操作列スライシングに基づく例示インタフェース

杉浦 淳† 古関義幸†

計算機上での繰返し操作をマクロやプログラムを用いて自動化することは，作業の効率化に有効である．しかし，マクロの作成のためにはプログラミングの知識が必要であり，特別な知識を持たないユーザは手作業で繰り返して操作を行わなければならない．例示プログラミング（PBD）はユーザの実演操作からプログラムを生成する手法であり，この問題を解決する有力な技術として注目されている．しかしながら，従来のPBDシステムでは，マクロ定義にかかる精神的/物理的コストが高く，PBD手法は有効には活用されていなかった．本稿では，操作列スライシングおよび自動スライシングという2つの手法を用いたPBDシステムDemoOfficeについて述べる．本システムの特徴は，これらの手法によりマクロ定義に必要な作業コストを最小化していることである．DemoOfficeでは，将来繰り返されることが予想されるユーザ操作が自動的にマクロに変換される．システムの推測が正しければ，ユーザはマクロ定義のためのいっさいの作業を行う必要がない．

Demonstrational Interface with Action Slicing Method

ATSUSHI SUGIURA† and YOSHIYUKI KOSEKI†

To automate repetitive tasks performed in computer applications, users are required to acquire special skills for writing macros or programs. Programming by demonstration (PBD), a method of converting a user demonstration into an executable code, is one possible solution to this problem. However, many PBD systems require users to spend much time and care in macro definition. This paper describes a PBD system, DemoOffice, which employs two techniques, action slicing and auto-slicing, to simplify macro definition significantly. The system is able to detect user actions which might be expected to be performed again in the future and to automatically convert those actions into a macro, for which no further definition is required.

1. はじめに

ソフトウェアアプリケーション上で繰り返して行う操作をマクロとして登録しておくことは，作業の効率化に有効である．たとえば，テキストエディタで文書中の“コンピュータ”という文字列をすべてボールドフォントにするという作業を考える．この場合，(1) 文字列“コンピュータ”を検索，(2) メニューからフォント変更コマンドを実行，という操作を何度も繰り返す必要があるが，これら一連の操作をマクロとして登録すれば，作業を自動化することが可能である．しかしながら，マクロやプログラムの作成には専門的な知識が必要とされるため，プログラミングの知識のないエンドユーザは手作業での繰返し操作を強いられることになる．

例示プログラミング（PBD: Programming by Demonstration）[2]はこの問題を解決する有力な技術の1つであるといえる．PBDはアプリケーション上でのユーザの操作からプログラムを生成する手法である．ユーザはプログラムの動作例を実演（例示）してみせるだけでよいため，プログラミングのための特別な知識を学習する必要はない．

このような特長のために，GNU Emacs，表計算ソフトなどにPBD機能が実装されてきた．しかし，これらのシステムでは多くの繰返し操作が行われるにもかかわらず，PBD機能は有効に活用されていない．この原因の1つは，マクロ定義におけるユーザの精神的負担が大きいことにある．多くのPBDシステムでは，ユーザはマクロ定義のために操作記録の開始/終了を指定しなければならない．これは，将来繰り返されるであろう操作をユーザがあらかじめ想定しなければならないことを意味する．しかし，同じような操作を2回目に繰り返すときになってはじめて，先ほどの操作をマクロ化すべきだったことに気付くのが普通である．また，仮に想定できたとしても，ユーザは例示操作の

† NEC C&C メディア研究所
C&C Media Research Laboratories, NEC Corporation

手順をプランニングし，その手順どおりに正確に例示することが要求される．これはユーザにはかなりの負担である．さらに，例示操作を行ってしまえば，目前の作業は完了してしまうため，操作記録を指定してまでユーザはマクロを定義しようとしないのである．

