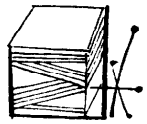
論　説



マン・マシン・インタフェースとしてのエディタ†

奥　乃　博††

1. はじめに

「エディタはテキストを編集したりプログラムを作成するのに用いるツールである．したがって，TSS 環境やパーソナル・マシンの下で計算機を用いるときには重要なツールとなる．しかし，バッチ環境や制御用では不用な代物—少なくとも，その有用さは発揮できない．」という意見が EDP 界では支配的なのではないだろうか．例えば，あの Unix* でさえ，Ed. 6 までは行エディタ ed しか提供されていなかった．

それに対して，「エディタはマン・マシン・インタフェースの根幹をなす．」というのが筆者の考え方である．マン・マシン・インタフェースとして提供すべき機能は人間どうしの会話のエッセンスでなければならない．そのような機能として次の5項目が挙げられよう．

① 画面表示の双方向性…ユーザの入力のみならず，計算機の出力も再利用可能．

② 主導権をユーザと計算機が分かち持つ…計算機が主導権を取る例としては，コンサルタントや助言・説明機能がある．

③ 非厳密性…代名詞の使用，エラー自動修正，等文脈を利用して厳密なものとする．

④ 拡張性…カムタム化，学習によるユーザのプロフィールの自動作成など．

⑤ マルチ・メディアを用いた会話．

エディタは上記の全項目を提供しているわけではない．エディタと関係するのは第一項の表示データの双方向性である．つまり，入力ルーチンをエディタにすればこの双方向性が容易に実現できる．

エディタに関しては，エディタがエディットすべきデータをどう見ているかというエディタ・モデルの問題がある．エディタが提供しているコマンド体系はこのエディタ・モデルに従って決まってくる．エディタ・モデルは，「**構造モデル**」と「**テキスト・モデル**」に大別できる．構造モデルではデータが構造体であると見るのに対して，テキスト・モデルではデータを文字列であると見ている．この区別は必ずしも排他的なものではない．Emacs[1] や Z[2] というエディタは基本的にはテキスト・エディタであるが，構造コマンドも提供している．例えば，テキストのエディットにおける構造とは単語，文，パラグラフであり，Lisp プログラムではS式（アトムやリスト），関数定義などである．筆者の経験から判断すると，「**テキスト＋構造**」が一番妥当なエディタ・モデルである．Emacs や Z が優れたエディタであるのもこの点に1つの理由があると考えている．

エディタがマン・マシン・インタフェースの中心的役割を果すために重要な要因として応答速度がある．いくら機能の優れたエディタであっても応答速度が遅ければその価値は半減する．エディタ処理系の実現者にとっては，機能の豊富さと応答速度の早さという相反する要求をどう解決するかが重要な課題となる．

本稿では，「マン・マシン・インタフェースの核はエディタである」という立場から見たエディタについて議論する．第**2**章では，エディタの高度な応用例について報告する．第**3**章では，エディタ・モデルとして「テキスト＋構造」が妥当であることを Emacs の実際の使用法から実証する．第**4**章では，応答速度を改善するために Z で行っている再表示のための工夫とその有効性を報告する．

2. エディタの高度な応用例

本章では，マン・マシン・インタフェースの核として使用されているエディタの例を報告する．

2.1 コマンド・エディタ

ある種のエディタでは，エディタのウインドウからコマンド・プロセッサが起動でき，コマンド・プロセ

† Editor as a Kernel of Man-machine Interface by Hiroshi G. OKUNO (Musashino Electrical Communication Laboratory, N. T. T.).

