Vol. 25 No. 8　　情　報　処　理　　Aug. 1984

総　論

画面エディタ†

角田博保††

1. はじめに

画面エディタについて利用者の視点を中心に総論する．おもに画面エディタに関して使われる用語の定義，画面エディタに特徴的な構成，機能について述べる．

コンピュータを会話的に使うことが近年一般化している．その利用に当たってよく使われるのがエディタ（＝会話的に使う文書編集システム）である．一般的にいって，エディタとは**利用者**が**端末装置**（入力装置＋出力装置）を介して，会話的にある**対象**に対してある**編集作業**をするのに使うプログラム，ということができる．本稿で扱う画面エディタは画面端末を持つ点が特徴であるが，さらに，いつも編集対象の一部を画面上に表示している点が重要である．したがって，タイプライタベースのエディタに画面端末をつないだというものは画面エディタとは呼べない．つまり，タイプライタを使うエディタがある入力に対してある出力を出すといった受動的なシステムとみなせるのに対して，画面エディタはいつでも内部状態を表示しているという能動的なシステムであるとみることができる．

エディタはその性質上，利用される環境(利用環境)に大きく影響されるものである．ここでは，エディタの利用環境の面から画面エディタの位置づけをしよう．エディタの利用環境は三つの次元に分けて考えることができる．それは，「だれが，何を，何を使って，編集するか」，に対応して，**利用者**，エディタで扱う**対象**，およびエディタが構築されている**外部構成**（システム環境）である（**図-1**）．

利用者には，計算機に対する知識の違い，打鍵速度の違い，技術力の違い，経験度の違いなどからさまざまな種類がある．熟練者と初心者とでは「よいエディタ」という概念がだいぶ違ってくる．ただ単に必要な作業をするのに要する打鍵数が少なければよい，というわけにはいかない．コマンド体系がどれだけ覚えやすいかとか，エディタを使うとどの位い疲労するかとかいった利用者の性質に大きくよった面からもエディタのことを考えなければならない．このような点を厳密に議論するために心理学的なアプローチがとなえられるようになってきた．

対象にもいろいろな種類がある．「Pascal」,「Lisp」などのプログラム言語から，表や図を含んだ文書まで多種多様である．エディタの内部構造は対象が持つ論理構造をうまく扱えるようになっていることが望ましい．特定の対象の論理構造にうまく対応した内部構造を持ったエディタ（＝構造エディタ）が開発されてきている．（構造エディタについては本特集の別解説を参照されたい．）

外部構成としても，どういう利用端末かということの他に，回線接続のしかた，回線速度，CPU の能力，記憶容量の違い，2次記憶の状況，さらにオペレーティングシステム（以下，OS と略す）の能力など種々の要因からいろいろの種類が見て取れる．

画面エディタの利用環境としては，画面端末を使い画面再表示をある程度高速にするための計算機能力が必要というように外部構成に対する制約が大きくなっている．対象と利用者に対しては個々の画面エディタでその扱いは大いに異なる．たとえば，Emacs は利用者＝熟練者，対象＝プログラムから文書まで，であり，Star のエディタは利用者＝オフィスの初心者（?），対象＝文書，である．個々のエディタについて考えるとき，どんな利用環境をねらっているかは大切な要点である．

画面エディタを構成面と機能面にわけ，**2**章と**3**章で総括的に述べることにする．画面エディタのとらえ方として，機能面・構成面というほかに抽象化のレベルの違いといった見かたもできる．一番抽象的なレベルを概念レベルと呼ぶことにしよう．つまり，画面エディタが論理的にみてどんな構成物を与え，それらに

† Screen Editor by Hiroyasu KAKUDA (Department of Computer Science, The University of Electro-Communications).
†† 電気通信大学計算機科学科

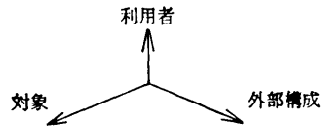

図-1 エディタの利用環境

対してどんな操作が適用できるかといったレベルである．基本コマンドの機能が何かを説明することになる．ついで，コマンドレベルである．利用者がどんなコマンドを使って編集対象に操作を与えることができるかというレベルである．最後に物理レベルとして，どのキーを叩けばどういうコマンドを指定することになるかといったレベルがある．この各レベルにいきわたるように説明を進めていくことにする．エディタに関するよくまとまった，大部な解説として文献 1) が有名であるが，そこで触れられている画面エディタに関する点については極力本稿でも説明していく．2章，3章は使う側の立場であるが，4章ではエディタを作る側の立場として，画面エディタの実現法について述べる．また，なるべく個々の事例には深入りしないようにしている．5章では今後の課題について述べる．なお，文献 1) には170あまりの文献のリストが載っているので興味のあるかたは参照されたい．

