ICOT Technical Memorandum: TM-0126

TM-0126

マルチウィンドウシステムを通して見たインターフェースの考察

内田俊一, 辻 順一郎

June, 1985

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

マルチウィンドウシステムを通して見たインターフェースの考察

新世代コンピュータ技術開発機構

内田 俊一　　辻 順一郎

1. コンピュータの使い易さとは？

コンピュータが，“パーソナル”なものとなり，オフコン，パソコン，ワープロといった形で使われるようになり，コンピュータの「使い易さ」の追求は，新たな展開をむかえようとしている．

我々が現在たずさわっている第5世代コンピュータプロジェクトの研究開発の中でも，この「使い易さ」の追求は，主要な研究開発目標となっている．“コンピュータが知識を持ち，人間の忠実な助手として働く”とか，“自然言語で人間と対話できる”というのは，第5世代コンピュータのキャッチフレーズの一部であり，「使い易さ」の別の表現ともとれるものである．しかし，より工学的にこの「使い易さ」を考えると，コンピュータ，すなわち，マシン側が，自然言語を理解したり，音声や図形を認識する機能を持っていれば，確かに人間側にとって便利であるが，はたして，それだけで使い易くなるだろうかという疑問が生じる．すなわち，これらの機能が別々にあっても，人間とマシンとの“対話”においては不十分であり，うまく“操作”できるようシステムとして“インテグレート（統合化）”されなければ決して“使い易く”ならないだろうということである．

第5世代コンピュータプロジェクトでは，「知的インターフェースシステム」として，この“統合化”も含めた目標を設定している．しかし，この人間とマシンの対話における“操作性”も加味した“統合化”をどう研究するかは，研究の対象の半分は“人間”という不可解な代物だけに，大変難しくなる．自然言語理解，音声・図形認識といった個別の機能は，いろいろな理論もあり，研究上のアプローチも客観性をもっているが，“操作性”を中心においた「使い易さ」の研究となると多分に経験的に研究をすすめざるを得なくなってくる．また，その研究の性格も，自然言語理解等に比べ，地味なものとなる傾向が強い．しかし，その重要性は，最近よく認識されるようになり，認知科学と呼ばれる分野とも結びついて，工学的な研究も盛んになってきた．

我々の研究所においても，第5世代コンピュータプロジェクトのための茶木的ツールとして，論理型言語をベースとするスーパパーソナルコンピュータの研究開発を行い，そのマン・マシンインタフェースを「使い易く」するために，かなりの努力を費した．そこにおける中心課題は，マルチウィンドウシステムと呼ばれるものである．この種のシステムは，LISP マシンを代表とする最近のスーパパーソナルコンピュータでは，標準装備となっており，使った人も多いと思われるが，これを最初から作った人は，我国では，きわめて少いと思われる．そこで，これを題材として，マルチウィンドウシステムがどうしてもてはやされるか，どんな効果があるかを紹介してみることとする．

2. PSI と SIMPOS

マルチウィンドウシステムについて議論する前に，これを含むスーパパーソナルコンピュータについて紹介しておくことが必要である．

第5世代コンピュータプロジェクトは，その計画の中で，論理型言語をソフトウェア，ハードウェアの共通的基盤と定め，特にソフトウェアの研究開発は，論理型言語を用いることとした．論理型言語の実用的な処理系としては，DEC-10 PROLOG 等があったが，大型汎用マシン上の PROLOG は，LISP 等と比較し，十分なプログラミング環境を持っていなかった．

このため，このプロジェクトでは，その10年計画の前期3年の間に，PROLOG を改良拡張した，より記述力の優れた新しい論理型言語とそのためのプログラミング環境を提供するスーパパーソナルコンピュータを逐次型推論マシン (SIM : Sequential Inference Machine) と名付けて，開発することとした．逐次型推論マシンの中核は，PSI (Personal Sim) とその OS, SIMPOS (SIM Programming and Operating System) である．

PSI は，マン・マシンインタフェースとして，ビットマップディスプレイ，マウス，キーボードを持ち，性能的には，DEC 社の大型汎用機 DEC 2060上で走る PROLOG と同等の 30 KLIPS の実行速度と，80 M バイトの大容量主記憶を持つ高性能マシンである．

