チュートリアル

予測 / 例示インタフェースの研究動向

増井 俊之

今やった 操作がそのまま プログラム — 暦本純一

1 予測 / 例示インタフェースとは

近年はあらゆる場面で計算機を使う機会が多くなっており，計算機操作の繁雑さが大きな問題となりつつある．計算機の機能が多いために操作が複雑になることはある程度仕方がないかもしれないが，現実には単純な目的の処理の操作を行なうためにも大きな手間がかかることが多い．同じ装置を使って同じ結果が得られるのであれば，なるべく少ない操作で処理を指示できることが望ましい．

例えば住所録を作ったり手紙を書いたりといった日常的な作業においても，似たような名前や住所を何度も入力したり，フォーマットを整えるための編集操作を何度も繰り返したり，定型文を何度も入力したり，冗長な作業を行なっていることが多い．無味乾燥な繰返し作業は多くのユーザにとって苦痛である．

このような問題は，計算機自身にユーザの操作を予測させることによりある程度改善されると考えられる．例えば，ユーザが連続する行に 1, 2, 3, 4 と数字を入力した場合，続く行に 5，6... を入力するであろうことが予測できるので，それに続く一連の操作をシステムに代行させることができれば、ユーザの操作量は減ることになる．また，数字をひとつずつ増やしながら行に挿入するというプログラムをこのような操作の例から自動生成することができれば，沢山の行に適用したり，別の機会に再利用したりすることも可能になる．

例示によってプログラムを自動的に作成する手法は古くから PBE (Programming By Example)， PBD (Programming By Demonstration) などと呼ばれて研究されていたため，このようなシステムを総称して PBE/PBD システムと呼ぶことも多いが，ここでは**予測 / 例示インタフェース**と総称することにする．

ユーザの操作をシステムに予測させるには各種の手法が考えられる．ユーザの以前の操作履歴を例データとして記憶しておくことにより次の操作を予測させることも可能であるし，例示によりデータをあらかじめ明示的に与えておくこともできる．例データから規則を学習したり他データに適用するための推論を行なうためには，機械学習や帰納的推論に用いられる各種の手法を使用することができる．例データからの推論をうまく行なうことにより，次の操作の単純な予測だけでなく，ユーザの意図を汲み取ったり，好みや癖を抽出して適応的な動作をさせることさえできるようになる可能性がある．このように，次の操作を予測したり，与えた例データから規則を推論したりする予測 / 例示インタフェース手法は，計算機などの機械を使いやすくするための基本的な技術のひとつであるということができる．

予測により操作の総量を大きく減らすことができれば，単に面倒が減ったり使いやすくなるというだけでなく，機械操作に慣れない人や運動能力が充分でない人にとっても計算機が楽に使えるようになるという点で意義が大きい．いわゆる障害者向けのシステムでは、予測機

Researches on Demonstrational and Predictive Interface Techniques.

Toshiyuki MASUI, ソニーコンピュータサイエンス研究所, Sony Computer Science Laboratory Inc.

コンピュータソフトウェア, Vol.14, No.3(1997), pp.4–19.
1997 年 3 月 13 日受付.

能を備えているものも多い.

Myers は，予測 / 例示インタフェースを，プログラムの作成を支援するかどうか / 知的処理を行なうかどうかの 2 つの基準で 4 種類に分類しているが[35]，実際には広汎な種類の予測 / 例示インタフェースが存在するため明確に 4 種類に分類できるわけではない．以下では便宜的に，アプリケーションの知識やユーザの操作履歴などからユーザの次の操作をシステムが予測する予測インタフェースシステムと，ユーザが明示的にシステムに例を与えることによりプログラムを生成する例示プログラミングの手法をインタフェースに応用した例示インタフェースシステムを分けて解説する.

2 予測インタフェース

コマンド行インタプリタや文書エディタのように，キーボードを使用して文字列のみを操作するシステムにおいては，各種の単純で効果の高い予測手法が従来から広く使用されている．予測を行なうためのデータとしてはユーザの操作履歴，予測実行時のコンテクスト，辞書などが使われることが多い.

2.1 予測インタフェースの実例

UNIX の`csh` や`tcsh` のようなコマンドインタプリタ (シェル) では，以前起動したコマンドに対しその省略形や番号を指定するだけで再実行することのできるコマンドヒストリ機能や，ファイル名の先頭の何文字かを指定するとそれに続く文字列の候補リストをシステムが提示してくれる補完 (completion) 機能[53]がよく使われているし，テキストエディタ GNU Emacs では補完機能に加え，テキスト中に既に現われている文字列を参照して次の入力を予測する dynamic abbreviation 機能も広く使用されている．これらにおいては，ファイル名 / コマンド名 / テキスト中の文字列などが静的または動的辞書として予測に使用されている.

Darragh らの Reactive Keyboard[10]では，UNIX のシェルのようなコマンド言語インタプリタやテキストエディタのキーボード操作において，ユーザの以前のキーストローク列の頻度情報から次の入力文字列を予測し，表にして提示してユーザに選択させたり，カーソルの位置に候補を表示したりすることにより，障害者やタイピングが苦手な人の補助に成功している．提示された予測結果がユーザの要求と一致していた場合はそれを採用し，一致していない場合は予測結果を無視して作業を続けることができる． Reactive Keyboard では，テキスト圧縮手法として性能の良い PPM (Prediction by Partial Match) 法[2] †1に似た頻度集計方式を使用してユーザが次に入力する文字列の予測を行なっている．同様の手法により，ユーザ操作を予測する電卓も試みられている[49].

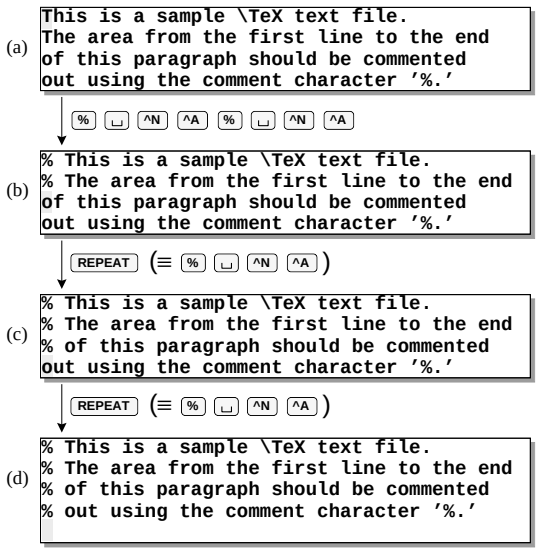

図 1 Dynamic Macro 使用例

筆者らの開発した Dynamic Macro[22][63]は， GNU Emacs 上で同じ操作を何度か繰り返した場合，ユーザからの指示により操作履歴から繰返しパタンを抽出して再実行を指示することができる． Emacs 上でのあらゆる繰返し操作に対して有効であり，マクロ登録が不要であるという特徴を持っている．図 1の例では，行の先頭に “% ” を挿入するという繰返し作業の 1 回分が，ユーザによる繰返し実行指示により自動的にマクロ化 / 実行される様子を示している.

Cypher の Eager[7]は， Macintosh の HyperCard 上で

†1 例えば “`abracadabra`” に続く文字を予測する場合，“`a`” の後には “`b`” が 2 回と “`c`”，“`d`” が 1 回ずつ出現した実績があり，“`ra`” の後には “`c`” が 1 回出現した実績があり，“`bra`” の後にも “`c`” が 1 回出現した実績があるといった出現頻度の荷重和をとることにより次の文字の予想出現確率を計算する.