2. 画面エディタの構成

本章では使う側の立場からみた画面エディタの構成面について述べることにする．物理的および論理的に利用者の目に見えるもの（＝構成物）を順に述べていくことにしよう．

2.1 物理的構成

（1） 入 力 装 置

入力装置を通しておこなう作業には（1)対象の要素の入力，(2)コマンドの入力，および（3)編集する要素の指定，がある．これらの作業にほぼ対応するように，入力装置には（a)文字入力装置（鍵盤)，(b)ボタン装置(機能キー)，(c)位置指定装置（ジョイスティック，ライトペン，マウスなど）の3種類があげられる．画面エディタでは(c)に当たる，「画面上のここ」といった指定法が特徴的となりうる．

（2） 出 力 装 置（＝画面端末）

画面端末には，大きく分けると（1)タイプライタとほぼ同じもの (glass typewriter)，(2)画面操作を持った CRT，それに（3)高解像度のビットマップディスプレイがある．(1)の場合はほとんど画面エディタを作りえないのでここでは特に触れない．(2)が一般的なものである．この分類に入るものは基本的に n 行，m 列といった固定の画面サイズを持っている．画面上には長方形や下線といった形をしたカーソルが一つあり点滅している．カーソルは画面上の位置を利用者に知らせるために使われる．任意の場所にカーソルを移動したり文字を表示したりする機能は画面端末に基本的なものである．また，適当な場所の輝度を変えたり，明暗反転したりする機能を持つものもある．任意の位置に文字を挿入したり，削除したりできるような高級な機能を持った画面端末もある．半二重の環境で使うものには1画面分の情報を記憶してホストへ転送する機能やローカルエディット機能を持つものもある．

高解像度のビットマップディスプレイでは1画面に何行といった制限は変えられる．文字の大きさを変えることも任意であるし論理的な画面をいくつもこしらえることもできる．これらのことを円滑におこなうために高速な処理のハードウェアと大きなメモリを持っている．(値段も高価になる.）普通は TSS の端末としてではなく，ワークステーションにおいてよく利用されている．また，位置指定装置が使えるものが多い．

（3） 計算機構成 (configuration)

エディタが使われる計算機構成としては，時分割型 (time sharing)，自立型 (stand-alone)，および分散型 (distributed) がある．各々について，CPU 能力，記憶装置の収容力およびその速度，通信回線の能力（全二重/半二重，回線速度など）などの点を考えてみよう．

時分割型では OS の与える機能によってエディタのありかたが大きく左右される．通信回線として全二重か半二重かは大きな問題である．全二重では利用者からの入力が1文字ごとに読み込めるので細かな制御が可能になる．半二重ではまとめて読み込まれるし，キーを打鍵した時点で画面上に対応する表示がおこなわれるので，融通性がなくなる（エコーバックの問題)．回線速度も大きな問題である．1200 ボー以下の速度ではほぼ画面エディタとしてのよさが損われるであろう．また，OS が割り込み処理や，並列処理機能を与えているかどうかもエディタの機能に大きく影響してくる．CRT の画面コントロールコードを転送できることは最低条件である．(漢字端末を使おうとするときこの点はやっかいな問題を引き起している.）

自立型，分散型では画面が CPU 本体と直結しているので上記の問題はなくなる．そのかわり，各々固有

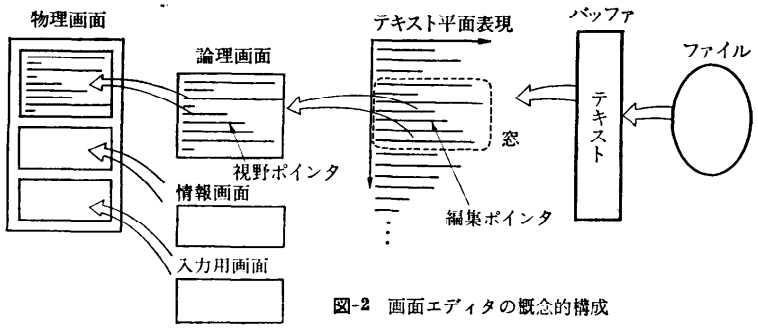

図-2 画面エディタの概念的構成

の実現上の問題がある.

