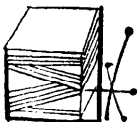
論　説



次世代エディタ†

小　島　啓　二††

1. はじめに

エディタは常に「好き嫌い論争」の種になるプログラミング・ツールである．これらの論争は大抵，各個人の趣味の問題に落ち着いてしまい，そもそもエディタを客観的に評価するのは不可能なのではないかと疑いたくなる．

エディタの客観的評価の困難さは，編集という行為自体が個人のプログラミング・スタイル，ひいてはその思考様式と深いかかわりを有していることに根ざしているように思う．

実際，対話的プログラミングの普及に伴い，プログラマは多くの時間を「エディタを使いながら考える」ことに費やしている．すなわちソフトウェアの世界では，従来紙と鉛筆を用いて机上で行っていた作業が，エディタを用いた端末での作業へと代わりつつある．一方各個人は，「紙と鉛筆を使いながら考える」際の個性的なスタイルをそれぞれ長期間かかって築きあげており，「エディタを使いながら考える」場合にもこれらのスタイルが強く反映するのは当然である．筆者にはこのような各自の思考様式と，エディタとの適合性の良否が，エディタの評価に大きな個人差が生じる要因の1つに思えてならない．したがって趣味の相違として片付けられることが多いエディタのデザイン上の問題も人間の思考様式の定式化にまで遡れば，客観的かつ意味のある議論が可能と信じる．

思考様式の定式化に基づいてエディタのデザインを見直すというのは魅力的なテーマではあるが，到底筆者の力の及ぶところではない．そこで本稿では客観性を追求するよりはむしろ，筆者のエディタに対する主観的指向をできる限り明確化することを目標としたい．具体的には，次世代のエディタに対する要求仕様を述べるという形で筆者のエディタ観をまとめることにする．

2. 次世代エディタ要求仕様

既に述べたように，筆者はエディタを思考補助のためのツールと考える立場に立っている．そして今日に至るまで人間の思考を助けてきた紙と鉛筆に迫れるようなエディタこそ，次世代のエディタとしてふさわしいと考える．この観点から，ここで仕様を述べる架空の次世代エディタを PAPILS (Paper and Pencil's Successor) と呼ぶことにする．

まずはじめに PAPILS のハードウェア構成について簡単に述べよう．

2.1 PAPILS のハードウェア構成

PAPILS には多様な入出力形態，例えば手書きによる入力，音声による入出力，画像による入出力等が可能であることが望ましい．特に手書きによる入力は，その自由度からみて必須と考える．そこで PAPILS のテキスト表示用ディスプレイは，ライトペン風の入力装置を用いてその上に直接文字や図形，あるいは種種の制御情報などを書き込めるものとする．この表示装置を PAPILS スクリーン，入力装置を PAPILS ペンと呼ぼう．そしてこの2つに通常のキーボード入力装置を加えたシステムを PAPILS の基本ハードウェア構成とし，他の各種入出力装置はオプションとして PAPILS スクリーンに接続可能とする (図-1)．

2.2 PAPILS の機能仕様

筆者の経験によれば，わかり易いエディタは使い易い．言葉を変えれば，ユーザからみた概念モデル (user conceptual model) が明解なエディタほど操作性も高いように思う．

実際，便利そうなコマンドを豊富に揃えることに意を用い過ぎて概念モデルの難解化を招き，かえって操作性を低下させている例は少なくない．そこで PAPILS では操作コマンドよりも，操作対象であるテキスト自体に重きをおいた仕様としたい．この概念モデル明解化のアプローチの源泉は，オブジェクト指

† Next Generation Editor by Keiji KOJIMA (Central Research Lab., Hitachi, Ltd.).