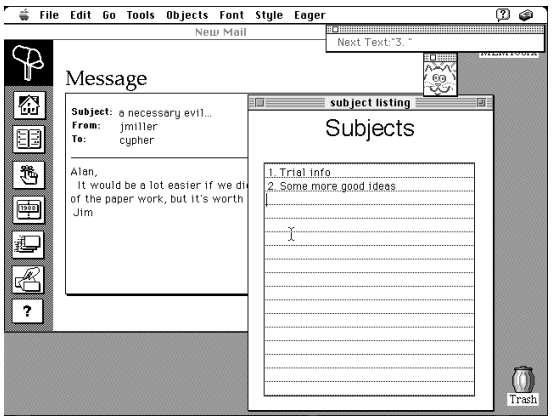

図 2　Eager

似たような GUI 操作をユーザが繰り返したとき，システムがそのパタンを検知してユーザの次の操作を予測するシステムである．ユーザの次の操作が予測できたとき，システムはユーザが次に実行すると思われる操作を先取りしてユーザに示す．例えばユーザが次に特定のウィンドウを選択するであろうと予測した場合，システムはそのウィンドウをハイライトすることにより予測が行なわれていることをユーザに示す．ユーザはシステムの予測の有無にかかわらず操作を続けることができるが，ユーザがそのウィンドウを選択したときは予測が正しかったことをシステムに伝えていることになり，そうでない場合はシステムの予測を否定したことになる．そのような操作を何度か繰り返した後，システムが正しい予測を行なっているということをユーザが確信した場合は，残りの処理の自動実行をシステムに指示することができる．システムはヒューリスティクスを用いてユーザの操作の繰り返しを検出する．例えばユーザが “月曜”, “火曜” と順番に入力した場合はシステムは次は “水曜” であることを予測する．

図 2に Eager の動作例を示す．ユーザは左上のウィンドウから`Subject:`の後の文字列を切り出して右下のウィンドウに貼り付けようとしているところであるが，このような操作を既に 2 回繰り返しているので，右上のような「猫」が出現してユーザに繰返しの存在を認識させ，次のユーザの操作の予測を行なう．

2.2　予測のレベル

シェルのコマンドヒストリや Dynamic Macro の場合は，ユーザからの指示により直前の操作を再実行しているだけなので，予測と呼ぶのはふさわしくないという見方もあるが，ユーザは必ずしも以前の操作列を完全に記憶しているわけではなく，意図通りの処理が行なわれてほしいという期待が含まれているため，ユーザは予測機構を使用しているという意識を持って操作を行なっていると考えられる．

直前の操作とコンテクストから次の実行内容が決まるという意味では GNU Emacs における罫線モード [61]も非常に単純な予測インタフェースと考えることができる．罫線モードでは，現在のコンテクスト (カーソルの位置に既に存在する罫線素片の種類) 及び直前の操作 (罫線描画方向) にもとづいて罫線素片を選ぶことにより，上下左右の矢印キーの操作により移動しながら罫線を引くことができる．動作機構は Dynamic Macro やコマンドヒストリの場合とほとんど違いは無いが，ユーザの期待と異なる動作をすることはほとんど無いため予測インタフェースとして認識されてはいない．また仮名漢字変換も，コンテクスト及び操作履歴から変換される漢字が決定されるので，一種の予測インタフェースとみることも可能である．

3　例示インタフェース

予測インタフェースでは，システムが予測を行なうために使用するデータが何であるかは必ずしもユーザが知る必要はないが，システムに対して明示的に例データを与えてユーザの意図や好みをシステムに推論させることにより操作を簡略化したりマクロを作成したりすることも可能であり，このようなインタフェースを例示インタフェースと呼ぶ．

例示インタフェースは，ユーザの個々の操作を予測することにより操作の総量を減らすことに使用できることに加え，自動処理を行なう手続きを作成したり，対話型プログラムそのものを作成したりするために積極的に利用することもできる．このような処理は従来はテキストベースのプログラミング言語を使用して記述するのが普通であったが，近年の計算機環境においてはエンドユーザプログラミングの支援や GUI プログラミングへの応

用という点で例示インタフェースの重要性が増している．

3.1 エンドユーザプログラミングへの応用

計算機が広い分野で多くの人々に使われるようになったため，ニーズに対してユーザが自分でプログラムのカスタマイズを行なったりプログラムを作成したりすることが望まれるようになってきている．エンドユーザがプログラムを作成するためには，依然としてテキストベースのプログラミングが主流であり， C, Java, Perl, Tcl などの汎用プログラミング言語やアプリケーションのマクロ言語の修得本が書店の柵を埋め尽くしている†2．しかし誰もがプログラミングに習熟しなければならないほど高度な処理が常に必要になるわけではない．現状でば例えばメールを条件に応じて分類するといった単純な処理を自動的に行なわせるだけでもちょっとしたプログラミングが必要であることが多いが，実際のメールを分類した作業例にもとづいて分類プログラムを自動生成することができればユーザはプログラムテキストを扱う必要が無くなる．多くの人間にとっては，対象の扱いを抽象的に指示するプログラミングを行なうよりは，具体的な作業を例示する方が理解が容易である．

3.2 GUI プログラミングへの応用

GUI における画面上の対象の表示やふるまいと，そのプログラムテキスト上での表現の間のギャップは非常に大きいため，抽象的対象の扱いに慣れたプログラマにとっても， GUI の開発においては例示インタフェースが有用である場合が多いと考えられる [67]．例えばボタン上でのマウスクリックのような基本操作を表現する場合でも，普通のプログラミング言語では状態遷移や座標計算が必要になってしまうが，そのかわりに実際のボタン表示やマウス操作によって操作を表現できればプログラミングがはるかに容易になると考えられる．このような考えにもとづき，特に GUI システム開発を目的とした各種の例示インタフェースシステムが開発されてきている．

GUI システムにおける表現とプログラムのギャップを埋めるための開発システムとしてはいわゆるインタフェースビルダがある程度の成功をおさめている [65]．インタフェースビルダとは，直接操作インタフェースによるグラフィック画面設計とテキスト編集によるプログラミングを併用しながら，ツールキットを用いてアプリケーションを作成するシステムである．ボタンやスライダのようなインタフェース部品それぞれに対してその画面上での配置を指定しつつ，部品が操作されたときの処理も簡単に指定できるようになっている．インタフェースビルダを用いると，インタフェース部品を画面上に直接配置できるため，高速にプロトタイプを作成することができるが，インタフェースビルダは大きな統合的環境になりがちで小回りがきかないうえに，使用言語やツールキットなどが制約をうけるし，あらかじめ用意されている仕様と異なるインタフェースを作ることが困難であることが多い．また， WIMP インタフェース†3に特化しているため音声やジェスチャのように現在一般的でないインタフェースには容易に対応できないし，状態遷移のような動的な処理を指定できないという欠点も持っている．

インタフェースビルダは GUI の画面配置を実例で指定することにより成功したシステムと考えることができるが，例示インタフェース手法をより積極的に活用することにより，システムの動的なふるまいを例示により示したり，音声などのインタフェースを併用することも可能になる．このように，特に将来のインタフェースシステムの開発においては，システム開発者にとっても例示インタフェースは非常に魅力的と考えられる．

3.3 例示インタフェースの実例

非常に多くの例示インタフェースシステムが提案されているが，ここではその一部を応用別に簡単に紹介する．

3.3.1 テキスト編集

Nix の Editing by Example システム [39]は，テキスト編集作業において，編集前のテキストと編集後のテキス

†2 音楽情報処理の分野では Max というデータフロー型のビジュアルプログラミング言語が音学家にも普及しており，盛んにエンドユーザプログラミングが行なわれているが，こういうケースは稀である．

†3 ウィンドウ，アイコン，メニュー，ポインティングデバイスを多用する，現在最も主流の GUI スタイル．

トの組の例をユーザが示すことによりシステムにその変換規則を推論させ，残りのテキストに対しその変換を適用させることができるようにするものである．多くの例示インタフェースシステムはユーザの実際の操作列を例データとして扱うのに対し， Editing by Example システムは操作前 / 操作後の変化のみを扱うという特徴がある．少ない例からの推論を行ないやすいような文字列置換操作 (正規表現による文字列置換のサブセット) を編集操作として利用することができる．

Mo らの TELS システム [29][50]は， GUI による単純なテキストエディタ上でのユーザの繰り返し操作から汎化によりプログラムを生成して以降の操作に適用可能とするシステムである．簡単のため編集操作は挿入 / 削除 / 移動 / 選択の 4 種類しか許されていない．似たような操作が繰り返された場合，各操作の前後でのコンテキストを汎化することにより積極的にループを検出してプログラムを生成する．例えば電話番号を順次選択していくような場合， “空白文字の後の数字`222-3456` の選択” と “空白文字の後の数字`234-5555` の選択” という操作が続けて実行された場合は “空白文字の後の数字`2**-**5*` の選択の繰り返し” というプログラムを生成する．このように，システムは同じ機能をもつプログラムのうちなるべく最小のものを見つけようと試みる．生成されたプログラムを実行するときに間違いが発見された場合は，ユーザが操作を訂正することによりインクリメンタルにプログラムを修正することができる．ヒューリスティクスにより一番もっともらしい操作が選択されるようになっていることに加え，修正により正しい条件判断ができるようになっているため， Nix の Editing By Example システムでは扱うことができない複雑な処理も例示のみで指定可能となっている．