2.2 概念的構成

画面エディタは図-2 で示す概念的構成をしている.

エディタが動く計算機環境において編集対象が保存されているところのことを**ファイル**と呼ぶことにする. エディタの編集対象はファイルからエディタ内の**バッファ**に取り込まれ, そこで編集作業を受けたあとでまたファイルへ保存されるとみることができる. この取り込まれた対象のことを「**論理テキスト**」(簡単に「**テキスト**」) と呼ぶことにする. 実際は, ファイルとバッファが同一場所であるエディタもあり得る. テキストは改行コードを含んだ文字列と表現される場合もあるし, 文字列 (行) の列と表現される場合もあるが, いずれの場合でも, 印刷された文書のように, 平面上にひろがった, 幅と長さを持ったものであるとみなされる. これをテキストの**平面表現**と呼ぶことにする. 概念的には, 利用者はこの仮想的なテキスト平面を編集対象と考えている. 平面表現には現在注目している場所をあらわすために**編集ポインタ**というものが設定されている. テキストはより細かな「要素」から構成されている. 「要素」には, 「文字」, 「行」, 「単語」, 「文」, 「段落」などがある. テキストを構成する要素に応じて, ポインタが指す単位が決まる. 最も基本的な単位は文字である.

平面表現は適当な大きさで切り取られて論理画面上一杯に配置される. 切り取られる平面表現上の領域のことを**窓**と呼ぶことにする. 論理画面上には編集ポインタの位置に対応して**視野ポインタ**がある.

論理画面は実際の画面 (物理画面) 上にいくつも作られうる (マルチウインドウと呼ばれている). 物理画面上では視野ポインタは普通**カーソル**であらわされる. カーソルは1点を強調する手段であるから, もっと大きい範囲を指定したいときにはその範囲の文字列の明暗反転とか輝度強調とかがおこなわれる.

画面上のカーソルは論理画面上の視野ポインタと同一視できる. また, 画面エディタの特徴は編集ポインタで指されるものが画面上で見えている点であるから, 大低の場合編集ポインタと視野ポインタとは同一視できる. したがって, カーソルが編集ポインタの位置をあらわしていると利用者には見えるのである. (以降, 特別に区別が必要である場合を除いてただ「ポインタ」ということにする.)

本稿では対象の持つ論理構造を反映した平面表現については扱わない. (構造エディタに関する解説を参照されたい.) 対象はテキスト平面といった平担な構造のみを持っているものとして扱っている.

物理画面上には, 論理画面が表示されるとともに情報 (メッセージ) 画面, 入力用 (テキスト/コマンド) 画面なども表示される.

エディタによってはバッファを一つ以上持つものもある. また, バックアップ用のバッファ, テキストの一部を記憶するためのバッファ, それをリストにしておぼえておく機構を持ったエディタ (Emacs の kill ring) もある.

この他に, 論理的な構成物として利用者側に見えるものに, コマンドの指定を自由に変える機能を提供するコマンド対応表やさらに適当な文字列/数を記憶できるレジスタやマクロをおぼえておくための領域とかも考えられる. 利用者が新たにコマンドを定義できるようなエディタではそのための言語機構を持っている. また, 前回実行された指令をおぼえておいたりある時点での内部状態をおぼえておいたりするための領域を持つエディタもある.

3. 画面エディタの機能

利用者がエディタに指令を与えるために使う言語はコマンド言語と呼ばれる. エディタの機能の多くはコマンドによって働かされる. コマンド言語によって利用者が画面エディタに指令を与え, 画面エディタはそれに対して画面表示を変えることで答える, というこ

とができる．本章ではまずコマンドの入力方法について述べ，ついで画面エディタの個々の機能について述べることにする．

3.1 コマンドの入力方法

画面エディタのコマンド言語の特徴は，なるべく短い打鍵数でコマンドを指定しようという点にある．これは画面を通じて利用者に簡単にフィードバックを与えることができるので，なるべく小さい単位でフィードバックをかけようとするからである．また，位置指定装置を使ったより直接的なコマンド指定法も特徴である．

