお手伝いします。

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