解　説

4. 新しい技術の応用

4.1 新しいプログラマ・インタフェースの利用†

久　野　　靖††

1. はじめに

近年，ソフトウェア開発のための計算機環境（言いかえれば，プログラミング環境のうちのハードウェア部分）としてビットマップ表示装置とマウスを備えたワークステーションの使用が一般化しつつあり，大型計算機（ないしスーパミニコン）の共同利用にとって代わるようになってきている．これはワークステーションを使用した開発の高生産性が広く認識されてきたためであろう．生産性が高まる原因には，個人が占有できる資源が格段に豊富になるというハードウェア面とビットマップ表示装置やマウスを活用したインタフェースの利用を初めとするソフトウェア面が考えられる．特に後者の分野はまだ発展途上であり，その発展によって今後さらに生産性が高まることが期待される．またそれによりはじめて，増大するCPU性能などのハードウェア能力を生産性向上のために有効活用する道がひらけるものと思われる．

本稿では以上のような視点に基づき，ビットマップ表示装置とマウス活用の現状，その利点と問題点，および将来の可能性について，プログラミング環境としての観点を中心に概説している．以下 **2.** ではビットマップ表示装置とマウスを用いたインタフェースの実例について，いくつかの代表的なシステムを取りあげて紹介し，**3.** ではこれらのもつ特徴と利点を一般化しまとめる．さらに **4.** ではこれらのインタフェースにみられる問題点およびそれらを解決する可能性について論じる．最後に **5.** ではまとめを行う．

2. ビットマップ表示装置とマウスを用いたインタフェースの実例

本章ではビットマップ表示装置とマウスを活用したインタフェースをもつ代表的なシステムをいくつか取りあげ，その特徴について概説する．

2.1 Star/JStar

Star は米ゼロックス社で開発された文書処理ワークステーションであり，また JStar[5] はその日本語版である．これらは世界最初のワークステーションである Alto の流れをくむものである．これらのシステムの大きな特徴は，さまざまな種類の文書やそれを集めて保管するフォルダ，文書を印刷するためのプリンタなど，利用者が扱う対象物をアイコンとして画面上に表現したことである．これにより利用者は自分が操作したいものを画面上で直接指して操作できる．たとえば，ある文書を編集したければその文書をマウスでクリックしてから「開く」指令を実行すると，そのアイコンは「開いて」窓となり，中身が見えるようになる．そしてその中の特定の場所を直したければ，その場所をマウスでクリックして修正を指定できる．

また，文書をフォルダに入れたり印刷する場合は，そのアイコンをドラグしてフォルダやプリンタのアイコンに重ねればよい．アイコンの操作のみで表せないような指令は，まず操作したい対象をクリックにより「選択」し，続いて指令キーによって操作を起動する．このため，通常の文字キーに加えて「複写」，「開く」など多数のキーが搭載されたキーボードを備える．

これらのシステムのもう一つの特徴は，文書や図形などを編集する際にビットマップ表示装置の特徴をいかし，できるだけ最終印刷出力に近いものを常に画面に表示する方式（見たまま方式，WYSIWYG—What You See Is What You Get）を採用したことである．線の太さを変更したり文字のフォントを変えたりすると，そのことはただちに画面上で確認できる．

2.2 Macintosh

Macintosh（以下 Mac）は米 Apple 社によって開発されたパーソナルコンピュータであるが，そのインタフェースはビットマップ表示装置とマウスを全面的

† Usage of Bitmapped-Display and Mouse in Programming Environment by Yasushi KUNO (Graduate School of Systems Management, the University of Tsukuba at Ootsuka).

†† 筑波大学経営システム科学専攻

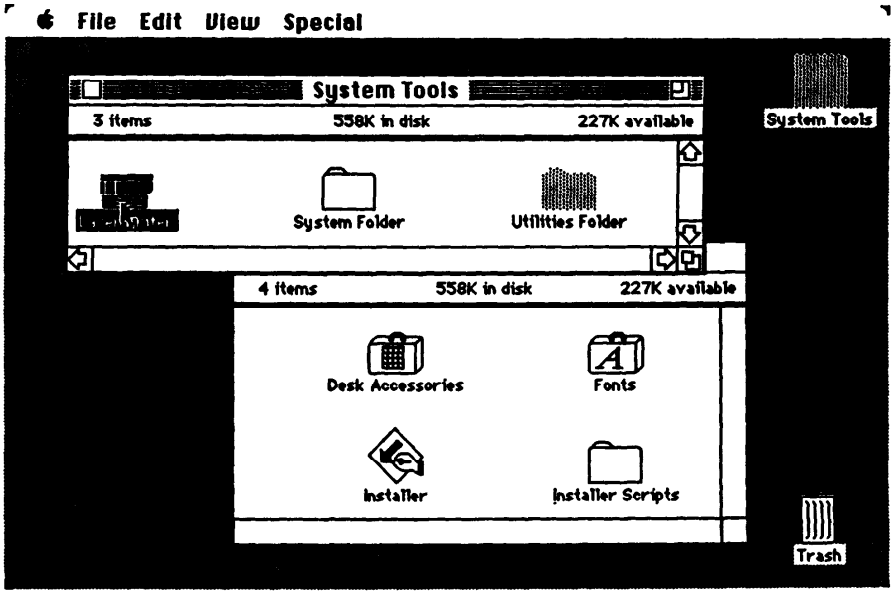

図-1 Mac の窓とアイコン（ファインダ画面）

に活用した先進的なものである．Mac のハードウェア上の特徴はマウスが1ボタンであることで，複数ボタンのマウスを備える他システムと対照をなす．1ボタンであれば利用者はどのボタンを押すか迷わずにすむが，一方でマウスで行える操作の種類が少なくなる．Mac ではこの問題を，ソフトウェア全体を1ボタンで使いこなせるように注意深く設計し，さらに2度続けてボタンを押す（ダブルクリック），左手で Shift や Option キーを押しながらボタンを押す，などの方法を併用してうまく回避している．