3.3.2 GUI 操作の自動化

Halbert の SmallStar [14]は， Xerox 社の Star ワークステーションのデスクトップの GUI 操作を例示によりプログラミング可能にするシステムである． GUI 操作のプログラミングを簡単にするため，ユーザはキーボードマクロを定義する場合と同様に，実際の操作を行なった後でその操作列を編集することによりプログラムを作成する．ユーザはプログラムしたい操作列の実行開始をシステムに指示してから実際の操作を行ない，最後に操作列の実行終了をシステムに指示することにより操作列を示すテキスト形式のプログラムを得て後でそれを修正することができる．例えば，「全ての “`*.bak`” という名前のファイルを “`bak`” というディレクトリに移動する」というプログラムを作成したい場合は，まず上記の手法により適当なファイルを “`bak`” ディレクトリに移動することにより「ひとつのファイルを “`bak`” というディレクトリに移動する」というプログラムを作成し，そのプログラムをエディタで編集して条件文を付加することにより必要なプログラムを作成する．

Bos の Edward [3]は GUI の操作手順から次の操作を推論するシステムである．手順の推論方式自体は Eager と類似しているが，自然言語でユーザが操作を指示することができ，またシステムも推論結果を自然言語で説明したりユーザに質問を行なったりする．推論された手順はマクロとして名前をつけてセーブしたり編集を行なうこともできる．

Modugno の Pursuit [30][31]は，ユーザの例示操作からの推論 / 汎化により理解の容易な視覚言語プログラムを生成し，それをユーザが直接編集することによりシステムの間違った推論を修正できるようにしたシステムである．生成されたプログラムの例を図 3に示す．視覚言語の表現を画面上での実際の操作で使われるアイコンに似せることにより，プログラムをより理解しやすいように工夫している．操作前の状態と操作後の状態を組にして示すことにより操作内容を直感的に表現している．図 3のプログラムには条件分岐が含まれているが，条件分岐や繰返しもユーザの操作からある程度自動的に抽出できるようになっている．

Potter の Triggers [41]は，任意のアプリケーションプログラムが表示するビットマップ画面上のパタンマッチングにより操作の自動実行を行なわせるシステムである．画面上のビットマップパタン及びそれに関連する操作を例示により定義することにより，それと同じビットマップパタンが表示されたときに，定義された操作を自動実行させることができる．例えば，表計算ソフトにおいて全ての負の数値データに印をつけたい場合は，マイナス記号 “`-`” を示すビットマップパタンを指定してそのパタンの近辺に印をつける操作を例示により定義することにより，画面上の残り全ての “`-`” の近辺に同じ

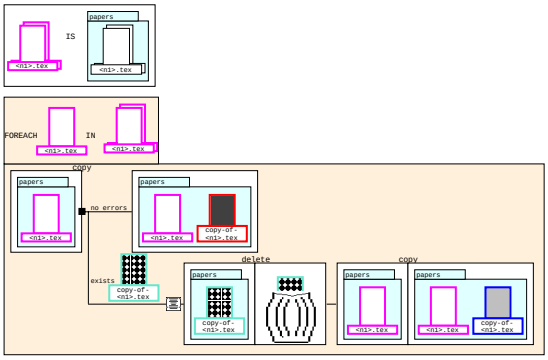

図 3 Pursuit (文献[30]より引用)

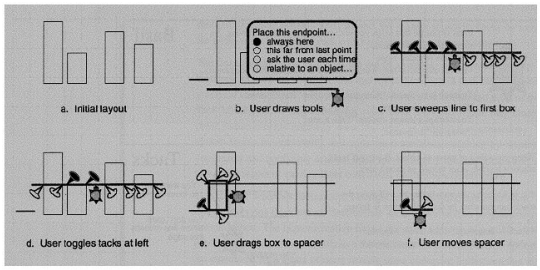

図 4 Metamouse の操作例 (1) (文献[24]より引用)

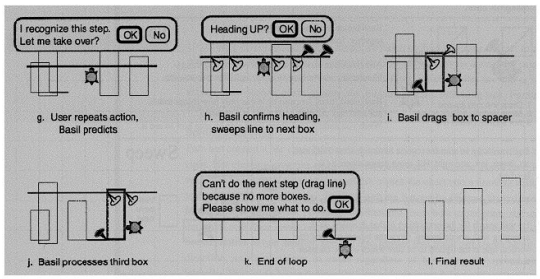

図 5 Metamouse の操作例 (2) (文献[24]より引用)

印をつけることができる．ほとんどの予測 / 例示インタフェースシステムでは，予測や例示を行なうためにアプリケーション内のデータを利用するのが普通であるが，Triggers はビットマップ画面のみを用いて操作の例示を行なう．アプリケーション内のデータには必ずしもアクセス可能とは限らないが， Triggers は画面に表示されるビットマップデータのみを使用するため，画面へ表示を行なう任意のアプリケーションに対して適用することができる．

杉浦らの DemoOffice[56]は，アプリケーション上の GUI 操作から再利用可能な部分を自動的に抽出してマクロとして登録するシステムである．メールボックスのようなデータ集合の要素 (1 通のメール) に対して選択やコピーなどの GUI 操作が行なわれた場合，システムは他の要素に対しても同様の操作が行なわれる可能性があると判断し，自動的にマクロ生成を開始する．またデータの依存関係を解析することにより，操作された要素に関連した操作のみを履歴から抽出 / 汎化してマクロを生成する．異なる操作が行なわれたときにはまた別のマクロを自動生成する．実際に他の要素に対して同じ操作が必要になった場合は，用意されたマクロに対して新しい要素を適用可能かどうかをユーザが実行前に試してみることができるようになっている．

3.3.3 グラフィクス編集

Maulsby の Metamouse[23][24]はグラフィカルな図形の編集操作から編集プログラムを自動生成するシステムである．編集操作は，「上に移動させた直線が矩形に接したとき直線の長さだけ矩形を移動させる」といったようなプロダクションルールの集合として表現される．ループを構成する規則を使うことにより繰り返しを含むプログラムも作成できる．規則を発火させる条件としては，図 4の例のように「接する」「交わる」といった幾何的条件が重視される．ユーザが以前の操作と完全に同じ操作を始めたときは，システムはすぐにユーザの次の行動を予測して提示するため，同じ操作を 2 度以上含む大きな繰り返しは検出できない．規則はユーザの与える少ない例からの汎化により生成されるが，システムは推論の様子がユーザによくわかるように擬似マウスや制約条件を表示しながら徐々に推論を行なう．ユーザはシステムに自分の意図を教えるという意思を念頭に置きながら，間違った推論を修正しながら正しいプログラムを作成していく．

Kurlander の Chimera[17]は，図形エディタなどにおける編集操作の履歴をプログラムとして後で編集することにより， GUI 操作をマクロとして定義可能にするシステムである． SmallStar では GUI 操作列をプログラムテキストとして表現して後の編集を可能としていたが， Chimera では GUI 操作をグラフィカルな形で表

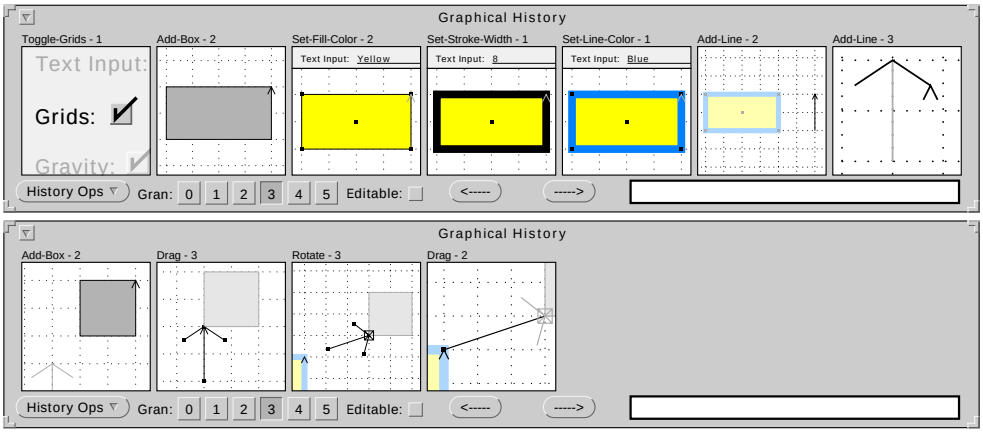

図 7　Chimera における図 6の操作履歴のグラフィカル表現 (文献[17]より引用)