ユーザにとっては，マクロ生成の必要性を感じたときに過去の操作を自由に再利用できることが好ましい[7]．操作記録の指定を不要にするためには，システムがユーザの操作をつねに記録しておくという方法が考えられる．この場合，ユーザが操作履歴の中から有用な操作を選択してマクロを定義することになるが，この作業にはかなり手間がかかる．マクロを定義するコストがアプリケーション上で操作を繰り返す手間より大きければ，ユーザはマクロを定義しない．

本稿では，マクロ定義に必要なコストを最少化するための2つの手法，操作列スライシングおよび自動スライシングについて述べる．操作列スライシングは，特定のデータの作成に関連する操作だけを自動的に切り出す手法である．この手法により，ユーザは注目するデータを指定するだけで，過去の操作をマクロとして再利用できるようになる．また，自動スライシングは，将来繰り返して行われることが予想される一連の操作をシステムが検出し，自動的にマクロに変換する手法である．したがって，システムの推測が正しければ，マクロ定義のためのいっさいの作業が不要である．

これらの手法は，筆者らが開発した例示プログラミングシステム DemoOffice に実装されている．以下，次章で関連研究について述べた後，本稿で提案する2つの手法および DemoOffice の機能ついて詳細を説明する．

2. 関連する研究

Metamouse[5]，Peridot[6]など多くの PBD システムでは，プログラム（マクロ）作成のために操作記録の開始/終了を指定する必要がある．これらのシステムは，汎用性の高い複雑なプログラムを例示から作成することを目的としており，プログラミングの意識が比較的強いユーザを想定しているといえる．

それに対し，操作記録の指定を必要としない PBD システムがいくつか開発されている．これらのシステムは，ユーザがプログラムの作成を意識せずに通常の作業を行えることを目的としている．ユーザは過去の操作が再利用できると気付いたときに，ユーザ自身でプログラムを定義するか，システムが自動的に定義したプログラムを呼び出せばよい．

Chimera[3]は，ユーザの操作をつねに記録しているシステムである．ユーザは，記録された操作履歴から必要な操作を選択してマクロを定義することができる．Chimera では，操作の選択を容易にするために操作履歴をビジュアルに表現するが，それでもなお操作選択はユーザにとってかなりの負担といえる．

EAGER[1]と Dynamic Macro[4]はユーザの操作をつねに監視しており，操作履歴から繰返しパターンを検出して自動的にマクロを生成する．したがって，マクロ定義のために特別な作業は不要である．しかし，マクロに変換される操作はシステムにより決められるため，定義できるマクロに自由度はない．

3. 例示プログラミングシステム DemoOffice

3.1 概　　要

DemoOffice は，電子メールツールと表（関係データベース）を統合した PBD システムである．ユーザはこれらのツールでテキスト入力やデータの drag&drop などの標準的な編集操作を行うことができる．システム設計における基本コンセプトは，最小限の手間でマクロを生成し，過去の操作を再利用できるようにすることである．

図 1 に DemoOffice でのマクロ生成プロセスを示す．マクロ生成は (1) ユーザ操作の記録，(2) マクロ化する操作の操作履歴からの抽出，(3) 抽出された操作の一般化という3つのステップからなる．

DemoOffice では，Chimera や EAGER 同様，マクロ定義での操作記録指定を不要とするために，ユーザの操作をつねに記録している．そのため，ユーザは過去の操作が必要となった時点で初めてマクロを定義すればよい．ただし，マクロ定義のためには，記録され

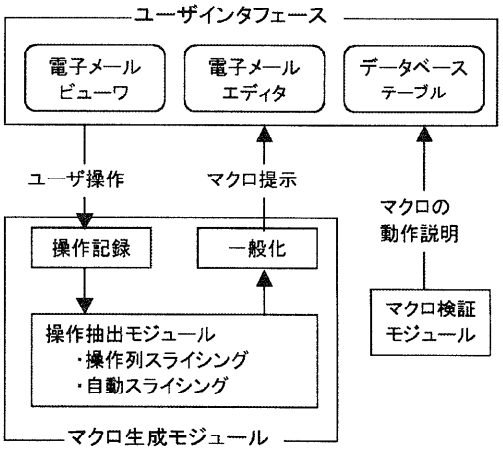


図 1 DemoOffice のマクロ生成アーキテクチャ
Fig. 1 Macro generation architecture in DemoOffice.