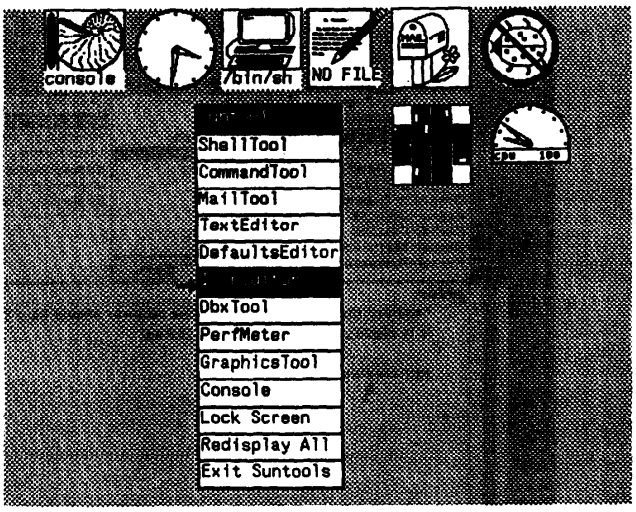

図-2 SunView のアイコンとポップアップメニュー

ソフトウェア的には，Mac でもさまざまな文書をアイコンで表し，それを指し示して開くところは Star などと同様であるが*，指令の起動にメニューをもちいる点が異なる．Mac では画面の一番上に現在利用可能なメニューの一覧が常に表示されており（メニューバー），そこをマウスでプレスするとメニューが降りてきて（プルダウンメニュー）その中から操作を選択できる．メニューの利点としては，指令キーと異なり画面から目を離す必要がない，必要に応じて動的に項目が変更できる，現在利用可能なもののみを表示することで利用者を導ける，などがあげられる．Mac の場合にはさらにメニュー名が常に見える．メニュー

* Mac の応用プログラムのアイコンについては特定の文書に対応していないので「起動する前の応用プログラム」，あるいは「応用プログラムの窓の閉じた状態」と考えることもできる．ここでは応用プログラムを起動すると何も入っていない文書の窓ができることから応用プログラムのアイコンは「からっぽの文書」に対応していると考えることにする．

バー→メニュー→項目という階層構造がある，アプリケーション間でメニュー構成の共通化が計られているなどの特徴もある．実際，文献[1]ではアプリケーション開発者が従うべきインタフェース設計の指針についてかなり具体的に記されている．

2.3 Unix ワークステーション

近年，Unix オペレーティングシステムを搭載したワークステーションが広く使用されるようになってきているが，これらの上で稼働するウィンドウシステムの代表的なものとしてX[13]，SunView[14]，GMW[2] などがあげられる．これらでは Star や Mac と異なり，窓は文書が開かれたものではなく，窓を作り出すような実行中のプロセスにほぼ対応している（一つのプロセスが複数の窓を管理してもよい）．したがってアイコンも単なる一時的に閉じられている窓に過ぎず，文書に対応しているわけではない．

これらのシステムでは操作の指示には主としてポップアップメニューが使用される．ポップアップメニューの利点は，マウスカーソルの位置にメニューが表示されるため，視線の移動および選択のためにマウスを動かす距離が小さいことである（このことは Mac に比べて大きな画面をもつシステムでは重要である）．一方，複数のメニューを使い分けるためには，マウスの複数のボタンを使い分けるのに加え，画面上のどの領域でボタンが押されたかによって異なるメニューが出てくるなどの方法を採用するのが普通である．

2.4 Smalltalk-80 と Lisp Machine

Smalltalk-80[7] システムは米ゼロックス社で開発されたオブジェクト指向言語 Smalltalk-80 のための環境である．また，Genera[15]は Symbolics Lisp Machine 上で稼働するソフトウェア開発環境である．これらのシステムではブラウザと呼ばれるツールが重要な役割を果たす．ブラウザはシステム内に登録されているコードや情報を(通常マウスを活用しながら)検索するツールであり，たとえば Smalltalk-80 のブラウザでは利用者はクラスカテゴリ，クラス，メソッドカテゴリ，メソッドという分類の各レベルを次々とマウスで選択していくことで目的のコードにすばやく到達でき