SIMPOS は，対話的利用に重点を置いたパーソナル OS であり，マルチウィンドウシステムのほか，ネットワークシステム，ファイルシステムなどを持つほか，エディタ，ディバッガ，ライブラリアン等，高度なソフトウェア作成に必要なツールを持つことを特徴としている．

プログラミング言語としては，PROLOG にクラスと多重継承機構に基くモジュール化機能，マクロ機能を融合化した ESP (Extended Self-contained Prolog) を提供しており，SIMPOS 自身もすべて，この言語で書かれている．

PSI は，すでに20数台が試作され，SIMPOS ともども，我々の研究所および，関連メーカ8社内で，実際にツールとしての利用が開始されている．

PSI と SIMPOS の設計・試作は，昭和57年の中頃より開始され，それ以降，「使い易い」マシンとは何か，「優れたプログラミング環境」とはいかなるものかという議論が続けられている．中でも，マン・マシンインタフェースの核となるマルチウィンドウシステムと種々のプログラミングツールから成るプログラミングシステムの部分には，多くの努力が注ぎ込まれた．

ここでの努力は，その視点の違いから2つに大別できる．その一つは，エディタ，ディバッガ，ライブラリ，エラー処理・ヘルプモジュールなどのプログラミングツールの機能，より具体的に言えば，コマンドの論理的機能をどう選ぶと使い易いかという「機能的」な視点からのものであり，

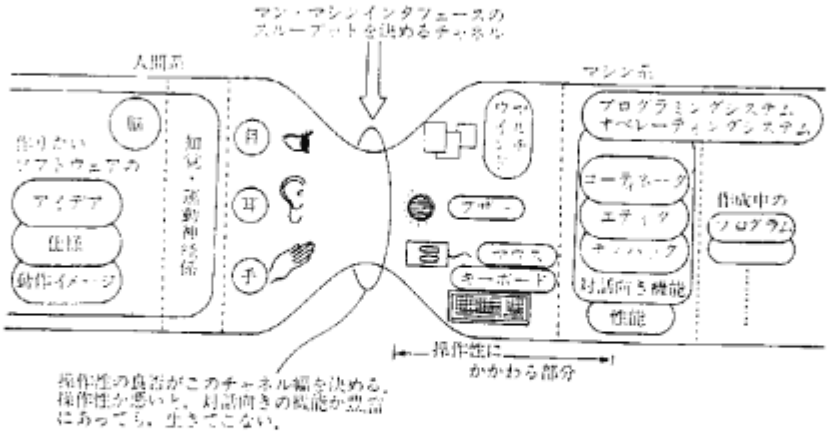


図 1　マン・マシン対話系概念図

もう一つはこれらコマンドをマルチウィンドウシステムやマウス等を用いてどのようにユーザ（プログラム開発者）に提示しどのように応答をもらったら使い易いかという「操作的」な視点からのものである．

「操作的」な視点から出てくるものは，プログラミングシステムとして統合化したマン・マシンインタフェースをどうユーザに見せるかという，まさに物理的な「使い易さ」に直接関係する部分の仕様であり，この中心は，ウィンドウやマウスの操作コマンドや，エディタやディバッガをどう呼出すかというOSのコマンドの仕様やこれを処理するプログラムモジュール（SIMPOSでは，コーディネータと呼ばれているもの）の仕様である．

このような操作的視点から見たシステム統合化の議論は従来から行われていたわけであるが，ここに，マルチウィンドウシステムが置かれたことで，この種の議論は，その検討対象に人間も含めた，認知科学的要素をより重視する方向へと新たな展開を見ることとなった．

3. マルチウィンドウシステムの狙い

マルチウィンドウシステムの狙いは，何枚ものウィンドウを，あたかも机の上にいろいろな書類やメモの類を広げたごとく，一つのディスプレイ画面上に表示し，人間が作業の流れを把握し易いようにすることと言えよう．