コマンドは普通一つの「命令」といくつかの「引数」を「区切り」で分けて並べたという形をしている．コマンドには「命令」を置く位置によって3種の型が考えられる．前置型，後置型，および中置型である．一つのコマンド体系では統一したコマンドの型をとることが大切である．

利用者の入力手段としては，鍵盤の打鍵，機能キーの打鍵，位置指定装置での位置指定，およびボタンキーの打鍵がある．また，鍵盤の打鍵には制御キーを押しながら鍵盤を打鍵する（**二重打鍵**と呼ぶ）ことも含まれる．この場合，単独で打鍵した場合とは違った文字コードが得られる．以上の各々がコマンドを構成する最小単位（字句要素）に対応しているといえる．**図-3** にコマンドの構成を示す．

コマンドの字句要素の割り当てかたにはモードを設けるかどうかで二通りある．モードレスという方式では，打鍵キーに対応する字句要素の意味はどんな場面でも同一である．モードを設ける方式では，各モードによって打鍵キーの意味が変わる．利用者にとってはモードレスの方がよいが，モードレスでは打鍵キーの種類を多くすることが必要となり，また，あまりにも多くすると使いにくいということから，完全にはモードレスになっていないエディタが多い．モードを設ければ，モードの違いで打鍵キーの意味を変えることができるので，打鍵の種類を少なくできるが，利用者がモードを間違わないような対策が必要になる．

コマンドの入力方法で一般的なのは，Irons と Djorup とによる画面エディタの文献 2) で紹介されたもので，Emacs や Z でみられるように，キーを打鍵するとカーソルの位置に対応する文字が挿入（あるいは置き換え）されるというものである．つまり，各打鍵文字はそれ自身がコマンドで自分自身を挿入（置き換え）するものとみることができる．この方法では挿入（置き換え）以外のコマンドをあらわすのに機能キーや二重打鍵が使われる．機能キーは鍵盤のホームポジションから手を離さないと打鍵できないので，二重打盤の方がよく使われている．機能キーなどの数は限られているので，よく使うコマンドくらいしか機能キーに割り付けておけない．それ以外のものはコマンド入力画面上で指定することになる．それにはまずコマンド入力画面へ移行するためのコマンドを機能キーなどで指定する．カーソルがコマンド入力画面へ移ったあとで，実際のコマンド名を入力することになる．「引数」を伴うコマンドも同様にしてコマンド入力画面へ切り換えて指定することになる．コマンド入力画面上では打鍵文字は論理画面上と同様に挿入（置き換え）コマンドになるのでコマンド入力モードというモードを設けたというわけではなくモードレスであるといえる．このように制御キーや機能キーを使うことでモードレスに設計できるのである．ただし，キー打鍵を挿入にするか置き換えにするかを選択する機能を持たせるとそこにモードができてしまう．パーソナルコンピュータなどでなじみ深い BASIC の画面エディタはコマンド数は非常に少ないが，この方式のエディタであるといえる．

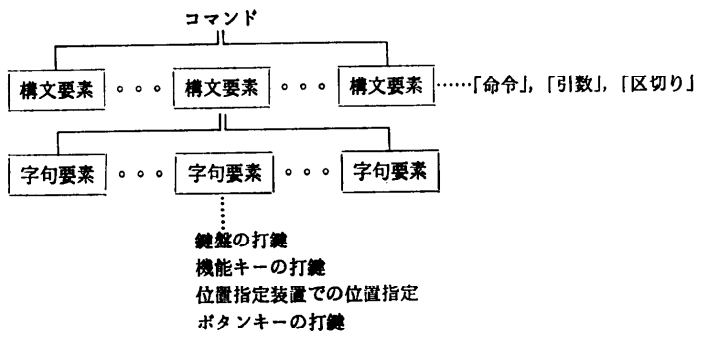

図-3 コマンドの構成

この方式に反して，バークレイ版 Unix の画面エディタ vi はモードエディタの典型といえる．vi を起動するとエディタはコマンド入力モードにいる．たとえば，「a」と打鍵すると vi は挿入モードに入り，続けて打鍵した文字が画面に挿入されていく．エスケープキーを打鍵することでコマンド入力モードに戻る．空白文字の打鍵は空白の挿入ではなくカーソルが右へ1文字移動す