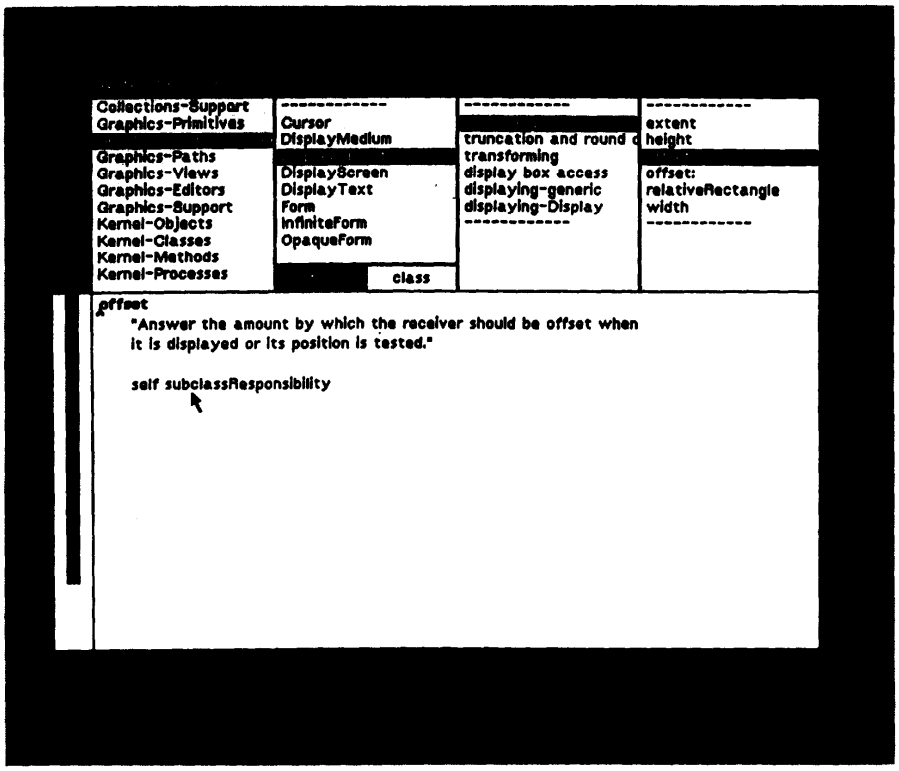

図-3 Smalltalk-80 のブラウザ

る．このような木構造の枝の向きに沿った検索に加え，コード中に現れる識別子からそれを定義している場所を見つけるなど横向きの(離れた枝に移っていく)検索機能も備えている．

これらのシステムではブラウザのほかにもデバッガやテスト実行などの機能も充実しているが，全体に「この部分を実行する」「この部分を調べる」などマウスにより表示されているテキストを直接クリックして操作の対象を指定できる．このような特徴は，これらのシステムが解釈実行（ないしそれに近い操作性）を基本とした単一言語系であるため言語処理系からの多様な情報を活用しやすいという「土壌」に最初からビットマップ表示装置とマウスの使用を前提としてインタフェースをデザインするという「種」がまかれた結果であるといえる．

3. ビットマップ表示装置とマウスを用いたインタフェースの利点

本章では，前章で解説したそれぞれのシステムの特徴から一般化して，ビットマップ表示装置とマウスを用いたインタフェースがもつ利点について整理してみる．

3.1 表　示

ビットマップ表示装置が従来の文字端末と大きく異なるのは，画面に表示されるものが基本的に「絵」だという点である．一般に「絵で見る」ことで「文字で読む」よりずっと多くの情報を短時間で獲得できるので，その意味ではビットマップ表示装置は大きな可能性をもつといえる．一方，現状ではビットマップ表示装置の使われ方は