†† 日本電信電話公社武蔵野電気通信研究所

* Unix は Bell 研の商標です．

ッサからの出力がそのウインドウに表示される．例えば，Zを用いたセッション・マネジャや Unix-Emacs を挙げることができる．これらのコマンド・エディタは本格的なエディタそのものであるので，エディタとコマンド・プロセッサとの交信が重たく軽快さが欠けるという欠点がある．

本節で紹介するのは TOPS-20 (Dec-20の OS) 上で実現されているコマンド・エディタ CEDIT である．CEDIT の機能はユーザの入力したコマンドをエディットするためのものに限定されている．ユーザが入力したコマンドは輪状バッファに格納されこのバッファ上でカーソルを進めてエディットすべきコマンドを選択する．CEDIT のエディット・コマンドは Emacs 風である．CEDIT は上矢印（^）をコマンド・プロセッサに与えるか，あるいは，特定の制御文字を入力することによって起動される．このときにはカーソルは直前に入力したコマンドを指している．特定の制御文字の入力はコマンド・プロセッサのいかなる時点でも有効であり，その時点で入力中のコマンドをカーソルが指すことになる．この制御文字の設定はユーザが自由に行うことができる．CEDIT はコマンド・エディタとして必要な機能しか提供していない Emacs の部分集合であるが，処理系はコンパクトでコマンド・プロセッサに容易に組込むことができ，軽快で重宝するエディタとなっている．

2.2 言語プロセッサのトップ・レベル入力

会話型言語では，①以前に使用した入力の再利用，②計算機からの出力の利活用，③以前の入出力をエディットして再利用，という機能があると非常に使いやすくなる．上記の機能は言い換えると計算機とのやり取りが双方向であるということに他ならない．Interlisp[3] では Programmer's Assistant という機能によって上記の機能の一部が実現されている．これらの機能を完全に実現するためにはトップ・レベル入力がエディタになっていなければならない．この機能が実現されている Lisp 処理系もある．

MacLisp や Interlisp では Emacs をトップ・レベル入力として使うことができる．この2つの Lisp に対する Emacs のライブラリは LEDIT, INTER と呼ばれている．Emacs の主な使用法としては，①カーソルのあるS式の評価，②カーソルのあるS式を評価し，その値をエディタに表示，③トップ・レベルでのやり取りを記録した履歴のエディットがある．もちろん，Lisp 内で定義された関数の定義を Emacs に与えてエディットすることができるのは言うまでもない．

Zetalisp (Lisp Machine Lisp)[4] では，通常のトップ・レベル入力は前節で述べた CEDIT と類似の機能を提供している．さらに，Emacs を拡張した Zmacs（マウスの使用が主な拡張点）をトップ・レベル入力として使用することもできる．この場合には，現在の入力の始まりを示す特別のマークが表示される．もし，このマークの後ろにS式が入力されるとそのS式が評価され，その値が表示される．同様のトップ・レベル入力は Interlisp-D でも提供されている．

2.3 データベース・マネジャ

ファイル・システムの管理を行うときに，ファイルの属性（ファイル名，作成者，作成日時，最近アクセスした日時，等）を知るだけでは不十分で，その中身を見てからどう処理するかの判断を下したい場合がある．このような目的で使用されるのがディレクトリ・エディタ DIRED である．

DIRED は Emacs の1つのライブラリとして実現されている．DIRED に入ると，指定したディレクトリに存在するファイル・リストがウインドウに表示される．このウインドウは変更できないが，カーソルの移動は自由にできる．カーソルのあるファイルに削除の印を付けたり，逆に削除の印を消したりすることができる．削除の印が付いたファイルは DIRED から抜けるときに一括して削除される．

ファイルの中身を調べる検査モードにはEを入力すると入る．すなわち，カーソルのあるファイルが新たなウインドウに読込まれてエディット可能となる．これは，DIRED の中から Emacs を再帰的に呼出すことによって実現されている．検査モードでは通常の Emacs と同じコマンドが使える．例えば，バージョン番号の異なる2つのファイルの比較を行ってその差異を確認することができる．DIRED を用いると，必要なファイルを中身を調べずに削除してしまうといった誤まりを避けることができよう．