†† (株)日立製作所中央研究所

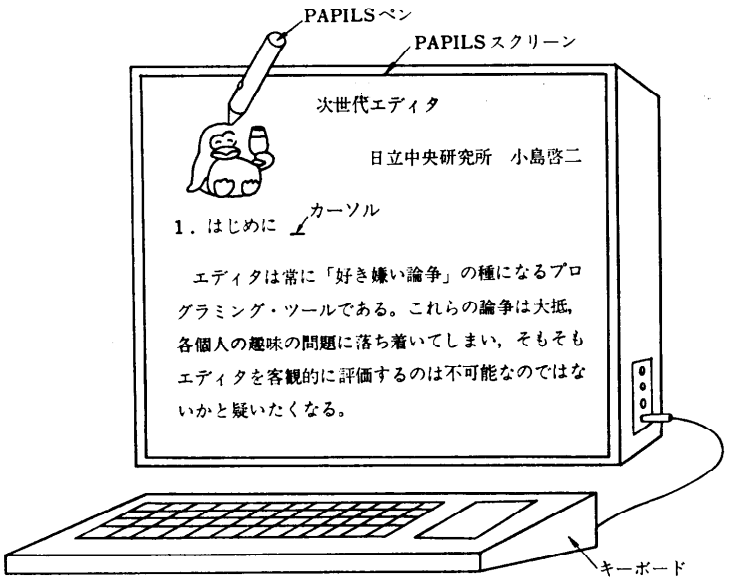


図-1 PAPILS システムの基本構成

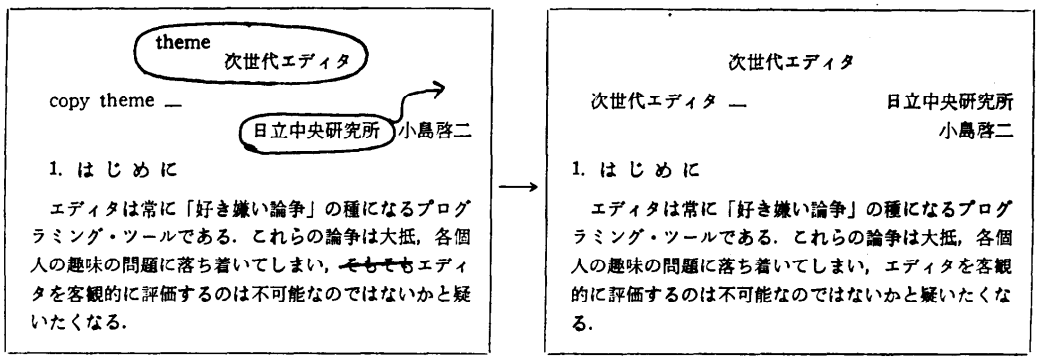
向[1]の考え方にある．

さて PAPILS では画面に表示されているものすべてを編集の対象とする．これらの編集対象をオブジェクトと呼ぶことにしよう．

オブジェクトをさらに制御オブジェクトと非制御オブジェクトの2つに分類する．非制御オブジェクトはいわゆるテキストである．例えば PAPILS を使ってある論文の原稿を作成しているとすると，この原稿を構成する単語や文章あるいは図など，編集の対象となるものはすべて非制御オブジェクトである．一方，制御オブジェクトは，非制御オブジェクトに対する編集コマンドと考えられる．制御オブジェクトにはさらに，キーボードから入力される通常のコマンドとしての記号列 (delete, insert, copy etc.) と，PAPILS ペンによって入力される編集用図形とがある．この後者，すなわち図形的な制御オブジェクトについて若干説明しよう．例えば原稿中のある文章を削除する場合には，ちょうど原稿用紙上での修正と同様に，PAPILS スクリーン上の文章（非制御オブジェクト）の上に削除を示す線（制御オブジェクト）を PAPILS ペンを用いて引いておく．そして制御オブジェクトであるこの削除を示す線の実行を指示するとはじめてこの文章が実際に削除される．図形的な制御オブジェクトにはこの他に，指示円や矢印などが考えられる．指示円はオブジェクトを指定したり命名したりするのに用いられる．すなわちあるオブジェクトを指定するにはそのオブジェクトを囲む閉曲線（指示円）を PAPILS ペンを用いて記入する．さらに指示円内に名前を記入すれば指定したオブジェクトに名前をつけることもできる．また矢印はオブジェクトの移動先を示すのに用いる．これらの制御オブジェクトとその機能の概略を**図-2** に示す．

制御オブジェクト，非制御オブジェクトはともにキーボードや PAPILS ペンを用いて入力される．入力されたオブジェクトが制御オブジェクトかどうかの判定は，例えばコントロール・キーのようなものを用意し，これを押している時に入力された情報は制御オブジェクトとする．PAPILS スクリーン上でも制御オブジェクトと非制御オブジェクトとは異なる色や輝度で表示すれば区別がつく．

編集用図形によるテキスト編集は，図形的な2次元情報を主体とするため，言語的なコマンドによるのに比べて人間のパターン認識力を活かせるという利点を有する．例えば前に例としてあげた削除や移動を指示する制御オブジェクトはたとえ実行しなくても，その意味するところが直観的に把握できる．また，ユーザが編集をコマンド単位に逐次進めていくタイプのエディタでは，数コマンド以上を要する一連の編集作業を行うのは決して容易ではない．というのはこの一連の編集作業の途中段階では，今何をやっているのかは完