る．このように，二重打鍵をしなくても大低のコマンドを入力することができるが，利用者はいつもどのモードにいるかを気にしていなければならない．画面エディタでは一つの入力に対して何らかの反応（画面表示の変化）をして利用者にフィードバックをかけるべきであるが，vi の場合「a」を打った瞬間には画面上にはなんら変化が起らない．少なくともモードの表示くらいはつけるべきであろう．

ビットマップディスプレイとマウスを使ったエディタではコマンドの入力方法に特徴がある．文字挿入（置き換え）については Irons の方法が使われるが，それ以外のコマンドの指定法は特徴的なものである．画面上の位置指定にはマウスが活用される．マウスにより指定された座標を「引数」として，マウスのボタンによって「命令」を与えるという形のコマンドが使われる．また，ビットマップディスプレイを活用した，メニュによるコマンド指定も特徴的である．画面上にメニュというコマンド名が並んだものを表示しておき，それをマウスで指定してボタンを押すことでコマンドを指定するものである．最近では必要なときだけメニュを表示させる方法も採用されている．このようにしてコマンドの指定に対して鍵盤を何文字分も打鍵するということはなくなるわけである．ただしメニュ方式が万全かというと，いちいちメニュから選ぶというよりはその名前を打鍵した方が速いというエディタ熟練者もいることから，とりあえずは初心者向きであるということができるであろう．

半二重端末を使った画面エディタでは，いままで述べた方法は使えない．コマンド文字列をコマンド入力画面上に入力して送信キーを押すというコマンド入力方法が昔からあるものである．この場合カーソルはコマンド入力にとられるので，ポインタの位置を利用者に知らせるためには高輝度化や明暗反転が使われる．文字の挿入なども画面上に直接することはできない．ローカルエディット機能を持った端末では画面の修正は端末上でじかにすることができるのでその機能を使った入力方法が最近では一般的となっている．この場合1画面分修正後，1画面分のデータを一括してホストへ送り，前回との内容の比較から利用者が与えたコマンドを読み取るということになる．また，機能キーがおおいに利用される．

3.2 基本的機能

画面エディタの基本的機能とは論理的構成に対して何らかの操作を与えるものである．これは，概念的にみたエディタの操作（論理的操作）に対応している．この操作はコマンドの基本操作ということができる．画面エディタにとって基本的な操作には，視覚化（画面の表示），ポインタの移動，テキストの編集，がある．

3.2.1 視 覚 化

画面エディタではテキスト平面を論理画面へ写す操作が必須である．この操作はじかにコマンドで指定されることもあるが，大低は **3.2.2** と **3.2.3** の操作に付随して自動的に呼び出されると考えられる．つまり，ポインタの移動や，テキストの変更によって画面の再表示が必要となったときに呼ばれる．画面には常にバッファの内容と一意に対応する表示がなされなければならない．

論理画面上への写像は，論理画面の大きさから対応するテキスト部分を切り出す操作と，それを画面に合わせて整形する操作とに分けてみることができる．テキスト平面を論理画面上に写像するには二つの方法がある．一つはテキスト平面上の矩形をそのまま論理画面上に配置するもので，もう一つはテキストの各行を論理画面の幅で折り曲げてできた矩形を配置するものである．前者はテキストが行の列という構造をしている場合，後者はテキストが文字の列という構造をしている場合に対応している．**図-4** にその状況を示す．

物理画面上にいくつもの論理画面を取り（マルチウインドウ）各々が部分的にかさなっている場合は，上の部分が表示される．画面の上下はマウスによって指定されるのが普通である．

3.2.2 ポインタの移動操作

ポインタを動かす操作には，相対的・絶対的な位置指定，内容による位置指定，画面上での直接位置指定

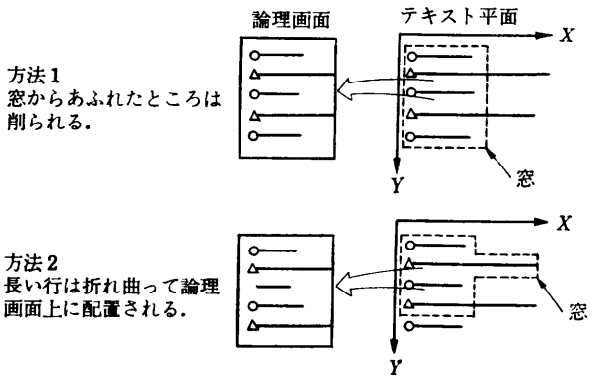

図-4 論理画面への切り出し方法

がある．また，画面上の位置を記憶しておいて，後からその位置へポインタを動かすという操作もある．ポインタは移動しなくても，対応するカーソルの実質移動が起る操作として窓の移動操作がある．さらに，バッファをいくつか持つエディタではバッファ間の移動という操作がある．ポインタは必ず窓の中になければならないので，ポインタの移動に伴って窓の移動が起る場合がある．以下，画面エディタで特徴的な操作について述べる．