- 図形またはイメージ表示としての活用

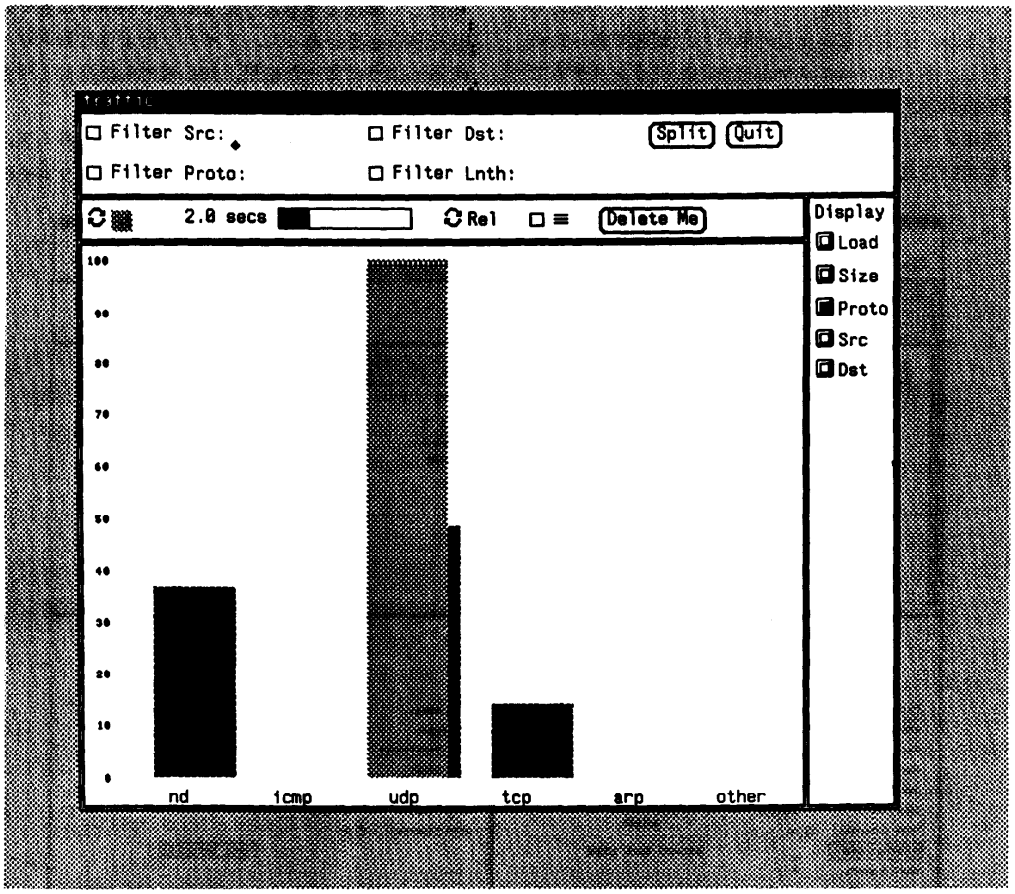

図-4 図形表示の例：ネットワークトラフィック表示ツール

- メニュー，フォームによる活用
- アイコンによる活用
- 文字表示による活用

の4種類に整理されるように思われる．以下，これらについて順次概説する．

まず図形やイメージが表示できるようになった結果，従来は数字や文字で表示していたものをグラフなどで視覚化して表すようになった．その例としては計算機内部の状況やネットワークの状況などをグラフとして表すツールなどがあげられる．また，エキスパートシステム構築系などでさまざまなパラメータを数値の代わりにメータの形で表示できるようにしているのも同様である．このような数値の視覚化とは異なる方向としては，ソフトウェアやその設計に関する情報を図形化して表示するなどの応用がすでに実用化されている．全体的に図形表示としての活用は従来の図形端末（Graphic Display Terminal）の延長であるともいえるが，図形端末が比較的限られた資源であったのに対し，多くのプログラマが日常的に使うシステムで図形表示が活用できることの意味合いは小さくない．

次にメニューやフォーム（書式）は従来の文字端末でも用いられていたが，文字端末では入力がキーボードであるため「n番目の項目へいく」「一つ前/後の項目へいく」などの操作がわずらわしく，これらの利用は一部の応用に限られていた．一方，ビットマップ表示装置とマウスを用いた場合にはマウスで直接目指す項目を指定できること，メニューやフォームの配置や外見が自由にデザインできることなどからソフトウェア開発環境を含め一般に広く使われるようになった．特にフォームの場合には従来の入力欄に加えて押しボタンや選択スイッチなど動的な要素が盛り込めるようになった点が大きな変化といえる．

これらの行き方とは異なり，図形端末時代には考えられなかった「絵」の使い方がアイコンである．アイコンとは基本的に「計算機内部の抽象的な実体や指令を絵として具体的に表した」ものである．アイコン以前にはファイル，ディレクトリなど計算機内部の見えない実体を，やはり目には見えないさまざまな指令を駆使して操作できるようになるまで利用者は多くの修練を積む必要があった．アイコンによりこれらが「見える」ようになった結果，初心者でも抵抗なく操作が行えるようになった．

これらの状況にも関わらず，画面に一番多く表示されるものが「文字」であることは（プログラムそのも

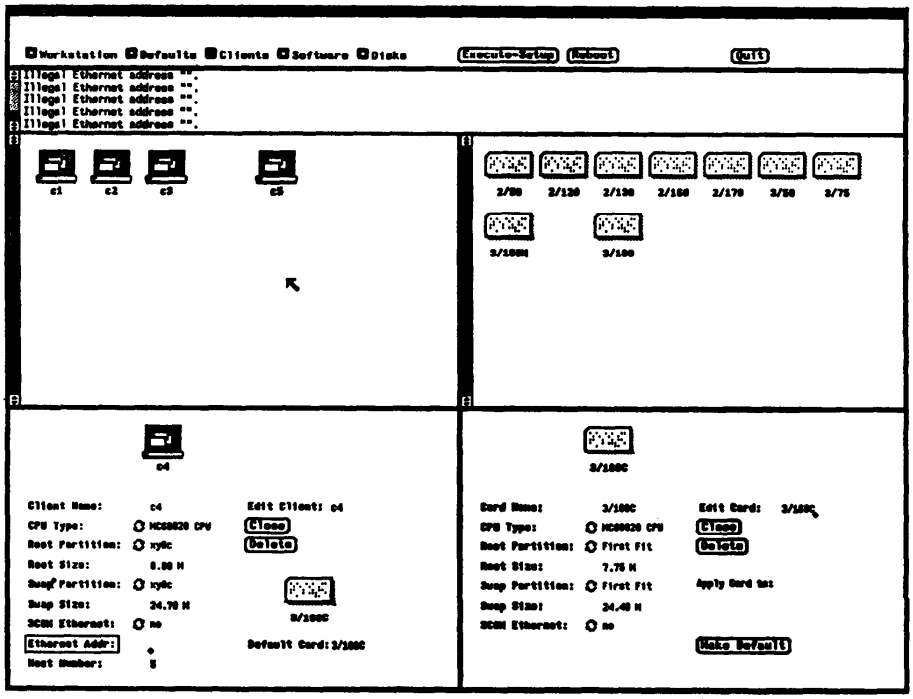

図-5 フォームの例：計算機構成設定ツール

のが文字で表現されている以上）避けられない．しかしビットマップ表示装置では文字の大きさ，形，配置の各面で従来の文字端末と比較して大きな自由度があり，その活用によって利用者により多くの情報を提供できる．そのような例としては，キーワードや現在注目している部分を書体，文字の大きさなどを変えて表示するなどの方法があげられる．