電子メールの管理もメイル・リストに対するDIRED を用いてファイル管理の場合と同様に行うことができる．電子メールから呼ばれた Emacs は DIRED の他に種々な機能を提供している．例えば，返事を送るときには画面が2つのウインドウに分割され，片一方のウインドウに返事を出すべき元のメールが表示される．

Prolog を始めとする論理型言語ではプログラムもデータもすべて節形式で内部データベースに格納され

る．内部データベースを操作するために，Prolog 処理系では種々な述語(コマンドのこと)一節の表示，節の挿入・削除等一が提供されている．Prolog では格納されている節の順序が実行上のセマンティックスを持っているので，節の順序が異なるとその動作も変ることがある．したがって，新たな節を挿入するときには挿入すべき節の位置に注意しなければならない．エディタを用いて内部データベースが操作できると，行おうとする操作が視覚的に理解できる．筆者は画面エディタを論理型言語の支援ユーティリティの中心に措えようという提案を行い，実際に Emacs を用いて実現している[5]．

3. エディタ・モデル

Emacs や Z は前述したように「テキスト＋構造」というエディタ・モデルを持っている．このモデルはプログラム・エディタとして 100 点満点ではないが，95 点は取れているという報告がある[3]．本章では，エディタが実際にどのように使用されているかという統計データをもとに，「テキスト＋構造」が妥当なエディタ・モデルであることを論ずる．また，構造エディタの問題点についても述べる．

エディタ使用の統計データが採られているエディタは Emacs[6]，Z[7]，Unix-Emacs[8] である．いずれの場合にも構造コマンドの使用頻度は全入力の 5% 程度である．全入力の約半分はエディタ・コマンドであるから，構造コマンドは全エディタ・コマンドの 10% 程度と，使用頻度は小さい．

Emacs のデータの被験者は大部分が Lisp プログラマであり，かつ，会話的にプログラムを作成している．このような環境では構造コマンドの使用が極端に少なく，文字単位の基本コマンドを根気よく連続して使用する傾向にある．また，対となるコマンド（例えば，次の画面へ移るコマンドと前の画面へ戻るコマンド）に対して割当てられたキー系列の長さが異なるときには，キー系列の長いコマンドの使用が抑制される．このように，会話型プログラミングにおいては，エディタの使用法は反射的であり，構造を無視して文字単位のコマンドで所望の位置までカーソルを移動させたり，不用なデータを削除している．つまり，エディタ・モデルとしてはテキストが中心であることが裏付けられている．

反射的なエディタの使用法は会話型プログラミングで特に顕著であると考えられる．構造コマンドを使用したり繰返しの回数を引数として指定するためには，どうエディットすべきかを考えなければならない．しかし，会話型プログラミングでの焦点はプログラムの創造にあり，どうエディタを使うかにはない．この結果がエディタの反射的使用となるのであろう．

構造エディタを使っていて困るのは，①エディタで行った変更がファイルにすぐには反映されないこと，②構造を劇的に変更したいときには，構造コマンドを使うよりは入力し直した方が早いこと，③アトムの名前を間違ったり，空白を忘れたために 2 つのアトムが 1 つのアトムとなったときには，再入力しなければならないこと，などである．③については，INTER では構造エディタから Emacs が呼出せるので，Emacs でミス・スペルの修正や空白を挿入することができる．構造エディタとテキスト・エディタの最大の違いは，特に Lisp エディタで言えることであるが，構造エディタが行エディタ的であるのに対して，テキスト・エディタが画面エディタである場合が多いということであろう．構造エディタの方がテキスト・エディタよりも括弧のミス・マッチが少ないと言っても，行エディタ的な使い方しかできなければ余り使われないと思われる．最近発表された Interlisp-D の構造エディタは，画面エディタであり，かつ，その中からテキスト・エディタも呼出せるようになっているので，理想的な構造エディタの雛型と言えるのではないだろうか．