（1） 相対的な位置指定

画面エディタの場合目標をとらえるのに，まず目標が画面に入るように窓を動かし，ついでポインタを上下左右に動かしてその目標の場所へ持っていくというやりかたが一般的である．

ポインタをテキスト平面上で左右方向に動かす場合は，テキストの内部構造によって，画面の右端，左端での動きが違ってくる．テキストが文字の列である場合は行は改行コードによってわかたれているのでポインタはただテキスト上を動くだけ，つまり，行の右端で1文字先へ動かすと次の行の先頭へ動く．テキストが行の列という場合には，画面の右端で1文字先へ動かしてもポインタが動かないという方法とポインタは動くがカーソルは動かないという方法（画面と窓とがずれるかなり例外的な事態）さらに窓自体が動くという方法がある．左端についても，前の行へいく方法とそこにとどまる方法とがある．テキスト平面の上下方向への移動は，論理画面上での真上（真下）へ移動するものと，テキスト平面上での真上（真下）に移動するものとにわかれる．大低の場合は両方とも同じ結果になるが，1行が長すぎて折れ曲って画面に表示されている場合には違う結果となる．画面の端から上下方向にはずれるようなポインタ移動指令は受け付けないという方法もあるが，ポインタが移動する場合には窓からはずれるので窓の移動が起る．その新たな行を含むような最小の移動をする方法とポインタが窓の中央へくるように移動する方法とがある．画面上ではスクロールあるいは一斉書き換えが起る．このほかに，直前（直後）行の先頭への移動，同一行内では行の先頭，行の末尾への移動がある．また，論理画面上の絶対位置への移動というものもある．このほかに，単語単位，パラグラフ単位，ページ単位での移動をする操作もよくみられる．以上の操作は機能キーや二重打鍵によって大低は1打鍵で指定できるようになっている．

（2） 内容によるポインタの移動（探索）

与えられた「パターン」によってテキスト平面上を探索して，見つかればその位置へポインタを動かす操作である．パターンによる探索の場合は編集範囲（**3.2.3** 参照）を同時に指定しているともいえる．同一画面上で見つかればそこへカーソルが移動し，画面からはずれる場合は窓が移動していく．相対的な位置指定操作の場合に反して，探索の場合はカーソルを見のがしやすい．そこで，カーソルのかわりに見つかった文字列を高輝度で表示するとか明暗反転させるとかして，利用者に目立つようにするという工夫がある．「文字列」といった塊ではなく，逐次1文字ずつ連結しながら探索するというものもある（Emacs のインクリメンタルサーチ）．

（3） 画面上での直接位置指定

ライトペンを使ったエディタでは直接画面上の位置を指定することができる．マウスは直接位置ではないが，マウスの動きに合わせてカーソルを相対的に動かすことができる．マウスは手の動きによって画面上のカーソルが動きそれをみて手にフィードバックをかけて意図した位置へカーソルを持っていくといった複雑なフィードバックを扱うものであるから，反応の遅れがないように実現することが必要である．位置指定装置で画面上の1文字を指定するのは手技的に難しいので，もう少し大きな要素単位での指定法（単語など）が使われる．この場合もポインタの移動とともに範囲が指定される．

（4） 窓の移動

ポインタを移動するといった指定ではなく，窓を画面いくつ分移動するといった指定のしかたもある．自動的にポインタの移動が伴う．数行分窓を動かす場合はポインタの移動が起らないこともある．

3.2.3 編集操作

画面エディタでは全テキストにわたってxをyに換えるといった一括型の操作よりも逐次画面で編集結果を確認する操作の方がよく使われる．一般的なエディタで必要となる操作に比べてより簡素なもので十分であるといえよう．

（1） 範囲の指定について

削除，置き換え，転送，複写といった操作ではテキストの一部分（＝範囲）を指定することが必要である．範囲の指定法には，(1)現在のポインタ位置を始点とし，それに相対位置を加えたものを終点とする方法，(2)パターン照合を使って指定する方法，(3)種類を与えて指定する方法（単語，行といった種類で範囲を