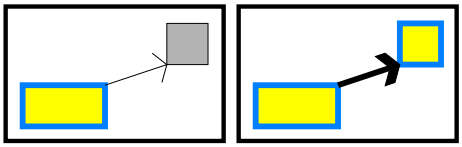

図 6　Chimera の編集操作例 (文献[17]より引用)

現したまま編集ができるような工夫を行なっている．GUI 操作履歴はユーザにわかりやすいように図 6の編集操作履歴を図 7のように紙芝居風に表現し，細かい操作列をひとつのコマで表現したり，局所的な操作において重要な部分だけがコマ上に表現されるような工夫を行なっている．

Lieberman の Mondrian[20]も Chimera と同じように，図形の編集操作履歴をマクロとして登録することのできるシステムである．登録される各編集操作は図 8のように編集前の状態と編集後の状態を組にしたドミノ状のアイコンで表現され，ビジュアルプログラミングの要素として再利用することができる． Chimera の場合と同様に，マクロ化にあたってシステムは各種の推論や汎化を行なう必要があるが，推論の様子を音声や自然言語テキストでユーザに知らせる工夫がされているため，間違った推論が行なわれた場合はユーザはすぐにそれに気付いて修正を行なうことができる．

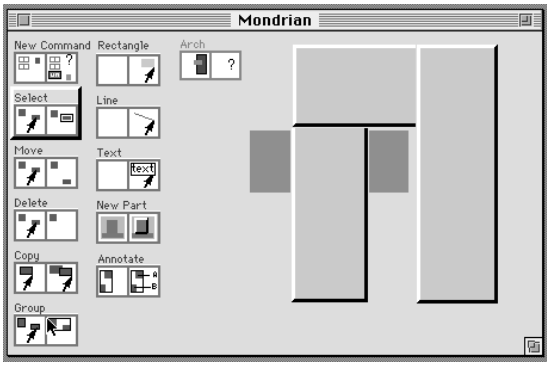

図 8　Mondrian (文献[20]より引用)

3.3.4　GUI プログラム作成

Buxton らの開発した MENULAY UIMS[5]は，画面表示を編集することによりそのインタフェースを示すプログラムが出力され，設計者がそのプログラムと実際の動作を行なう関数をリンクすることによって目的のシステムを得ることができるという，例示による GUI 生成の先駆的システムである．

Myers は例示によりインタフェースプログラムを作成する数多くの試みを提案している． Peridot[32][33]では，ユーザがシステムに示した GUI の操作例から推論を行なうことによりメニューやスクロールバーなどの動きをカスタマイズすることができる． Marquise[37]は，描画エディタなどにおける GUI プログラミングのほとんどの要素を例示のみで実現しようとした野心

的なシステムである．インタフェースビルダは一般に静的な画面設計を行なうモード及びそれをテストするモードを持っているが，Marquise ではこれに加え，マウスドラッグ時等のシステムの動的な反応を定義するためのモードをふたつ持っており，ユーザ操作に対してシステムがどのように反応するかも例示により指定することができる．Myers の率いる例示インタフェースグループでは，スライダやボタンなどを手書きすればすぐに試用できるようにした SILK システム[18]や，ゲームプログラムに特化した Gamut システム[25]など，数多くの例示による GUI 作成システムが開発されている．

その他，例示により配置やダイアログを指定する Druid UIMS[45]，X ウィンドウシステムユーザに広く使われている Tcl 言語 /Tk ツールキット及び X 上のインタフェースビルダ Xf を使って，操作履歴から GUI プログラムを自動生成させる試み[46]など，数多くの研究が行なわれている．

3.3.5 ビジュアルプログラミング

MIT における Logo 教育の経験における知見として，子供やプログラマ以外の人々にプログラミングを教えるには抽象的な概念を使わず具体的な操作を示すことが有効であるということが明らかになり，具体的な操作履歴のマクロ化によるプログラミングを可能とした Instant Turtle システムが成果をあげた．Lieberman の Tinker[19]では，これを拡張して引数の変数を指定したり条件分岐を示したり再帰呼び出しを定義したりできるようになっている．

Furnas は，ビットマップ画面上での単純な書き換え規則の集合だけを用いることにより細線化や計算などの各種の興味深い効果が得られることを BITPICT システムにおいて示した[12]が，山本は Visulan システム[70]において，ビットマップ書き換え規則の集合が汎用のビジュアルプログラムとして使用できることを示した．Visulan は BITPICT と異なり，プログラムを示す書き換え規則は，書き換え対象となるビットマップ画面の中に存在している．

グラフィカルな書き換え規則は子供でも充分理解できるという考えにもとづき，Smith らは子供用の例示プログラミング環境 KIDSIM[9][47]を提案した．KIDSIM は碁盤の目状の盤面とドミノ状の書き換え規則の集合から構成されており，盤面上のパタンにマッチする規則があればそれに従って盤面がどんどん変化していく．KIDSIM は現在 Cocoa[1]という名前で商品化され，Apple Computer より入手可能である†4．

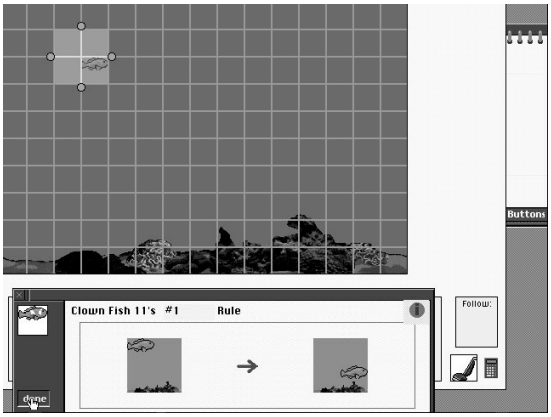

図 9 KidSim (文献 [9]より引用)

Wolber は stimulus(ユーザの操作)-response(システムの反応) モデルにもとづく例示インタフェースシステム Demo[52]，DemoII[11]を開発してきた．これらは静的な GUI の作成に有効であったが，その後開発された Pavlov[51]では，時間経過やユーザの操作を刺激とするアニメーションやシミュレーションも例示により作成することができる．描画モード / 刺激定義モード / 反応定義モード / テストモードを使いわけることにより，ユーザ操作や時間などの刺激信号に呼応するオブジェクトの生成 / 変型 / 削除 / 前進 / などの反応を定義していく．Pavlov では時間経過も刺激として使用することができるため，市販のオーサリングシステムのような，時間に基づくアニメーション作成システムと似た使い方をすることができる．ひとつの刺激 / 反応を定義するためには一度だけ例示を行なう．システムが誤った推論を行なった場合はユーザがテキスト編集により修正する．反応は，刺激の種類 / パラメータ / 反応の生ずる条件により定義される．例えば「ハンドルを右に回すと車が右を向く」という刺激 / 反応の組を定義する場合，システムがヒューリスティクスにより選択した各種の条件から選択を行なうことにより，「車が障害物に重なっていないと

†4 http://cocoa.apple.com/cocoa/

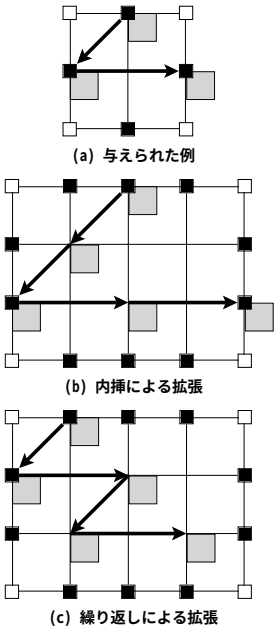

図 10　Layout by Example における推論 ([15]より引用)

き」などといった条件を追加することができる．

3.3.6　グラフィックレイアウト

Hudson の Layout By Example システム[15]は，図形を配置するためのレイアウト規則を例示により推論するシステムである．図形が 2 個の場合の配置例，3 個の場合の配置例... をユーザがシステムに与えることによりシステムは配置アルゴリズムを推論する．アルゴリズムは図 10に示したような「内挿」と「繰り返し」をもとに構築する．

例の数が少ない場合は，その例を満足する配置アルゴリズムはいくつも存在するが，このような場合システムは別のデータに対しそのアルゴリズムを適用してその結果をユーザに示すことにより，ユーザはそのアルゴリズムが適当なものであるかどうかを判断する．このように，実際は推論された複数のプログラムの中からユーザの要求に一致するものを選択しているにもかかわらず，ユーザとシステムの間では例データのみがやりとりされるためプログラムを明示的に扱う必要が無いという利点がある．