4. 画面表示の高速化

いくら機能が優れたエディタであっても応答速度が遅ければその機能は半減し，宝の持ち腐れとなってしまう．画面エディタでは，画面の表示に変更が生じたり，カーソルの移動があった場合には，速やかに画面を表示し直さなければならない．これは，再表示のための計算機から端末への転送量をいかに減らすかにかかっている．このアルゴリズムは**再表示 (redisplay) アルゴリズム**と呼ばれている．本章で紹介するのは，挿入やカーソル移動に対していかに早くエコー・バックするかという工夫である．

テキストの入力には挿入モードと置換モードがある．Emacs は通常挿入モードであるのに対して，Z は置換モードとなっている．Z で置換モードが暗黙のモードとして採用されたのは，エコー・バックを高速にするためである．エコー・バックはエディタが行うのが普通であるが，Z では OS (TOPS-20) を変更して，置換，カーソル移動については OS レベルでエ

コー・バックするようになっている．この方式を**即時エコー (immediate echoing)** と呼んでいる．即時エコーの効果は統計データを下に確認されている[7]．

Z がどのように使われているかという統計データによると，次のようなことがわかっている．

① 非コマンド文字（全体の 43.8%）のうち，置換モードで入力されたのが 81%，挿入モードで入力されたのが 19% である．

② 挿入モードのうち，86% が行末に挿入されている．

③ 挿入モードでしか仕事をしないユーザでは 90% 以上が行末に挿入している．

④ カーソル移動コマンドは全体の 45.1% である．

置換モードでの入力とカーソル移動コマンドはディスプレイの端に来ない限り即時エコーできる．実際に即時エコーされたのは全入力の 75% である．また，挿入モードで行末に挿入された場合も即時エコーが可能である．もし，これも即時エコーの対象とすると全入力の 82% が即座に出力されることになる．

Emacs でも行末に入力された場合には，再表示アルゴリズムを介さずに即座にエコー・バックするようになっている．そのために，新たな行を挿入するときには，まず空行を作ってからその行の入力を行うように勧めている[1]．ただし，Emacs では Z のような OS が行う即時エコーは採用していない．

5. おわりに

本稿では，エディタの高度な応用，「テキスト＋構造」というエディタ・モデル，高速画面表示について述べた．これらの議論を通じて「エディタはマン・マシン・インタフェースの中心的役割を果す」という考え方を明らかにした．筆者は Emacs を愛用しており，Emacs を使用する中からこの考え方に到達した．大学や研究機関では会話型プログラミングが日常化しつつあるのでエディタ，特にテキスト・エディタが注目されつつある．しかし，大部分の計算機はバッチあるいは制御用として使われておりエディタとは縁遠い世界にある．最近，ユーザ・インタフェースを改善しようという目的で知識工学が注目を集めているが，その前にエディタを応用したシステムがまず登場するのではないかと期待している．

参考文献

1) Richard M. Stallman: EMACS Manual for TWENEX Users, AI Memo 555, MIT (Oct. 1981).
2) Steve Wood: Z—95% Program Editor, Proc. ACM SIGPLAN/SIGOA Symp. on Text Manipulation, SIGPLAN NOTICES, Vol. 16, No. 6 (Jun. 1981).
3) Warren Teitelman et al.: The Interlisp Reference Manual, Xerox PARC (1978).
4) David Moon, Richard M. Stallman and Daniel Weinreb: Lisp Machine Manual, Fifth Edition, LMI (Jan. 1983).
5) 奥乃 博：述語型言語の支援ユーティリティの検討，情報処理学会人工知能と対話技法研究会資料 26-3 (Jun. 1982).
6) 奥乃 博：画面エディタ Emacs のユーザ特性について，第 25 回プログラミング・シンポジウム報告集 (Jan. 1984).
7) Steve Wood @ YALE-RES, Z editor usage statistics, Yale 大学のファイル (Oct. 1982).
8) Steve Zimmerman @ CCA-UNIX, EMACS Keystroke statistics, 同上 (May 1982).

（昭和 59 年 4 月 10 日受付）