決める)，(4)小さな範囲を初めに指定しておき，ついでその範囲を拡大するといった方法，(5)ポインタ移動操作を使って始点終点を与える方法，などがある．範囲を利用者に明確にするために対応する文字列が強調されたりする．画面エディタでは，(1)と(3)を組み合わせて，1文字削除，1行削除といったコマンド(打鍵数1で指定可能)を使うことが多い．また，(4)や(5)にあるように範囲の指定を会話的におこなうものも画面エディタのコマンドの特徴である．(5)の方法ではポインタを移動している最中に利用者にフィードバックをかけてどこまで指定したかをわからせるようにしている画面エディタもある．その場合指定された行の輝度が上がり，範囲が拡張されていく状況が利用者によくわかるようになっている．

(2) 挿入操作・置き換え操作

3.1 でも述べたように画面エディタではタイプライタで文字を打ち込んでいる場合と同様に，1文字単位の挿入/置き換え操作が基本である．タイミングというものはエディタにとって大切なものであるから，鍵盤でキーを打鍵した瞬間にその文字が画面上に挿入されるということ，つまり，打鍵したということをすぐにフィードバックできるように実現することが必要である．

(3) 削除・転送・複写

削除操作ではテキストのある特定の範囲を取り去る．削除されたテキストの部分はのちに利用可能な削除用バッファに保存される場合が多い．ポインタが指している1要素を削除するコマンドが基本的である．

ある指定の範囲に対する転送と複写の違いは転送ではもとの部分はなくなるのに対して，複写ではコピーがとられるという点である．この操作は範囲の指定と行く先の指定とが必要となるので利用者がうまく指定できるような具体的な操作を設計するには工夫がいる．削除と挿入の組み合わせがわりとうまく指定できると思われる．複写の場合は削除バッファのかわりに複写バッファといった別のものを使うエディタもある．

3.3 補助的機能

エディタの機能としてテキスト編集上は必要ではないがあると便利なものがいろいろある．

エディタの信頼性を向上させるための機能としては，既に実行したコマンドをあとで取り消す機能，実行中のコマンドを取り下げる機能，いままで実行されたコマンドの列をあとで取り出す機能，コマンド呼び出し列を作っておき一括して実行する機能，エディタの内部状態を記憶しておきあとでその状態へ戻す機能などがある．

使いやすさを向上させる機能としては，前回与えたコマンドをもう一度実行する機能，ヘルプ機能などがある．

専用化機能としては，システム変数を変える機能，個人用にシステム変数の値を記憶しておき，エディタ起動時に暗黙に実行する機能(プロファイル機能と呼ばれる)，キーと字句要素との対応表を変更する機能(コマンドの再割り付け)，新たなコマンドを定義するための機能などがある．

対象向け機能はたとえば，Lisp のかっこのバランスをとるコマンドなどである．これに関しては本特集の構造エディタに関する解説を参照されたい．

さらに，コマンド言語そのものを新たに定義する機能，エディタの本体自体を利用者が改良できるようにした機能もある．

以上の機能の多くは既存の画面エディタで採用されている．特に画面エディタにとって有効であったものをいくつか挙げてみることにしよう．まず，ヘルプ機能である．画面エディタでは画面を活して一度に多くの情報を表示して利用者に示すことができる．特にマルチウインドウの機能を備えた画面エディタでは十分うまく機能するであろう．システム変数を利用者が変更できる機能も画面を使うとうまく実現できる．いくつかあるモードのどれを選択するかということも画面上で指定すればしやすいし，どのモードであるのかも知りやすい．使いやすさを向上する機能にはコマンドの実行時間が長いものは何がエディタ内でおこなわれているのかを定期的に画面に表示して利用者にフィードバックをかけるようにしたものもある．

図形を扱った機能や右そろえといった清書系にかかわる機能はビットマップディスプレイを使った画面エディタでは特徴的である．図形の指定にはおもにマウスが使われている．作図された図形はビットイメージで保存されるものが多い．ビットマップディスプレイ上の一つの実現例としては本特集の別解説を参照されたい．