高橋，宮下らの TRIP2[48]，TRIP3[28]は制約指向の宣言的図形配置システムである．最初に開発された TRIP は，図形間の制約関係を Prolog で宣言的に記述することにより図形の自動配置を行なうシステムであったが，TRIP2 では図形の実際の配置の変更により制約の記述を修正するような双方向変換が可能となり，TRIP3 ではさらに複数の配置例から宣言的配置制約を推論することができるようになっている．また TRIP3 の後継である IMAGE[27][68]では，推論された配置制約を別の例に適用した結果をユーザに提示することによりその推論が正しいかどうかを判断できるようになっており，Layout by Example システムと同様の効果が得られる．

Myers の例示による文書整形システム[34]では，フォントの選択やセンタリングなどという文書整形の例を示すことにより，システムが汎化によりマクロを生成する．例えば章タイトルの型式を例として与えると，システムはそれを生成するような整形スタイル (位置 / フォント / 番号付け方法など) をヒューリスティクスにより生成する．

グラフなどの配置を行なう場合，遺伝的アルゴリズムや焼きなまし法 (Simulated Annealing) のような確率的アルゴリズムを使用すれば配置の評価値を最大にする配置を自動生成させることができる．しかし評価基準が自明でない場合には評価値の計算方法がわからないことが問題となる．筆者の開発した遺伝的プログラミングによる有向グラフ配置システム[21][62]では，ユーザが良い配置例と悪い配置例の組をシステムに与えることにより，システムが遺伝的プログラミング[16]により配置の良否を決定する評価関数を抽出することができるため，ユーザが評価関数を明示的にプログラミングしなくても，与えた例による評価基準にもとづいて良い配置を生成することが可能になっている．良い配置と悪い配置の例として図 11のような配置の組をシステムに与えることにより，システムは評価関数を自動生成する．生成された評価関数と遺伝的アルゴリズムを用いて異なるグラフの配置を行なうと図 12のような配置結果が得られる．

3.3.7　表データの視覚化

市販の表計算ソフトウェアでは，表データを棒グラフをはじめとする各種のグラフに変換する機能を備えているものが多いが，このような機能はシステム組み込みのものであるためグラフの種類は限られており，システム

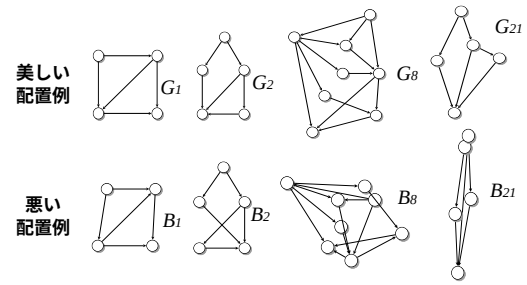

図 11 システムに与える配置例

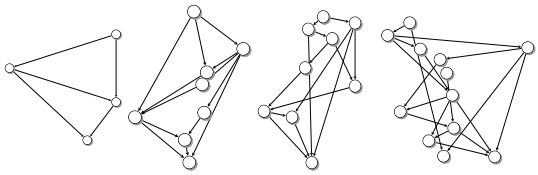

図 12 計算された評価関数にもとづいた配置結果

に用意されていない型式のグラフを書かせることはできない． Myers らの Gold[36]は，表の中のデータの値とグラフ描画に使用される図形の属性との対応を例示により指定することにより特殊なグラフ化を可能にするシステムである．例えば棒グラフを作成する場合は，最初に軸と矩形をひとつ描き，矩形の位置 / 大きさ / 色などの属性が表中のひとつのデータとどのように対応しているかを指定し，残りのデータにも同じ対応関係を適用する．システムは「棒グラフに使われる矩形の片側は軸に接する」といった知識を持っているためユーザはあらゆる属性の視覚化手法を指定する必要はない．

Roth らの Sage[42]は Gold と同じように例示により表データの図示を行なうシステムである． Sage は視覚化のためのツール SageBrush と視覚化のための知識ベース SageBook から構成される．ユーザは SageBrush を使用して描いた図形の属性と表データを関連付けることにより視覚化規則を定義し，その知識を SageBook に格納する． SageBook を使用することにより以前の視覚化規則の例を検索し再利用することができる．図 13はナポレオンのロシア行軍記録データ中の軍の位置，兵隊の数，気温など数多くの属性を座標，色，線の太さなどの 様々な視覚属性にマッピングすることによりひとつのグラフとして表現したものである． SageBrush では例示によってデータと属性を容易に関連付けることによりこのような複雑なグラフも生成することができる．

4 予測 / 例示インタフェースの要件

予測インタフェースを使う目的は計算機操作を簡単にすることであるが，予測システムを使わなくても手間さえ気にしなければ作業は遂行できるので，予測を使わない場合に比べて少しでも不都合がある場合には予測インタフェースは使われない傾向がある．例えば，予測や推論に時間がかかり操作性が悪くなるようなことがあれば使用されにくい．また，どのような予測システムを用いた場合でも予測を実行するための余分な手間が必ず発生するので，予測による効果がそれに見合うだけ大きくなければ普及はおぼつかないし，予測を行なう場合に漠然とした不安感がつきまとったり，誤った予測を実行したときの被害が大きいようなものも問題がある．また， Eager や KeyWatch[26][†5]， OpenSesame[6][†6]などの製品のように，バックグラウンドに隠れてユーザの操作を監視するシステムの場合，予測システムを実装したことすら忘れてしまったようなときに突然予測結果の実行を提案してユーザを驚かせてしまうことがある．また， Reactive Keyboard[10]のように頻度情報から次の出現文字を予測するような場合，同じことを繰り返していた場合でもある瞬間から突然予測結果が変わってしまうことがあるが，こういう突然の挙動の変化もユーザを驚かせてしまうことになる．予測による不都合が発生せずかつ予測の効果が大きい手法は多くはないのが現状である．

一方，プログラミングに例示を用いる手法の場合は，例示によりプログラミングできること自体が大きなメリットとなるため他の面での多少のデメリットは問題になりにくいはずであるが，この場合でもテキストを用いたプログラミングに比べて全体的な手間がかなり改善されることが明らかでない場合は導入されづらい．例えば

†5 IBM-PC 上で，同じキーストローク列の繰り返しを発見すると音などでユーザに知らせ，マクロとして再実行可能にするシステム．

†6 Macintosh 上でのユーザの操作パタンを常に監視しつつニューラルネットで傾向を推論し，マクロ化をユーザに提案するシステム．

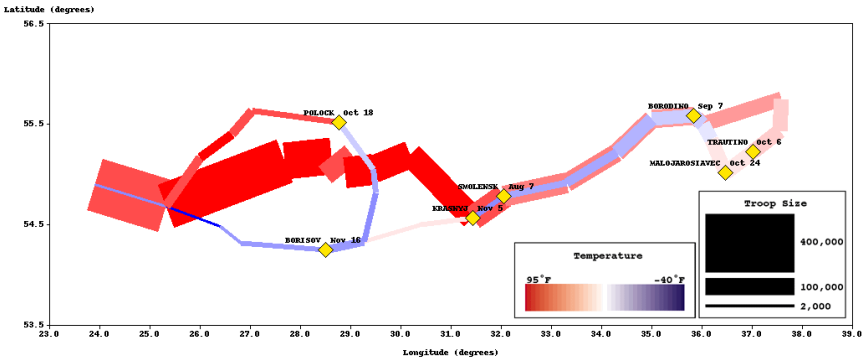

図 13　Sage による表データの視覚化例 (文献 [42]より引用)

例示インタフェースを用いて GUI プログラムを生成する場合，ユーザの操作からシステムがうまく意図を推論することができずに多大な修正作業が必要になるような場合は，例示システムを使わず普通のプログラミングを行なった方が得策であろう．実際，3.3節で紹介したシステムがまだ広く実用レベルにはなっていないのは，例示インタフェースについての世間での認知度が低いという理由だけではなく，例示による効果がそれほど大きくないことが主な原因だと考えられる．

4.1　Just-in-time Programming

一般に，新しいシステムを導入するときには必ず，導入するかどうかの判断を行なうタイミングが重要になる．予測 / 例示インタフェースの場合も，どの時点で予測を実行させたり例からプログラムを自動生成させたりするべきかの決定がむずかしい場合がある．複雑な編集操作を 100 回行なうことが最初からわかっている場合には，なんらかの手段でその操作をマクロ定義したり繰返しを自動実行させたりするほうが効率が良いだろうと判断することができるし，逆に単純な操作を 5 回だけ実行するような場合は，自動実行の工夫をするよりも実際に人手で 5 回操作を繰り返した方が簡単であろう．しかし，例えば複雑な操作を 5 回だけ行なうような場合，自動実行のための工夫をする方が良いか，何も考えずに操作してしまう方が良いかの判断はむずかしい．また，実際に何回操作を行なわなければならないかがあらかじめ判断できないような場合には，移行のタイミングの判断がさらにむずかしくなる．すぐ終わると思っていた仕事が意外と量が多く，最初から自動化する工夫をしておけばよかったと後悔することは多い．