作業が進むに従い，参考にするために使った書類やちょっとしたメモの類，作りかけの新しい書類がどんどんたまってくる．一つのディスプレイの面積は狭く，（たとえば机の上といえどもそれ程広くないが）古い書類やメモは，だんだん下の方へ埋もれていく．時々は，下に埋もれた書類をいちばん上へひっぱり出して来る．でき上った書類は，ファイルへしまったり，手紙としてどこかへ送ったりする．

このような作業環境を作り出すのがマルチウィンドウシステムの役割であろう．ここで言う書類をソースプログラムと考え，メモ類をエディタやディバッガ，コンパイラ等とのやりとり（コマンドやエラーメッセージ等）と考えれば，そのままプログラム開発者がコンピュータの前に坐ってプログラムを作っている状況となる．

このような人間向きの対話手法が，コンピュータのマン・マシンインタフェースとして自然に受入れられるようになったのは，それ程古いことではない．上に述べたような状況を作り出すのは，マシンやOSの開発者側から見れば，きわめて面倒な作業であり，OSの設計思想，構造，ハードウェアの機能，構造，さらには，プログラミング言語（特に，システム記述言語）の性格さえも変えなければならない大きな問題を含んでいる．

従って，その背景には，ソフトウェア，ハードウェア技術の進歩と蓄積とともに，プログラム作成をはじめとして，各種システムの設計，オフィスにおける文書作成等が，人間の能力をフルに引き出すべき知的作業であり，そこにおける人間のコストは，マシン側にかかるコストに比べ十分大きいという認識があることとなる．しかし，いくら人間の好みにマシン側の機能や操作手順を合わせるといっても，マシン側に持たせ得るソフトウェアの複雑さや，ハードウェアの処理速度，所要メモリ量，入出力機器の能力などには限界があり，そこには，上記の人間優先を前提としつつも，おのずとコストパフォーマンスを考慮した最適点が存在する．

現時点におけるウィンドウシステムの本格的なものは，LISPマシン，Small-talkマシン，そして我々のPSIマシンくらいと思われるが，これらにおいては，マシンのパワーのかなりの部分を，このウィンドウを用いたマン・マシンインタフェースのスループット拡大にあてている．マウスを含むウィンドウ操作の応答速度を早くすることは，なかなか大変なことである．ともすれば，従来，マシンにおいて主な処理とされていた記号処理や数値計算等のCPUのみが働いていればよい処理と同等以上になり得る．

しかし，知識情報処理における多くの応用を考えると，それは本来，人間を相手とし，より高度なマン・マシン系を構築することを大きな課題としており，マルチウィンドウシステムを出発点とするマシンのソフトウェア，ハードウェアの再構築は，このような新しい方向への第一歩ととらえることができよう．

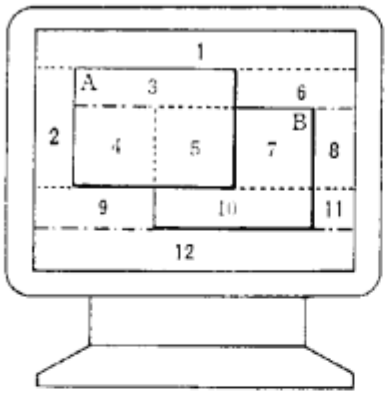
4. マルチウィンドウシステムの基本操作

人間にとっては，ごく自然な操作であっても，往々にして，マシン側での実現は面倒なものが多く存在する．ここでは，どんなふうにウィンドウを用いた入出力が実装されるかを，いくつかの基本操作をもとに説明する．

1) ウィンドウの表示と管理

マルチウィンドウの環境下では，一つのディスプレイ画面（ビットマップディスプレイの画面）上に，多数のウィンドウが表示される．当然いくつかのウィンドウは，その一部もしくは全部が隠される．隠されていたウィンドウが，最上層へ引き出されたとき，新しい位置関係に従った表示画面を作り出し再表示しなければならない．

このためには各ウィンドウの位置，大きさ等の属性をオブジェクトとして保持するとともに各ウィンドウの画面イメージをドット単位でメモリ中に保持する．また再表示の高速化のために図2に示すように実際のディスプレイ画面を多数の矩形領域に分割し，その各部分にどのウィンドウが含まれているかというウィンドウ間の位置関係を保持し，動的に管理する機構を設けることが必要となる．