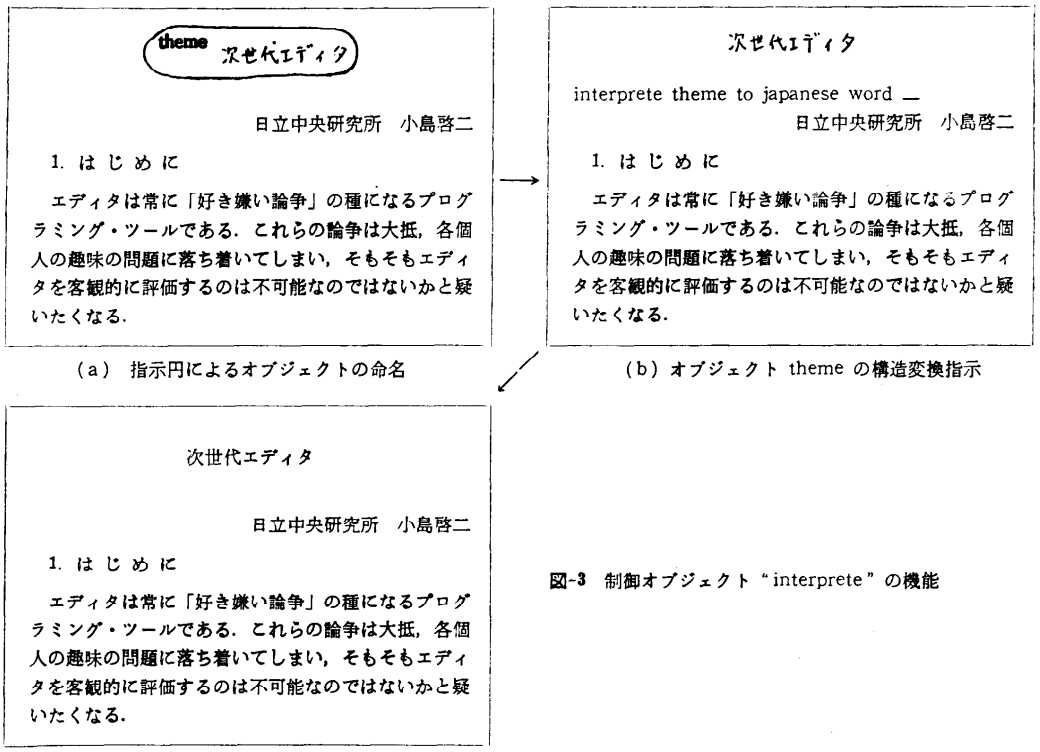


(a) 制御オブジェクト実行前　　(b) 実 行 後

図-2 制御オブジェクトとその実行効果

全にユーザの記憶のみに頼っているためであり，少し気が散ったりするとしばしば混乱を生じる．これらの不透明な中間段階をできる限りなくすことはエディタに限らず対話的システムでは重要である．PAPILS では編集コマンドである制御オブジェクトをテキストと同格に扱っており，また制御オブジェクトはいつ実行しても良いので前述のような不透明な中間段階が生じにくい．すなわち前の例の場合，一連の編集作業に必要な数個の制御オブジェクトをすべて入力してからその妥当性を PAPILS スクリーン上で視認し，しかる後にこれらの制御オブジェクトを一斉に実行させれば良い．

ここまでは主に，PAPILS の画面エディタとしての側面について説明してきた．以下では PAPILS の



(a) 指示円によるオブジェクトの命名　　(b) オブジェクト theme の構造変換指示

(c) 変 換 結 果

図-3 制御オブジェクト "interprete" の機能

構造エディタとしての要求仕様を述べる。

テキストの編集に際し，その構造を利用することが極めて有効であることは論をまたない．PAPILS においてもテキストの有する構造をどう統一的に扱うかが最大の課題といえる．そこで PAPILS では，オブジェクトの構造 (structure) という概念を導入し，これを扱う制御オブジェクトを用意することにより構造エディタとしての機能をもたせる．

オブジェクトの構造というと複雑に聞こえるが，要はそのオブジェクトがどういった性格のものであるかを示すにすぎない．構造の例としては，文字 (character)，単語 (word)，文 (sentense)，図 (figure)，表 (table) 等数多く考えられる．