例えば，簡単なプログラムを作ろうとしたとき，一回で問題無くコンパイル / 実行がうまくいった場合はコンパイラを手動で起動するのが一番簡単であるが，バグのために何度もコンパイルが必要になったり，別のライブラリが必要であることが途中で判明したような場合は，Makefile を作って自動コンパイル / リンクを行なわせた方が手間が少ない．作業が何回繰り返されるかは最初からわかっているわけではないので，どの時点でプログラミングを行なうかの判断がむずかしい．

これらの場合のようにプログラムが必要であると判断されたその時点で効果的にプログラムを作成する手法を “Just-in-time Programming” と呼ぶ [40]．この例の場合は，コンパイル / リンク操作の履歴からファイルと操作の間の依存関係を解析することにより自動的に Makefile を作成することが可能である [57][69]が，どのような状況でもこれが可能であることが望ましい．

5　予測 / 例示インタフェースシステムでよく使われる手法

2.1節及び 3.3節で紹介した各種の予測 / 例示インタフェースシステムでは，以下のような手法が共通に用いられていることが多い．

- 操作履歴，コンテクスト，辞書の利用
- アプリケーション依存のアドホックな知識の利用

要件	対応
予測のオーバーヘッドを最小にしたい	最も手軽に入手できる履歴データや既存の辞書を活用
予測により反応が遅くなると困る	単純な予測手法のみを使用し，誤りがあればユーザが訂正
小数の例しか利用できない	ユーザ介入により推論誤りを早目に訂正
実用性を重視	アプリケーション個有の特徴を利用
システムの挙動がわからないと不安になる	ビジュアルな表現によるユーザへの情報提示

図 14 予測 / 例示インタフェースの要件と手法の対応

- ユーザの介入による推論誤りの修正
- 例示データの生成の自動化
- ユーザ操作やプログラムのビジュアルな表現

予測 / 例示インタフェースシステムに共通に見られるこれらの特徴は 4節で述べた要件に起因しており，図 14のような対応関係があると考えられる．

6 予測 / 例示インタフェースに対する批判

Brooks は， 1993 年ごろまでの予測 / 例示インタフェースシステムについてまとめた Cypher の本[8]の書評[4]の中で，「本書で紹介されている予測 / 例示インタフェースシステムにより自動化できる仕事はプログラムを数行書けば実現できるようなものばかりであり，それにより飛躍的に使い勝手が良くなるようなものではない」と，予測 / 例示インタフェースの有効性に疑問を投じている．また， Nardi はエンドユーザプログラミングに関する著書[38]の中で，予測 / 例示インタフェースシステムには終了条件や条件分岐を示すことができない / 操作が間違いを多く含んでいる場合に困る / 推論エラーを修正できない / 推論に高いコストがかる / といった欠点があると指摘し，ユーザが簡単にプログラムを書けるような環境を作った方が有効だろうと述べている．

プログラムを誰でも簡単に作れるような作業であれば，予測 / 例示インタフェース手法を使う必要が無いのは確かであるし，少数の例からの推論に困難がともなうのも事実である．これらの批判がなされた時点において，説得力のある予測 / 例示インタフェースシステムはあまり存在していなかったし，現在でも広く一般に使われている予測 / 例示インタフェースの数は多くないことがこのような批判が出る一因であろう．しかし最近では操作履歴などを活用して効果的に Just-in-time Programming を行なう手法など，説得力のある予測 / 例示インタフェース手法が各種提案されて実用になっているし，また 3.2節で述べたように，予測 / 例示インタフェース手法はシステムのふるまいやユーザの操作とプログラムの間にギャップが有る場合でも適用できるという大きな特徴を持っており，認識を用いたインタフェースや実世界におけるインタフェースのように，タスクをプログラムに簡単にマッピングできないような場合でも適用可能であるため，将来のシステムにおいてはさらに重要となってくると考えられる．

7 予測 / 例示インタフェースの将来

これまでの予測 / 例示インタフェースの研究は，研究が行なわれた時点において広く普及していたインタフェースを増強するためのものが主流であり，テキストベースの文書エディタや GUI などが主な対象であった．前節で述べたように，予測 / 例示インタフェースはこのような従来型のシステムよりも，将来の広い範囲のインタフェースにおいてより有効と考えられる．以下にそのいくつかを示す．

7.1 実世界指向インタフェースにおける応用

計算機の小型化 / 低価格化及び無線ネットワークの整備により常に身のまわりに計算機が存在して生活のあらゆる局面で計算機を利用する世界が目前となっており，このような状況で計算機を活用するための実世界指向インタフェース[54][71]が注目を集めている．実世界指向インタフェースでは，実世界における状況を計算機が認識して常に適切な対応をすることが重要である． 3.2節において， GUI のプログラミングにおける画面上の表示とプログラム内の表現とのギャップを解消するために例示インタフェースが有効であることを述べたが，実世界指向インタフェースの場合は計算機の外側の状況までを考慮しなければならないため，そのギャップはさらに大きくなっており，予測 / 例示インタフェースがさらに有効となる可能性がある．

例えば，各種のセンサを備えた携帯端末の利用者が「夜の品川駅で電源を入れたときは時刻表を表示する」という機能を登録したい場合，通常のプログラミング

言語を使うとすると，センサから得られた緯度 / 経度情報，電界強度，時刻などの数値を判断し，条件を満たした場合に時刻表を起動するようなプログラムを書く必要があるかもしれない．しかし端末が事例にもとづいて適切な動作をするようになっていれば，一度夜の品川駅で時刻表を使ったことがあるという例情報にもとづいて，それに近い別の状況においても時刻表を表示することが可能になるであろう．

7.2 検索インタフェースとの融合

従来の予測 / 例示インタフェースシステムでは， 2.1節で紹介した補完機能を除けば，例データをユーザ自身が作成するものがほとんどであったが，既存のデータを例として積極的に利用することも考えられる．

検索を基本とするインタフェースは見方を変えると予測 / 例示インタフェースと考えることができる．たとえば仮名漢字変換システム SKK[55]では，システム標準の辞書に含まれなかった単語をユーザが辞書に登録することができるが，ユーザが登録した辞書は標準辞書と全く同じ構造をしているので， SKK の使用開始時の状況は，あらかじめ誰かが例示により標準辞書を作成した状態と同じことになる．つまり， SKK は完全に例示インタフェースにもとづく仮名漢字変換システムであるとも考えることができるわけである．

また，筆者が開発したペン計算機向け文章入力システム POBox[66]は，単語の読みの部分指定及び入力位置直前の文字列からの予測により絞り込んだ候補単語の集合の中から単語を選択するという操作を繰り返すことにより高速に文章を入力可能なシステムであるが，あらかじめ用意された単語辞書と例文辞書を初期データとして使用しつつ，ユーザが入力した単語列を例文として追加していくことにより予測の精度を上げている．

検索システムと予測インタフェースとは全く異なる扱いをうけていることが多く，例えば辞書プログラム，文章入力システム，スペルチェッカは独立に実装されているのが普通である．しかし POBox では辞書の検索と作成が統合されているために，ひとつの機構でそれらの機能を実現することが可能になっている．近年は大規模な辞書やデータベースを手軽に利用できる機会が多くなっているが，まず既存のデータの検索にもとづく予測インタフェースシステムを構築し，それにユーザによる例データを追加していくという手法をとることにより，統合的な検索 / 予測インタフェースシステムを構成することができる．このように予測 / 例示インタフェースと検索インタフェースを融合した統合的インタフェースを**例にもとづくインタフェース** (Example-based Interface) と呼ぶ．例にもとづくインタフェースでは，例えば新たにプログラムを作成しようとする場合，ユーザの与える例データ / ユーザが以前作成したプログラム例 / 世の中に存在するプログラム例など，あらゆる例データを統合的に活用することができる．

図 15 POBox．「非常に」を確定した後，読み「おも」を指定したことにより，予測された候補単語が下部にリストされて選択可能になっている．

7.3 適応型インタフェースとの融合

計算機を使いやすくするための別のアプローチとして，ユーザの特性を何らかの方法で検出してそれに応じて挙動を変える適応型 (adaptive) インタフェース[43]が研究されている．