含まれるウィンドウ	矩形領域の番号
ウィンドウなし	1, 2, 6, 8, 9, 11, 12
ウィンドウ A のみ	3, 4
ウィンドウ B のみ	7, 10
ウィンドウ A と B	5

図 2　高速表示のためのウィンドウの管理

ウィンドウの配置（ウィンドウの位置，サイズ）はもちろん各プログラムから変更可能ではあるが，ユーザが作業の過程において自由に各ウィンドウの配置を変えられることが必要となる．ウィンドウの配置を対話的に変更する際にはキーボードからデータとして入力するのに比べ，マウスを使用して変更を行なう方が，視覚的な面からもはるかに容易である．

SIMPOS のウィンドウシステムでは，ウィンドウマニピュレータと呼ばれるウィンドウ操作用のツールを用意している．このウィンドウマニピュレータでは各ウィンドウの MOVE, RESHAPE, SHOW, HIDE などの項目を選択することにより，マウスを使って自由にウィンドウの配置を変えることが可能になっている．

そのほか，ディバッガ等のプログラムにおいては，ユーザが指定できるパラメータがかなり多く，また，ディバッガがプログラマに提示する情報も多くある．

このような場合，いくつものウィンドウが使われるが，各々のウィンドウ間には，上下関係があり，ある局面では，これらを一括して扱いたい場合がある．（たとえば，消去するときなど）（図3，図4を参照）

このような要求に対処するために，各々のウィンドウに対して親子関係を指定できるようにする．つまり，いくつかのウィンドウに対してひとつの親ウィンドウを定義し，それらのウィンドウがその親ウィンドウに属しているものと考えるわけである．SIMPOS では，親ウィンドウとしての機能を提供する部品クラス，子ウィンドウとしての機能を提供する部品クラスを用意しており，この親子関係において親ウィンドウは各子ウィンドウの親ウィンドウ内での配置，重なり具合の制御を行っている．また，いくつかあるいは全部の子ウィンドウのキーボード入力の入口を親ウィンドウと共有する機能を持たせることができ，これによってユーザ（プログラマ）がどのウィンドウに対する入力かを意識せず，そのプログラム全体への入力としてキーボード入力を扱うことができる．

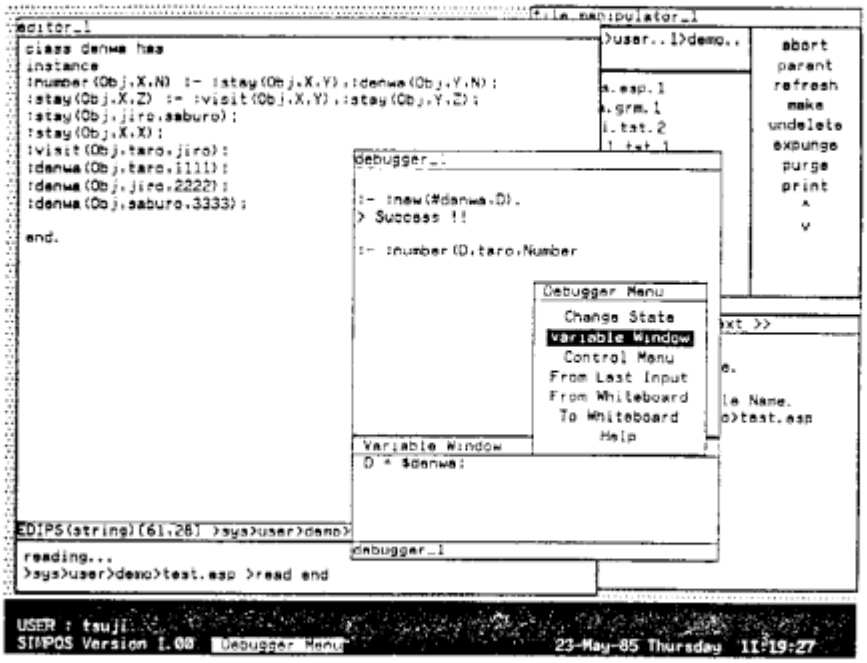


図 3 ディバッガの子ウィンドウの例（1）
ディバッガウィンドウ，変数ウィンドウ，ディバッガメニューウィンドウの3つが表示されている．

2) キーボードおよびマウスからの入力

マルチウィンドウの環境下では各ウィンドウが従来のキャラクタディスプレイの一画面に相当していると考えることができる．これに対してキーボードはただ一つしか存在せず，どのウィンドウとキーボードを結びつけるかをユーザにわかりやすく示す必要がある．