以上のように現在のところビットマップ表示装置に表示されるもののバリエーションは意外に少ないが，それでも従来の文字端末より格段に大きな情報を利用者に提供し，利用者インタフェースの高度化に貢献しているといえる．さらに今後ここにあげていない新しい方式が生み出されることも期待される．

3.2 直接操作

直接操作によるインタフェース (Direct Manupilation Interface) とは，画面上に表示されている「もの」（アイコンなど）を直接マウスなどのポインティングデバイスで指して操作する方式をいう．直接操作の利点は，操作の対象を直接画面上で指示できるため，直感的に分かりやすいこと，また操作の結果が画面上にただちに反映されるため何が起きているのかが常に確認でき，安心感が得られることである．

この方式が有効であるためには画面上に表示されている「もの」が利用者が操作したいと思う計算機内部の対象物に対応していることが必要である．直接操作の対象となるものはアイコンに限らず，たとえばファイル名やモジュール名の文字列をファイルやモジュールそのものに対応していると考えて直接操作の対象とすることも可能であるし，またマウスを活用するテキストエディタでは画面上に表示されたテキストを直接指して操作するので，これも一種の直接操作であるといえる．さらに見たまま方式も「整形済みの印刷出力を直接操作している」感じを与える，という意味では直接操作の一種であるといえる．

3.3 モードレス

ビットマップ表示装置とマウスを用いたインタフェースの多くがもつ特性のうちに「モードがない」という点があげられる．これは使いやすさのための指針として従来からいわれてきたことであるが，文字端末インタフェースではあるプログラムを起動するとキーボードと画面はそのプログラムが占有するため，それが終了する（かそれを一時中断する）までは他の操作はできず，実際上モードをなくすことは困難であった（ただし Emacs などのエディタやある種のツールでは端末窓の画面を複数に分割してそれぞれを独立に使用できる．しかし切り替えの指令にキーを取られる，文字端末では分割すると個々の領域が狭い，などの問題があり，常にそのようなツールを介して作業を行うプログラマは必ずしも多くない）．これに対し，ビットマップ表示装置とマウスを用いた場合にはツールにそれぞれ自分の窓をもたせることでそれらを並行して存在させられるため結果としてモードが少なくできる．実はこれは時間的なモードを（「マウスがこれこれの場所にいるときはこのように動く」という形で）空間的なモードに取り替えたに過ぎないが，人間は空間的なモードにはもともと慣れているためこのほうが使いやすく感じるのは確かである．

モードレスを個々の指令の実行まで推し進めたのが選択＋操作型インタフェースである．選択＋操作型インタフェースではどれかの対象に操作を施したい場合，まず対象を「選択」する．選択された対象はたとえば色が変わるなどして，選択されていることが明示される．続いてメニューなどで削除，複写などの「操作」を指定すると現在選択されている対象に対して操作が実行される．メニューは表示させた後でもどの項目も選ばずに消すことが通常可能なので，操作を実行する直前まで「戻れる」状態にあり，したがって「選択されている」という一種のモードがあるにも関わらずモードがないのと同様な使いやすさが得られる．

3.4 オブジェクト指向

文字端末から指令を打ち込むインタフェースでは，同種の操作であっても操作する対象が異なれば指令も異なることが多かった．しかし直接操作方式により画面上のものを操作する場合には操作される対象によらず同種の操作は同じ動作で指定できるようにするのが普通である．たとえば Mac などでアイコンはその種類に関わらず同じようにドラグして移動できたり，ダブルクリックにより開くことができるのがこれに相当する．このように内部で起こることは異なっていても操作の仕方は同一にしておき，操作された対象の種類によりシステムが適切な解釈を適用することで利用者が覚えなければならない操作の種類を減らすことができる．

また選択＋操作型のインタフェースとメニューの組合せについても，メニューを出す操作は対象によらず同一であるのが普通であるが，一方，メニューの項目は選択された対象によってそれぞれ可能な操作のみを列挙したメニューが出てくるので，利用者はどの種類

の対象にはどのような操作が可能かを覚えている必要がなくなる．メニューの中の各項目については，再び対象の種類が異なっても同種の操作には同一の名前がつくようにしておくことができる．

このように「具体的な動作の詳細は個々の対象物に付随させておき，外部からはどれでも類似したやり方で扱える」ことはオブジェクト指向の考えと相通じるものであるが，ビットマップ表示装置とマウスを用いたインタフェースにおいて適用しやすく，使いやすさのために有効である．

3.5 学 習

ビットマップ表示装置とマウスを利用したインタフェースは文字端末を使用したものに比べて覚えやすく，そのことが使いやすさを増している．覚えやすさの理由としては次のようなものがあげられよう．

- 図形表示を活用するための情報が見てとりやすい．
- 文字だけの画面に比べて個々の画面の違いが明確なので，互いに異なる状況の区別がしやすい．
- モードがなければモードによる指令の違いを覚えなくて済む．
- メニューを使用することで，どの対象にはどの操作が行えるかを覚えなくて済む．
- 対象の種類に係わらず操作の仕方が統一できていれば覚える操作の種類が少なくて済む．
- 打ち込む指令列のテキストを覚えるより，画面上でのマウスによる操作系列を覚えるほうが「視覚的」であり，覚えやすく思いだしやすい．
- 指令列と異なり「打ち間違い」が起きない（間違って異なる指令を実行してしまうことはあり得るが，少なくとも指令が不完全なため拒否される，ということはなくなる）．