4. 画面エディタの実現法

エディタの実現において重要となる点は，信頼性と高速性である．また，TSSで使う場合はほかのジョブを圧迫しないこと，自立型では記憶容量の制約がある

こと，などが挙げられよう．画面エディタの特徴としてつねにバッファの内容と一致した画面表示をしなければならない点からいかに高速にバッファの変化に対応できるかが重要である．

信頼性向上のためには，なるべく高級言語でエディタを記述した方がよいが，それは高速性追及と密接にかかわってくる．少なくともコマンド解析部は高級言語で書く方がよいであろう．画面エディタでは画面表示の部分が一番実現において苦労する個所である．これをOS上で複数のプロセスとして実現できるならば実現方法も簡単で，信頼性のあるものとなろう．

高速性においては特に画面の再表示アルゴリズムが問題となる．画面再表示アルゴリズムは二つに分類される．画面の書き換えとバッファの書き換えを同時にやるか，分けるかである．

前者はバッファに対する書き換えが起ったら画面上にも同じ書き換えをほどこせばよいので作成手順は簡単であるが，対応する書き換えを画面上に反映させるのがむずかしい場合には大変てこずるし，プログラム全体の見通しがわるくなるといった欠点がある．

後者はバッファの書き換えは画面を気にせずにやり，ときおり画面書き換えルーチンが呼ばれ，それは昔のバッファとの違いを調べてつじつまが合うように画面の書き換えをするというものである．この方法のよい点はプログラムがすっきりとして，信頼性が上がり，保守もしやすくなる点である．欠点としては効率が多少わるくなる点であるが，これはアルゴリズムを工夫することによってかなり克服できる．この場合，「文字列間の相違検査法」に基づいて，前画面から現画面を作るための最小制御コード列を見つけ出す方法を使うとデータの転送量が少なくて済み即応性がよくなる．

どういう端末装置を使うかに対するエディタの設計思想は大きく分けて二派がある．ある特定の端末を使うことに決め，その端末の機能を最大限に使えるようにしようというものと，なるべく多くの端末で使えるようにしようというものである．前者にはワークステーション上で動くエディタが挙げられ，後者にはTSSで使われるエディタが挙げられる．

不特定多数の端末で使えるように考えてあるエディタでは仮想画面端末といったものを定め，そこでの仮想画面操作を決めておく．仮想操作を実現できる機能を持った端末においてはそのエディタがすぐ使えるようになる．この機能を持ったエディタで有名なものにバークレイ版Unixの画面エディタviがある．viでは「termcap」という名のファイルに仮想画面操作に対応する制御コード記述を個々の端末ごとに入れておくことになっている．termcapは端末属性データベースということができる．viの起動時に登録された端末名を指定してやれば，viはtermcapから対応する制御コード列を探してきてそのコード列を発生するので，その端末でviが使えるようになる．TSSで使われるエディタは今後このような実現法がとられていくであろう．

5. おわりに

一般的な画面エディタはどういうものかを述べてきた．ここで，これからの画面エディタについて展望することにしよう．時分割型の画面エディタはほぼ完成のいきにいるのではないかと思われる．ただし，一般に普及している画面端末を使うという条件付きの話である．修正したいものをこれだといってたやすく指定できるかというとまだなかなかそうはいかない．やはり，端末自体の能力の大きい高解像度のビットマップディスプレイと位置指定装置のマウスを使ったエディタがこれからの主流となるであろう．この型のエディタについてはまだ研究が始まったばかりである．今後の進歩が期待される．

また，あるエディタが使いやすいかという議論はもっと科学的であるべきである．あまりに主観的な議論が横行しているように見受けられる．一つコマンド体系を取ってみても客観的な理由づけが必要である．それにはまず，本当に人間にとっての使いやすさを科学的に計測できるような妥当性のあるエディタモデルの出現が望まれるのである．

参考文献

1) Meyrowitz, N. and van Dam, A.: Interactive Editing Systems: Part I, II, ACM Computing Surveys, Vol. 14, No. 3, pp. 321-416 (Sep. 1982) (訳書：石井，瀬川：対話型編集システム：第1部，2部，bit別冊，共立出版，1983. 12).
2) Irons, E.T. and Djorup, F.M.: A CRT Editing System, CACM, Vol. 15, No. 1, pp. 16-20 (Jan 1972).
3) Gosling, J.: A Redisplay Algorithm, Proceeding of Symposium on Text Manipulation, pp. 123-129.

(昭和59年6月22日受付)