キーボード入力を送るウィンドウの指定方法としては，まず，ディスプレイ画面上を自由に動きまわれるマウスカーソルで指定する方法が考えられる．ところが画面上の何かを選択する時以外には，ユーザはマウス位置にあまり注意を払わない

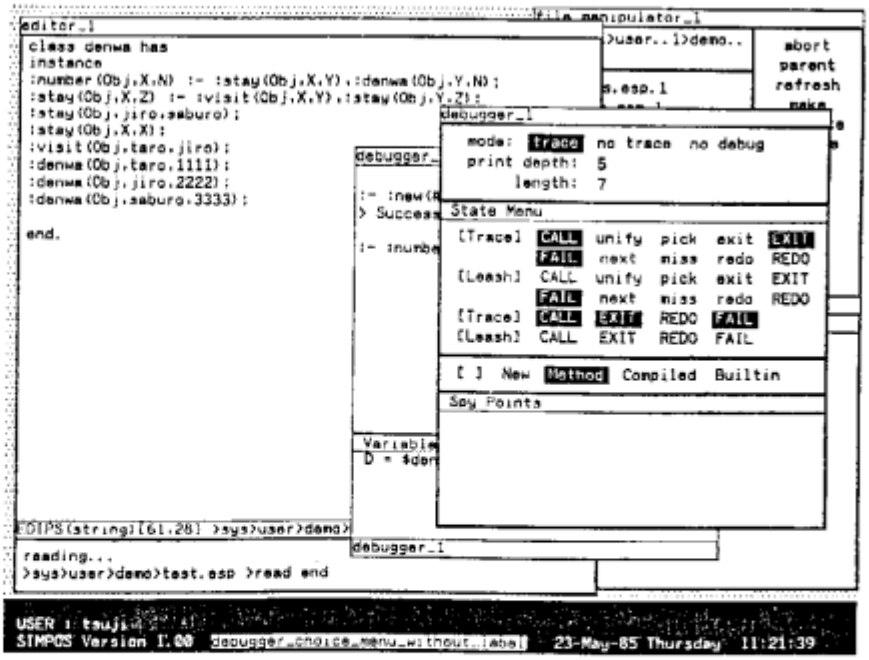


図 4 ディバッガの子ウィンドウの例（2）
ディバッガのモードや，トレース情報のどれを出力するか等の指定をする場合で，4つの子ウィンドウから成るパラメータ設定用ウィンドウが表示されている．

傾向がある．従って常にユーザに対してキーボード入力を行いたいウィンドウ上にマウスを置かせることには多少のむりがある．

一方，ユーザが一つのウィンドウに注目する場合，そのウィンドウを画面の最上層においている（そのウィンドウを画面上で完全に見えている状態に保っている）場合がほとんどである．SIMPOSでは，これを利用し，各ウィンドウにそのウィンドウがディスプレイ画面上で完全に見えているか否かという2つの状態をもたせ，完全に見えているウィンドウを，何も指示されない場合の選択対象とした．

このウィンドウでは文字入力用のカーソルが点滅し，ディスプレイ画面の最下行にその選択されているウィンドウ名が表示されるので，ユーザは容易にキーボード入力の送られるウィンドウを知ることができる．

また，マウス入力に関しても同様の問題があり，これに関しては，SIMPOSでは画面上で完全に見えているウィンドウの上にマウスカーソルがある時は，そのウィンドウにマウスボタンのクリック入力が送られるようにしている．

このほか，マウスについては，マウスのカーソルをディスプレイ画面上で，実際のマウスの動きに追随させねばならない．人間は，マウスを動かす時，早く動かしたり遅く動かしたりするから，これに対し，マウスの座標データを過不足なくとり込み，カーソルを移動することはなかなか難しい．高速な動きに追随するためには，ソフトウェア，ハードウェア双方をかなりチューン・アップすることが必要である．