また，これら全般を通じてインタフェースが日常生活において現れる動作や用語にならうようにして，利用者が実生活からの類推や対応付けを行えるようにすることは学習しやすさを増す有効な手段であるが，ビットマップ表示装置とマウスを用いた場合には直接操作をはじめそのような方法が採りやすいという面もある．

4. ビットマップ表示装置とマウスを用いたインタフェースの問題点

前章で述べたように，ビットマップ表示装置とマウスを用いた利用者インタフェースには文字端末にはないさまざまな可能性が開けているが，一方でそのようなインタフェースが作られるようになってから日が浅いこともあり，特にソフトウェア開発用環境としてみた場合さまざまな問題点もあるように思われる．本章ではこれらの問題点について考えてみる．

4.1 ツール開発の負担

ビットマップ表示装置とマウスを駆使したツールを開発することは，従来の文字端末ベースのツールを開発することに比べて大きな労力を要する，というのが現状である．その理由としては

- 画面のデザインやツールへの入力のデザインに大きな自由度があるためそれだけ手間を要する．
- 文字のみを扱う端末と比べ入出力の基本機能が大きくその操作方法も必然的に複雑である．しかもウィンドウシステムごとにその規約が異なっている．
- 「モードレス」を実現するためには利用者の個々の操作に対応して動作するイベント駆動型の構造を取るため，従来のアプリケーションと構造が異なる．
- 「直接操作」をはじめツールの「手触り」が従来以上に重要になりそのためのコードが大きくなる．
- 以上全般にわたり，標準的なデザインや手法が確立されていないため開発者がそれぞれ基礎から積み上げていく必要がある．

などがあげられよう．

これらに対処するための試みとして，MacToolbox, Smalltalk-80, Symbolics Lisp Machine のように標準のツールキットと規約を確立しツール開発者はこれらに準拠し，またこれらに含まれている部分を利用することで一定水準のインタフェースをもつツールが作れるようにするやり方，UIMS (User Interface Management System) としてツールとは独立に利用者インタフェースの操作を受けもつ部分を用意するやり方などがある．しかし MacToolbox を使用したプログラム開発は一般のプログラム開発より大変であるとの認識が一般的であるように，ツールキットとその部品構成や規約を身につける負担は依然として大きい．UIMS の場合にもツールとは別個にインタフェースの記述を用意し，さらにツールと UIMS のインタフェースを取らなければならないなど煩雑な面が多い．

さらにビットマップ表示装置とマウスを用いたインタフェースについては現在もさまざまな模索が行われている段階であり，そしてツールキットや UIMS がサポートしていないような新しい方式を試みようとすればこれらの「標準的な道具を用意する」やり方にたよ

ることはどのみち不可能である，というのが現状である．この面からみれば Smalltalk-80 や Lisp Machine の場合には，すべての部品がソースコードで提供されているため必要なら部品そのものを好きなように修正でき，その結果より高い柔軟性を提供しているという，やや皮肉な状況が存在する．

4.2 端末窓

ソフトウェア開発用ワークステーションとして現状では Unix オペレーティングシステムを搭載したものが多く使用されているが，これらのシステムにおけるウィンドウシステムの利用状況をみると，（ソフトウェア開発環境に話を限れば）その上で最も多く利用されているツールは画面上に従来の文字端末と同等の機能をもつ窓（端末窓）を作り出す端末エミュレータであるように思える*．

Unix はもともと端末装置を前提に作られているため，端末窓があればそこから従来の Unix の機能は一とおり利用可能であるし，複数の端末窓を利用すれば指令の実行経過をみながらファイルを編集する，などの作業がスムーズに行えることも確かである．しかし，端末窓だけを使っているのではビットマップ表示装置とマウスの能力を十分活用しているとはいいがたい．実際，端末窓には次のような問題点がある．

- 利用者が打ち込んだものと，指令の出力が表示の上で混ざってしまう．
- これらの表示は実行された順に並んでいるだけで，重要なものとそうでないものの区別がない．
- 新たな入力や表示により，画面上の情報がスクロールして消えていってしまう．
- 画面エディタなど，全画面を使う指令を起動すると，それまでの表示は見えなくなってしまう．

もちろん，これらの欠点は従来の画面端末でも存在したものであるが，画面端末では表示できる情報の量が少ないため情報を区別し整理した形でずっと表示しておくことは実用的でなく，したがって問題にならなかったといえる．しかしビットマップ表示装置では端末窓の外部にも表示する場所があり，表示の形態もさまざまにできるので，上記の問題を問題として取りあげ，対策を考えるべき状態にあるといえよう．

一つの行き方はソフトウェア開発に使用する各種のツールをビットマップ表示装置とマウスを活用する形で組み立て直すことである．実際，たとえば SunView 上では dbxtool などのソフトウェア開発に有効なツールがいくつか開発されている．しかし，現在 Unix で使用されている指令数および前章で述べたようなツール開発の負担を考えるとあらゆるツールをそのような形で作り直すことは莫大な労力を要するうえ，そのようにして作りなおした環境は現在の Unix とはまったく異なるものであり，それが Unix のもつ使いやすさとバランスを兼ね備えたものであるかどうか疑わしいであろう．