真の適応を行なうためにはユーザの特性の深い部分をシステムが知る必要があるが，限られたバンド幅のインタラクションから充分な情報を得ることはむずかしいため，深いレベルまでユーザに適応可能なシステムは数少ないのが現状である一方，表層的な情報を利用するだけでも効果的な適応を行なわせることが可能である．例えば SKK などの仮名漢字変換システムでは，直前に確定した単語が次回は第一候補として提示されるという単純な適応により変換効率が向上しているし，プルダウンメ

ニュー中の項目のうちよく使われるものを他の項目と分離してメニューの最上部に表示されるようにしたSplit-Menu[44]や，階層的電話帳メニューにおいてよく選択される名前がメニューの階層の上の方に出現するようにしたシステム[13]において，通常のメニュー構成をとるのに比べ操作効率が上がったことが報告されている．これらのシステムに共通しているのは，検索や予測を含むインタフェースにおいては単純な適応手段が効果的に働くということである[64]．2.1節で紹介したような各種の手法を適応的に使用することにより，より効果的な予測インタフェースを構築できる可能性がある．

8 おわりに

予測/例示インタフェースの研究の現状について簡単な解説を行なった．1993年頃までの研究のほとんどについては[8]に良くまとめられている．また，例示インタフェースとビジュアルプログラミングとの関連，特に推論手法とインタラクションの関連については[58]に詳しく，GUIに関連した最近の例示インタフェースについては[67]に詳しい解説がある．またWeb上では，MITのHenry Liebermanが予測/例示インタフェース全般についての情報をまとめたページを管理しており(http://lieber.www.media.mit.edu/people/lieber/PBE/index.html)，筆者のページでも予測/例示関連の文献情報をまとめているので利用いただければ幸いである(http://www.csl.sony.co.jp/person/masui/bib/PBE.html).

謝　辞

コメント及び“研究川柳”を提供いただいたソニーコンピュータサイエンス研究所の暦本純一氏及び図版の転載を許諾いただいた研究者の方々に感謝します．

参考文献

[1] Apple Computer, Inc.: *Welcome to Cocoa – Internet Authoring For Kids*, February 1997. http://cocoa.apple.com/cocoa/.

[2] Bell, T. C., Cleary, J. G., and Witten, I. H.: *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[3] Bos, E.: Some Virtues And Limitations Of Action Inferring Interfaces, *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92)*, ACM Press, November 1992, pp. 79–88.

[4] Brooks, R.: Watch What I Do - reviewed by Ruven Brooks, *International Journal of Man-Machine Studies*, (1993). http://www.atg.apple.com/Allen_Cypher/Watch-WhatIDo/Reviews/ijmms.html.

[5] Buxton, W., Lamb, M. R., Sherman, D., and Smith, K. C.: Towards a Comprehensive User Interface Management System, *Proceedings of SIGGRAPH*, Vol. 17,No. 3(July 1983), pp. 35–42.

[6] Charles River Analytics: *Open Sesame!*, 55 Wheeler Street Cambridge, MA 02138 USA. http://www.cra.com/products/sesame/sesameinfo.html.

[7] Cypher, A.: Eager: Programming Repetitive Tasks By Example, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, Addison-Wesley, April 1991, pp. 33–39. http://www.research.apple.com/people/cypher/Eager/Eager.html.

[8] Cypher, A.(ed.): *Watch What I Do – Programming by Demonstration*, The MIT Press, Cambridge, MA 02142, 1993. http://www.atg.apple.com/Allen_Cypher/Watch-WhatIDo/WatchWhatIDo.html.

[9] Cypher, A. and Smith, D. C.: KIDSIM: End User Programming of Simulations, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95)*, Addison-Wesley, May 1995, pp. 27–34. http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/ac1bdy.htm.

[10] Darragh, J. J., Witten, I. H., and James, M. L.: The Reactive Keyboard: A Predictive Typing Aid, *IEEE Computer*, Vol. 23,No. 11(1990), pp. 41–49. ftp://ftp.cpsc.ucalgary.ca/pub/projects/the.reactive.keyboard/.

[11] Fisher, G. L., Busse, D. E., and Wolber, D. A.: Adding Rule Based Reasoning to a Demonstrational Interface, *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92)*, ACM Press, November 1992, pp. 89–97.

[12] Furnas, G.: New Graphical Reasoning Models for Understanding Graphical Interfaces, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, Addison-Wesley, April 1991, pp. 71–78.

[13] Greenberg, S. and Witten, I. H.: Adaptive Personalized Interfaces - A Question of Viability, *Behaviour and Information Technology*, Vol. 4,No. 1(1984), pp. 31–35.

[14] Halbert, D. C.: Programming by Example, Technical Report OSD-T8402, Xerox Office Systems Division, December 1984.

[15] Hudson, S. E. and Hsi, C.-N.: A Synergistic Approach to Specifying Simple Number Independent Layouts by Example, *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems (CHI'93)*, Addison-Wesley, April 1993, pp. 285–292.

[16] Koza, J. R.: *Genetic Programming*, The MIT Press, Cambridge, MA, 1992.

[17] Kurlander, D.: *Chimera: Example-Based Graphical Editing*, chapter 12, pp. 270–290. In Cypher [8], May 1993.

[18] Landay, J. A. and Myers, B. A.: Interactive Sketching for the Early Stages of User Interface Design, *Proceedings*

of the ACM Conference on Human Factors in Computing Systems (CHI'95), Addison-Wesley, May 1995, pp. 43–50. http://www.cs.cmu.edu/People/landay/research/publications/SILK_CHI/jal1bdy.html.

[19] Lieberman, H.: An Example-Based Environment for Beginning Programmers, *Instructional Science*, Vol. 14(1986), pp. 277–292. http://lieber.www.media.mit.edu/people/lieber/Lieberary/Tinker/Tinker.html.

[20] Lieberman, H.: *Mondrian: A Teachable Graphical Editor*, chapter 16, pp. 340–358. In Cypher [8], May 1993. http://lieber.www.media.mit.edu/people/lieber/Lieberary/Mondrian/Mondrian.html.

[21] Masui, T.: Evolutionary Learning of Graph Layout Constraints from Examples, *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'94)*, ACM Press, November 1994, pp. 103–108.

[22] Masui, T. and Nakayama, K.: Repeat and Predict – Two Keys to Efficient Text Editing, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94)*, Addison-Wesley, April 1994, pp. 118–123.

[23] Maulsby, D. L. and Witten, I. H.: *Metamouse: An Instructible Agent for Programming by Demonstration*, chapter 7, pp. 154–181. In Cypher [8], May 1993.

[24] Maulsby, D. L., Witten, I. H., and Kittlitz, K. A.: Metamouse: Specifying Graphical Procedures by Example, *Proceedings of SIGGRAPH'89*, Vol. 23, No. 3, Boston, MA, July 1989, pp. 127–136.

[25] McDaniel, R. G.: Improving Communication In Programming-by-Demonstration, *CHI'96 Conference Companion*, ACM Press, April 1996, pp. 55–56. http://www.cs.cmu.edu/afs/cs/user/richm/public/www/doct96.html.

[26] Micro Logic Corp.: *KeyWatch*, POB 70, Hackensack, NJ 07602, 1990. http://www.miclog.com/.

[27] Miyashita, K., Matsuoka, S., Takahashi, S., and Yonezawa, A.: Interactive Generation of Graphical User Interfaces by Multiple Visual Examples, *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'94)*, ACM Press, November 1994, pp. 85–94.

[28] Miyashita, K., Matsuoka, S., Takahashi, S., Yonezawa, A., and Kamada, T.: Declarative Programming of Graphical Interfaces by Visual Examples, *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92)*, ACM Press, November 1992, pp. 107–116.

[29] Mo, D. H. and Witten, I. H.: Learning text editing tasks from examples: a procedural approach, *Behaviour & Information Technology*, Vol. 11,No. 1(1992), pp. 32–45. also in [8].

[30] Modugno, F.: *Extending End-User Programming in a Visual Shell With Programming by Demonstration and Graphical Language Techniques*, PhD Thesis, Carnegie Mellon University, 1995. http://www.cs.washington.edu/homes/fm/thesis-summary.html.

[31] Modugno, F. and Myers, B. A.: A State-Based Visual Language for a Demonstrational Visual Shell, *Proceedings of 1994 IEEE Symposium on Visual Languages (VL'94)*, 1994.

[32] Myers, B. A.: Creating Interaction Techniques by Demonstration, *IEEE Computer Graphics and Applications*, (1987), pp. 51–60.