以上，かなり細かな処理の例を挙げたが，このほかにも，コマンドメニューの提示方法や，各種のパラメータのセットとそれに対する応答の表示方法など，人間にとって「使い易い」ようにするためには，多くの試行錯誤的な改良をくり返す必要がある．

ウィンドウシステムを用いると，簡単なコマンドの説明程度の情報は，メニュー表示と併せて提示できるし，プログラマからいろいろなパラメータを指定させる場合も，マニュアルを横目でにらみながら指定する従来の方法であれば使われないような細かな機能も使われるようになる．

この結果，使い易くしようとする努力は，コマンドの機能や応答方法等，各種プログラミング・ツールの仕様の改良にまで及ぶこととなる．

5. ウィンドウシステムの効果的利用

マルチウィンドウシステムは，作業経過の途中状態を保存し，過去に逆のぼって参照することを容易にしてくれる．このような特徴は，プログラムの作成過程で，その有効性を十二分に発揮してくれる．

まず，エディタによってプログラムを作成する際には，既存プログラムをファイル中からさがし出し，その一部を抜き出してきたりするが，このような場合，いくつものウィンドウがあることは大変便利である．

次に，コンパイラを使って，プログラムをコンパイルすると，多くの場合，コンパイル・エラーが発生し，そのエラーメッセージが出される．このエラーメッセージも，保存できるから，再びエディタに戻っても，プログラムのエラー発生個所とこのメッセージを照合しながら修正ができ，人手でメモをとったり，プリンタ用紙を見なくてもすむ．

ディバッグの段階でも，ディバッガのトレース出力と，エディタのウィンドウに表示されたソースプログラムを見比べながら，実行経過が追跡できる．もし，ディバッグ対象となっているプログラムが，対話をするプログラムであり，画面への出力やキーボードからの入力がある場合では，これらの入出力は，また，別個のウィンドウに表示されるから，ディバッガ等への入出力と明確に分離され，不要な混乱を避けることができる．

以上のような利点だけでも，十分な効果があるが，このレベルまでくると，さらに次のような拡張も行いたくなる．

すなわち，ディバッガにより，バグを発見するとエディタでそれを修正し，コンパイルして再びディバッガへ戻る．そして，先ほどバグの出た個所までプログラムをトレースしながら走らせることとなる．この時，以前，ディバッガへ与えたコマンド列が保存されているから，これを再びディ

バッガへ与えれば，速かに，先程バグの出た個所まで到達でき，さらに便利となる．このような機能の追加は，マルチウィンドウの機能が土台にあって，はじめて有効になるものである．

このようなセッション管理機能は，人間の操作の手間の軽減にきわめて有効であり，今後，ますます発展していくものと考えられる．

6. おわりに

マルチウィンドウシステムの効果をプログラミングの観点から紹介し，いくつかの具体例を示した．このシステムは，人間とコンピュータ間の情報流通のチャネルの幅を従来の方法に比べ，何倍にも拡げたと考えられる．従って，今後，エキスパートシステムや自然言語処理システムなどの複雑なシステムが作られ，人間がマシン側の内部の様子を把握するのがより難しくなるに従って，その重要性は，ますます高まるはずである．

このようなシステムでは，ウィンドウを用いた情報提示法も，より複雑かつ専用化されるから，ウィンドウ機能のカスタム化が不可欠の要素となってくる．その場合には，クラスと継承機能を用いたオブジェクト指向の概念に基くウィンドウシステムおよびこれを含むOSのモジュール化等が求められよう．

そして，それはシステム記述言語やそれをサポートするアーキテクチャ，ハードウェアシステムへも新たな要求を投げかける．

SIMPOSやPSIは，このような基本機能をすべてとり込んでおり，実際使えるようになってはいるものの，「使い易さ」とか「操作性」という面では，まだ多くの改良すべき点を含んでいる．今後，多くのユーザによって本格的に利用されることで，いろいろな改良がなされ，それに伴いまた新たな知見が得られるはずである．そして，それらの知見は，第5世代コンピュータの研究開発における「使い易さ」の追求のために有益な基礎データとなることが期待されている．