オブジェクトに，ある構造を与えるための制御オブジェクトとして，“interprete” コマンドを用意する．すなわち，このコマンドは “interprete 〈オブジェクト名〉 to 〈構造名〉” の形で用い，〈オブジェクト名〉によって指定されたオブジェクトに，〈構造名〉で示される構造を与える機能を有する．以下このコマンドの使用例を，**図-3** に従って説明する．

今あるユーザが PAPILS ペンを用いた手書き入力によって，非制御オブジェクトとして「次世代エディタ」と書き込み，このオブジェクトに “theme” という名前をつけたとしよう (**図-3**(a))．この時点では PAPILS システムは，theme が手書き情報であったため，まだ theme が文字列かどうか等の認識は一切行わず theme は単なる図形情報と判断する．したがって theme の構造としては figure 構造が暗黙のうちに仮定される．さてユーザが次に theme を日本語の単語として PAPILS に認識させたい場合，“interprete theme to japanese word” なる制御オブジェクトを入力して実行すれば良い (**図-3**(b))．このコマンドを受けた PAPILS は手書き入力された図形情報であった theme に対しパターン認識を行って対応する仮名と漢字に変換する (**図-3**(c))．

図-3 の例は手書き文字の認識を行うものであったが，interprete をこの他に，言語の簡易翻訳にも使いたい．例えば図-3 の例においてさらに非制御オブジェクトである theme を English 構造に interprete するよう指示すれば，「次世代エディタ」が英訳されて「Next Generation Editor」などとなる．少なくとも主要な言語（プログラミング言語を含めて）の中の諸構造は PAPILS システムに標準的に定義されていることが望ましい．

また文字列だけでなく，図形についても種々の構造を定義しておけば同様に手書き図形を清書したり，座標を変換したりするなど多様な図形編集が可能になる．

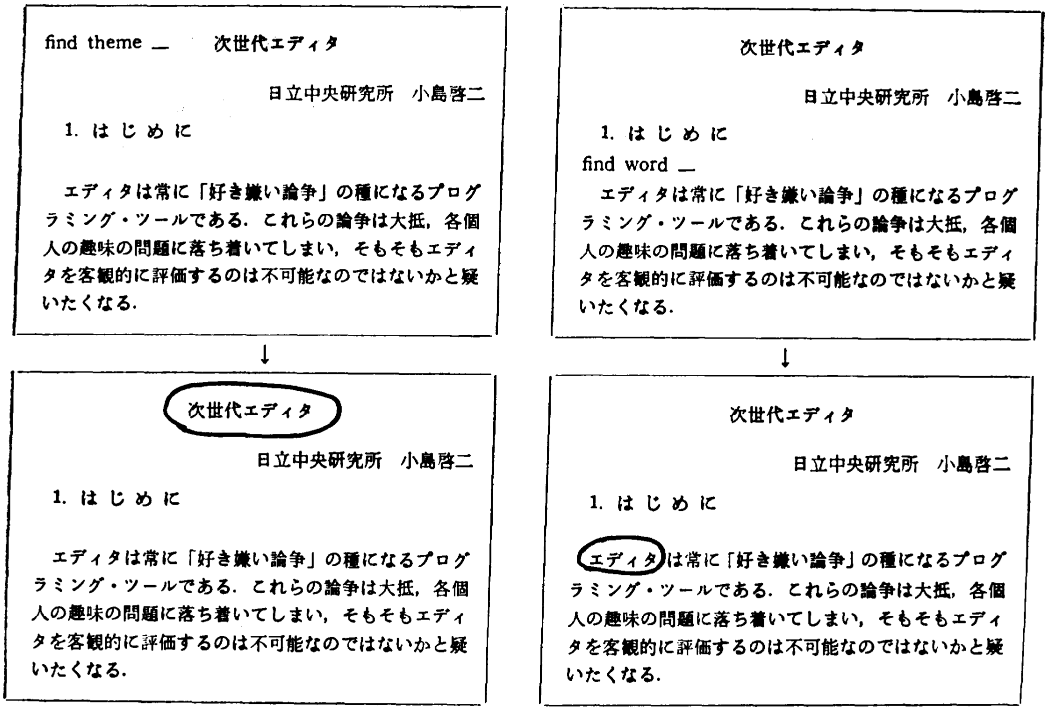
ユーザが新しい構造を定義する機能も必要であろう．例えばある特定の書式のみが許される書類の形式を新たに構造として定義しておけば，ユーザは PAPILS で作った書類がその書式にのっとっているか否かをチェックしたり，別の書式で作られた同目的の書類をこの書式に変換したりできる．