[33] Myers, B. A.: *Creating User Interface by Demonstration*, Academic Press, San Diego, 1988.

[34] Myers, B. A.: Text Formatting By Demonstration, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, Addison-Wesley, April 1991, pp. 251–256.

[35] Myers, B. A.: Demonstrational Interfaces: A Step Beyond Direct Manipulation, *IEEE Computer*, Vol. 25,No. 8(1992), pp. 61–73.

[36] Myers, B. A., Goldstein, J., and Goldberg, M. A.: Creating Charts by Demonstration, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94)*, Addison-Wesley, April 1994, pp. 106–111.

[37] Myers, B. A., McDaniel, R. G., and Kosbie, D. S.: Marquise: Creating Complete User Interfaces by Demonstration, *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems (CHI'93)*, Addison-Wesley, April 1993, pp. 293–300. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/garnet/doc/papers/marquiseCHI93.abstract.

[38] Nardi, B. A.: *A Small Matter of Programming*, The MIT Press, Cambridge, MA, 1993. http://www.research.apple.com/personal/Bonnie_Nardi/ASmallMatter.html.

[39] Nix, R. P.: Editing by Example, *ACM Transactions on Programming Languages and Systems*, Vol. 7,No. 4(1985), pp. 600–621.

[40] Potter, R.: *Just-in-Time Programming*, chapter 27, pp. 513–526. In Cypher [8], May 1993.

[41] Potter, R.: *TRIGGERS: Guiding Automation with Pixels to Achieve Data Access*, chapter 17, pp. 361–380. In Cypher [8], May 1993.

[42] Roth, S. F., Kolojejchick, J., Mattis, J., and Goldstein, J.: Interactive Graphic Design Using Automatic Presentation Knowledge, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94)*, Addison-Wesley, April 1994, pp. 112–117. http://www.cs.cmu.edu/Groups/sage/sage.html.

[43] Schneider-Hufschmidt, M., Kuhme, T., and Malinowski, U.(eds.): *Adaptive User Interface – Principles and Practice*, North-Holland, Amsterdam, 1993.

[44] Sears, A. and Shneiderman, B.: Split Menus: Effectively Using Selection Frequency to Organize Menus, *ACM Transactions on Computer-Human Interaction*, Vol. 1,No. 1(1994), pp. 27–51.

[45] Singh, G., Kok, C. H., and Ngan, T. Y.: Druid: A System for Demonstrational Rapid User Interface Development, *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST90)*, October 1990, pp. 167–177.

[46] Slagle, J. R. and Wieckowski, Z.: Ideas for Intelligent User Interface Design, *Tcl'94 Workshop Proceedings*, 1994. ftp://ftp.cs.umn.edu/dept/users/wieckows/Ideas_for_Intelligent_User_Interface_Design.ps.

[47] Smith, D. C., Cypher, A., and Spohrer, J.: KIDSIM: Programming Agents Without a Programming Language, *Communications of the ACM*, Vol. 37,No. 7(1994), pp. 55–67.

[48] Takahashi, S., Matsuoka, S., Yonezawa, A., and Kamada, T.: A General Framework for Bi-Directional Translation between Abstract and Pictorial Data, *Proceedings of the ACM SIGGRAPH and SIGCHI Symposium on User Interface Software and Technology (UIST'91)*, ACM Press, November 1991, pp. 165–174.

[49] Witten, I. H.: *A Predictive Calculator*, chapter 3, pp. 66–76. In Cypher [8], 1993.

[50] Witten, I. H. and Mo, D. H.: *TELS: Learning Text Editing Tasks from Examples*, chapter 8, pp. 182–203. In Cypher [8], May 1993. 182–203.

[51] Wolber, D.: Pavlov: Programming By Stimulus-Response Demonstration, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)*, Addison-Wesley, April 1996, pp. 252–259. http://web.usfca.edu/~wolberd/chi96.html.

[52] Wolber, D. and Fisher, G.: A Demonstrational Technique For Developing Interfaces With Dynamically Created Objects, *Proceedings of the ACM SIGGRAPH and SIGCHI Symposium on User Interface Software and Technology (UIST'91)*, ACM Press, November 1991, pp. 221–230.

[53] 井田昌之, 亀井信義: Emacs 解剖学 – 入力の補完, *bit*, Vol. 29, No. 2 (1997), pp. 85–95.

[54] 小島啓二: ビジュアルインタフェースの研究動向と応用, chapter 2.9, pp. 168–175. In 平川, 安村 [60], February 1996.

[55] 佐藤雅彦: かな漢字変換システム SKK, *bit*, Vol. 23, No. 5 (1991), pp. 793–802.

[56] 杉浦淳, 古関義幸: 例示プログラミングにおけるマクロ定義の簡略化, インタラクティブシステムとソフトウェア *IV:* 日本ソフトウェア科学会 *WISS'96*(田中二郎 (編)), 近代科学社, December 1996, pp. 101–110.

[57] 中山健, 宮本健司, 川合慧: コマンド履歴からの動的スクリプト生成, インタラクティブシステムとソフトウェア *II:* 日本ソフトウェア科学会 *WISS'94*(竹内彰一 (編)), 近代科学社, 1994, pp. 155–164.

[58] 萩谷昌己: ビジュアルプログラミングと自動プログラミング, コンピュータソフトウェア, Vol. 8, No. 2 (1991), pp. 27–39. ftp://nicosia.is.s.u-tokyo.ac.jp/pub/staff/hagiya/vjidou/vjidou.dvi.

[59] 原田康徳, 宮本健司: Visible Dispatch: Visibility に基づくアプリケーション構築法, インタラクティブシステムとソフトウェア *IV:* 日本ソフトウェア科学会 *WISS'96*(田中二郎 (編)), 近代科学社, December 1996, pp. 61–70.

[60] 平川正人, 安村通晃 (編): ビジュアルインタフェース – ポスト *GUI* を目指して, bit 別冊, 共立出版, February 1996.

[61] 増井俊之: `keisen.el` プログラム, July 1991. ftp://etlport.etl.go.jp/pub/mule/contrib/keisen-mule.tar.gz.

[62] 増井俊之: 進化的学習機構を用いたグラフ配置制約の自動抽出, インタラクティブシステムとソフトウェア *II:* 日本ソフトウェア科学会 *WISS'94*(竹内彰一 (編)), 近代科学社, 1994, pp. 195–204.

[63] 増井俊之, 中山健: 操作の繰返しを用いた予測インタフェースの統合, コンピュータソフトウェア, Vol. 11, No. 6 (1994), pp. 484–492. http://www.csl.sony.co.jp/person/masui/JSSSTDmacro/JSSSTDmacro.html.

[64] 増井俊之: 適応 / 予測型テキスト編集システム, インタラクティブシステムとソフトウェア *II:* 日本ソフトウェア科学会 *WISS'94*(竹内彰一 (編)), 近代科学社, 1994, pp. 145–154.

[65] 増井俊之: *GUI* ベースのプログラミング, chapter 2.2, pp. 45–64. In 平川, 安村 [60], February 1996.

[66] 増井俊之: ペンを用いた高速文章入力手法, インタラクティブシステムとソフトウェア *IV:* 日本ソフトウェア科学会 *WISS'96*(田中二郎 (編)), 近代科学社, December 1996, pp. 51–60. http://www.csl.sony.co.jp/person/masui/POBox/.

[67] 松岡聡, 宮下健: 例示による *GUI* プログラミング, chapter 2.4, pp. 79–97. In 平川, 安村 [60], February 1996.

[68] 宮下健, 松岡聡, 高橋伸, 米澤明憲: 複数の視覚的例による直接インターフェイスの対話的実現, インタラクティブシステムとソフトウェア *I:* 日本ソフトウェア科学会 *WISS'93*(竹内彰一 (編)), 近代科学社, 1994, pp. 241–248.

[69] 宮本健司: 汎化履歴にもとづく予測インタフェースの拡大, インタラクティブシステムとソフトウェア *III:* 日本ソフトウェア科学会 *WISS'95*(田中二郎 (編)), 近代科学社, 1995, pp. 181–190.

[70] 山本格也: ビットマップに基づくプログラミング言語 Visulan, インタラクティブシステムとソフトウェア *III:* 日本ソフトウェア科学会 *WISS'95*(田中二郎 (編)), 近代科学社, 1995, pp. 151–160.

[71] 暦本純一, 長尾確: ポスト *GUI:* 今後の展望, chapter 3, pp. 178–198. In 平川, 安村 [60], February 1996.