これを解決する一つの行き方として，8) のようにビットマップ表示装置を活かしながら同時にさまざまな指令と組合せて使えるような汎用的なツールを用意する方法も考えられる．その場合，よく使われる指令については労力をかけても専用のツールとしてビットマップ表示装置とマウスを活用できるようにし，それ以外の部分については汎用のツールとの組合せでカバーしていくというやり方を採ることができる．

4.3 直接操作の限界

Smalltalk-80，および X，SunView を初めとする Unix ワークステーション上のウィンドウシステムではアイコンは「窓の閉じたもの」に過ぎない (Genera ではアイコンそのものがない)．これに対し Star/JStar や Mac ではアイコンは文書やフォルダなどの「もの」に対応し，窓はそれが開いた状態に相当する．この両者を比較した場合，一般的にいえば後者のほうがアイコンが具体的な「もの」に対応しているため直接操作の流儀にかない分かりやすいものと思われる．

にも係わらず，上記のようなソフトウェア開発環境で前者が主流なのは，ソフトウェア開発環境では非常に多数のファイルを扱うため後者の流儀では限界があるからだと思われる．実際，Unix ワークステーションで Mac のようにディレクトリやファイルをすべてアイコンで表すインタフェースが利用できるものもあるが[10]，多数のファイルがある場合にそれらをアイコンで表されると目的のものを目で探すのに時間がかかり煩わしい．

改めて考えてみると，直接操作型インタフェースとは計算機内部の状況をできるだけ利用者が日常生活で接している物理的な環境に近づけ，両者の類似性によって操作の仕方を直感的に分かりやすくするものであるといえる．しかし，机の上にさまざまなものが山積みになった状況を考えれば分かるように，物理的環境というのは必ずしも便利で快適であるとは限らず，そ

* Smalltalk-80 では Transcript の窓，Genera では Lisp Listner の窓などが端末窓に相当する．これらのシステムでは端末窓への依存度は Unix ワークステーションよりも小さいがやはり「なしで済ます」ことは考えにくく，また端末窓自体としての問題点は同様である．

れに近づけるほど制約が多く使いにくいものになる可能性もある．との指摘もある[11]．

また，ソフトウェア開発環境の場合には利用者が操作する対象がファイル，ディレクトリ，プリンタなどに加えてプロセス，文字列／パターン，指令のオプション，ホスト，デバイス，ユーザなど非常に多岐にわたるためそれらをすべてアイコン化して扱うのが大変であるし，そうしたとしてもそれぞれの性質が大幅に異なるため統一的に操作できる部分が限られる，という側面も無視できない．

4.4 選択＋操作およびメニューの問題点

前述のように選択＋操作型のインタフェースはモードをできるだけ少なくするために広く採用されているやり方である．また，操作の指定方法としてはメニューを使用するものが一般的である．しかし，この方式には

a）複数の対象を扱う場合にうまく適用できない．

b）メタな操作が指定しにくい．

c）メニューの選択が遅い．

d）あらかじめメニューに登録された操作しか指定できない．

e）操作の数に限界がある．

などの問題点がある．以下，これらについて検討する．

まずa）については，基本的に「現在選択されているもの」が一つ（または1群）しかないことに問題がある．たとえば複数のファイルを一度に消去するような場合にはこれらをまとめて選択してから「消去」を指定すればよいが，「ファイルAについて，ファイルBと異なる部分を打ち出す」などの場合にはAとBという二つの対象が平等でないので困ることになる．一般に複数の（しかも互いに性質が異なる）引数をもつ指令は多数あるので，これらをうまく扱えないのは問題である．Mac などではこのような場合，最初の一つは選択＋操作で指定し，残りのもの（オプションや補助的なファイルなど）については別にフォームが出てきて指定する．この方式ではフォームが一種のモードであるし，また対象の指定方法が複数あるという弱点をもつ．

b）については，たとえば Unix では

nice 指令 引数 …(優先順位を下げて指令を実行)

time 指令 引数 …(時間を計りつつ指令を実行)

などのメタな指令がいくつもあるが，選択＋操作型のインタフェースでは操作を指定したとたんに実行されてしまうのでこのようなメタな指定が行いにくい．このような指定を行うにはたとえば「次の指令は優先順位を下げる」という指定に続いて主たる動作を指定することであるが，これは一種のモードということになってしまう．

c)～e）はいずれもメニューに付随する問題である．もともとメニューは「メニューを出す操作→メニューが出る→選択したい項目を探す→ポインタを移動してその項目へいく→選択」のように選択までの操作系列が長く時間がかかる（これに対処するため，メニューを「はぎ取って」画面の別の場所に置いておくことで2回目以降の選択を速くするシステムもあるが，これだと環境に応じて構成が柔軟に変化するというメニューの利点が失われるため広くは用いられていない）．またメニューの項目はあらかじめ定まっていて，急にある操作を繰り返し行いたくなってもそれを（少なくともマウス操作などで）簡単に入れることが難しい（運がよい場合でもメニュー記述ファイルをエディタで編集してメニュー管理機構に読み込ませるなどの操作が必要である）．さらに，たとえば Unix でファイルを操作する指令の数は百を下らないが，これらを一つの（あるいは利用者に使い分けられる程度の数の）メニューにすべて入れるのは現実的でない．たとえば枝別れメニューなどにより項目数を増やしていく方法もあるが，それでは選択までの時間がよけい長くなる一方である．