構造の定義方法としては，その構造が満たすべき規則を記述する方法が考えられる．代案として，例を記述する方法等も有力であるが，以下では規則を記述する方法を前提とする．規則自体の形式としては述語論理を若干自然語風にして書き易さと読み易さを増したようなものが妥当であろう．構造に関する規則が格納されるデータベースを考え，これを構造辞書と呼ぶことにする．

あるオブジェクトの構造を interprete コマンドで別の構造にしようとした場合，PAPILS はその interprete が可能か否かを構造辞書を参照して調べる．可能であればむろん直ちに結果を表示する．不可能である時，すなわち，構造辞書の規則からは複数の interprete の結果があり得たり（あいまいな場合），あるいは逆に何かの規則に反するために結果が得られない時には，システムが構造辞書の該当部分を表示すれば有益である．

構造に関する規則のデータベースである構造辞書と共に，オブジェクトに関する辞書も欲しい．これをオブジェクト辞書と呼ぼう．オブジェクト辞書には，現在テキスト中にある命名されたオブジェクトをすべて登録する．図-3 の例ではオブジェクト theme の命名と同時に自動的にその名前，テキスト中での位置，そしてその構造などがオブジェクト辞書に記憶される．ユーザが自分がどの様なオブジェクトを定義したか，あるいはオブジェクトにどんな構造を与えたか，などを忘れてしまった際にはこのオブジェクト辞書の助けを借りることになる．また，編集コマンドの説明等を表示するいわゆる HELP 機能もオブジェクト辞書の制御オブジェクトの項を参照することで得られる．

以上，PAPILS における構造エディタとしての側面について述べた．さて画面エディタ，構造エディタを問わず基本的であって，かつ PAPILS についてはまだ説明していない重要な操作に，テキスト内の自在



(a) オブジェクトの探索　　(b) 構造の探索

図-4 制御オブジェクト“find”の機能

な移動，いわゆる traveling がある．traveling にはカーソルを移動させたり，テキストの別の部分を表示させたりする目的の種々の機能が必要である．ここでは紙面の都合もあり，PAPILS の制御オブジェクトの1つである“find”の機能についてのみ述べる．この制御オブジェクトは“find”〈オブジェクト名〉”もしくは“find〈構造名〉”の形で用いる．

“find〈オブジェクト名〉”を実行すると，そのオブジェクト名をもつオブジェクトが探索され，それを含むテキストの部分が表示されるとともに，そのオブジェクトが指示円(制御オブジェクトの1つ)で囲まれる(図-4(a))．一方，“find〈構造名〉”を実行すると，その構造をもつ次のオブジェクトが探索され，やはりそれを含むテキストが表示されるとともにそのオブジェクトが指示円で囲まれる(図-4(b))．

find をこのような仕様にしておけば，単に，記号列のマッチングのみならず，構造としてのマッチングをとりながらの traveling を行える．

3. まとめ

次世代には筆者としてはこんなエディタが欲しいといった願望を架空の編集システム PAPILS の仕様をスケッチする形で述べた．かなり抽象的でとりとめのない仕様記述となったが，概略のイメージはおわかり頂けたと思う．筆者が要求仕様として特に強調したいのは次の諸点である．

(1) 画面エディタとしての能力向上

筆者がエディタで最も重要と考えるのは操作性である．高い機能を持つコマンドが数多く用意されているよりも，マウスなど秀れた操作インタフェースがある方が使い易い場合が多い．画面に直接情報を入れるなど入力インタフェースの進歩を期待したい．

(2) 構造エディタとしての能力向上

構造エディタとは「構文的に正しいテキストの作成を支援するエディタ」であると解釈している．この観点から筆者は，次世代エディタの構造処理能力を高めるには機械翻訳に代表される自然語処理技術が必須であると思う．

よく画面エディタと構造エディタは対立的にとらえられるが，上記(1)と(2)は決して対立する要素ではなく，高いレベルで両立させうる点を指摘しておきたい．そしてそれが実現された時，エディタは多くの場面で「紙と鉛筆」にとって代わり得ると信ずる．

謝辞 本稿作成にあたり大変お世話になった慶応大学の永田先生をはじめ本特集の編集委員の方々に深く感謝致します．

参 考 文 献

1) byte, Vol. 6, No. 6, McGraw Hill (Aug. 1981).

(昭和59年5月16日受付)