このように選択＋操作型のインタフェースとメニューにはさまざまな限界があり，Star/JStar や Mac などの比較的限定された環境や作図ツールなど個別のツールによっては適する場合もあるが，ソフトウェア開発環境などにおいて全面的に適用するのは問題が多いように思われる．将来的には(たとえば[12]のように)選択＋操作型以外の手法も取り入れ，場合に応じて適するものを「選択」することが望ましいであろう．

4.5 マウスとキーボード間の往復

ビットマップ表示装置とマウスを用いた利用者インタフェースについてよく言われる意見として「マウスに手を動かすのが面倒である」というものがある．実際，X上の複数ツールについて使用状況の時間記録を解析した結果でも，マウスとキーボードの間の往復にかかる時間が無視できない比率を占めるとの結果が出ている[4]．これに対する自明な対応策としては

a）マウスのみを使用し，キーボードを使用しない．

b）キーボードのみを使用し，マウスを使用しない．

があり得る．たとえば Mac の shell に相当するファ

インダはファイル，フォルダ，ディスクの操作がすべてマウスのみで行えるのでa）を実現した例であるといえる．しかし，ソフトウェア開発という立場からみるとプログラムコードはテキストの形で表現されているので，まったくキーボードに触らないというのは実際的でない．

一方，キーボードのみを使用するという立場は，端末窓を複数開いてその上で開発を行う，という現状からみて効果的である可能性がある．しかし多くのウィンドウシステムでは複数の窓の間で入力を切り替えたり窓を操作したりする場合にマウスを使用するようになっているので，ほとんど常にキーボードに手があるにも係わらず窓を切り替えるときどうしてもマウスに手が動いてしまう．これに対処する一つの試みとして，キーボードのみで窓の間の入力切り替え，窓の移動や大きさ変更が行えるウィンドウマネージャを作成し評価した例もあり[9]，そこではいくつかの典型的な作業状況においてマウスを使う従来の方式より短時間で操作が行えるという結果を得ている．

しかし，キーボードのみを使用するのでは直接操作を初めとするビットマップ表示装置とマウスを用いたインタフェースの利点のいくつかを放棄することになるので，全面的にb）の方針を採用することも望ましいとはいえないように思われる．最終的に両者を統合した．

c）キーボードに手があるときはできるだけキーボードのみで操作でき，逆にマウスに手があるときにはできるだけマウスのみで操作が行えるようにする．という方針を採用し，どうしても現在手があるほうのデバイスに適さない操作をしたくなったとき初めて手を動かすようになることが必要であろう．これによってたとえばキーボードとマウスの往復が10分に1回程度になればこの問題は解決したといえよう．

5. まとめ

ここまで述べてきたように，ビットマップ表示装置とマウスを使用する利用者インタフェースには多くの特長があるが，その一方で今後解決されるべき問題点も多い．しかし，この種のシステムが開発されるようになってからまだ日が浅いことを考えれば，今後の研究によってこれらの問題が解消される可能性も十分あるし，またこれまでに考えられなかったような新しい操作方式の出現によりここであげなかった新たな利点が明らかになる可能性も大きい．

言い換えれば，ビットマップ表示装置とマウス（ないしその他のポインティングデバイス）は計算機利用者に初めて日常的かつ実用的に使うことのできる（実世界における意味での）「窓」と「指」を与えたものであり，それをどう活かすかについてはほとんど無限の可能性があるといえるのではないだろうか．

参考文献

1) Apple Computer, Inc.: Inside Macintosh, Vol. I～IV, Addison-Wesley (1985).
2) 萩谷：GMW ウィンドウシステムについて，bit, Vol. 19, No. 3, pp. 4-19 (1986).
3) 淵監修，古川・溝口共編：インタフェースの科学，知識情報シリーズ第5巻，共立出版，p. 206 (1987).
4) 角田，久野：流れて行かない Unix 環境の評価，情報処理学会第29回プログラミングシンポジウム報告集，pp. 95-104 (1988).
5) 上谷編：JStar ワークステーション，丸善 (1986).
6) Lampson, B. and Taft, E.: ALTO Users' Handbook, Xerox Palo Alto Research Center, p. 150 (1978).
7) Goldberg: Smalltalk-80: The Interactive Programming Environment, Addison-Wesley (1984).
8) 久野，角田：流れて行かない Unix 環境，情報処理学会論文誌，Vol. 29, No. 9, pp. 854-861 (1988).
9) 久野，角田：窓はねずみ無しでも操れるか？，情報処理学会第30回プログラミングシンポジウム報告集 (1989).
10) OFIS/DESK-EV 文法/操作書，2050-30010-20, 日立製作所，p. 176 (1986).
11) 坂村：BTRON におけるヒューマン・インタフェース設計手法，in 坂村編，TRON プロジェクト '87-'88，パーソナルメディア，p. 407 (1988).
12) 佐藤，久野ほか：CLU マシンのユーザーインターフェース，情報処理学会第29回プログラミングシンポジウム報告集，pp. 13-22 (1988).
13) Scheifler, R.W.: X Window System Protocol —X Version 11 Release 3, Massachusetts Institute of Technology (1988).
14) Sun MicroSystems Inc.: SunView Programmer's Guide, Sun Microsystems, Mountain View (1986).
15) Symbolics Inc.: Symbolics Lisp Machine Manual, Symbolics, Cambridge (1986).

（昭和63年12月21日受付）