た操作履歴から有用な操作のみを抽出しなければならない．DemoOfficeの特徴は，この作業コストを最小化するために次の2つの手法を用いていることである．

- 操作列スライシング
  特定のデータの作成に関連する操作のみを操作履歴から切り出す手法である．この手法により，ユーザは作業中のウィンドウで注目するデータを指定するだけでマクロを定義できる．具体的には，ユーザは特定のデータをウィンドウ内のマクロバー（図2a）と呼ばれる領域にdrag&dropすればよい．Chimeraのように，操作履歴を直接見て操作を1つ1つ指定する必要はない．
- 自動スライシング
  将来繰り返されることが予想される一連の操作を自動的に検出する手法である．システムの推測が正しければマクロ定義のためのいっさいの操作が不要である．本手法では，EAGERやDynamic Macroのように操作の繰返しパターンを利用するのではなく，ユーザに操作されたデータの種類によりマクロ化する操作を決定する．そのため，ユーザは操作を繰り返さなくともマクロが利用可能となる．

システムはこれらの方法により切り出された操作を一般化し，実行可能なマクロに変換する．生成されたマクロは，ユーザが作業中のウィンドウのマクロバーにおいて，コマンドボタン（以下マクロボタン）として利用可能となる．

上記のようにDemoOfficeでは，操作の抽出，一般化というマクロ定義での重要なプロセスが自動的に行われるため，生成されたマクロの検証は重要である．DemoOfficeでは，単にマクロ生成に必要な手間だけでなく，マクロの動作検証のコストを最小化することにも注力している．DemoOfficeでは，マウスカーソルをマクロボタンの上に移動させることにより，プログラムの検証を行える．システムは，マクロ動作の簡単な説明をチップヘルプで行い，さらにプログラムの実行結果を前もってユーザに示す．その結果が所望のものであれば，ユーザは実際にボタンをクリックしてマクロを実行すればよい．

3.2 例　　題

本節では例題を用いてシステムの動作を説明する．ここでの例題は，電子メールとデータベースを用いた集計作業である．ある人の歓送パーティを行うことになり，その出欠を電子メールで確認するといった状況を考える．最初にユーザが次のメールを友人に送ったとする．

Hi all,

I will be giving a farewell party for Mary next Sunday. Please let me know if you can come or not. Anything you might bring to drink would be appreciated.

--- cut here ---

Attendance: Yes/No

I will bring: ___

--- cut here ---

Regards, Atsushi

ユーザの目的は返信メールの中から必要なデータだけをデータベース表に挿入することである．例示のトレースを**図2**に示す．

最初の返信メール（図2a）では，ユーザは出欠の回答“Yes”を選択し，表Table1の1行目へdrag&dropでコピーする（図2a）☆．システムはこの操作が他のメールに対しても行われる可能性が高いと判断し，マクロボタンMacro1を自動的に生成する（図2b）．さらにユーザは，差入れの飲み物“red wine”およびメールアドレスを同様にコピーする（図2c）．システムはこれらの操作も再利用性が高いと判断し，Macro1に追加する．したがって，この時点でMacro1には表の1行目を作成した全操作が含まれる．

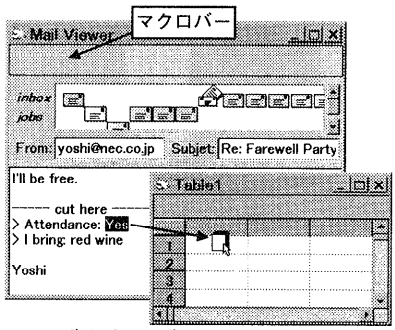
2番目の返信メール（図2d）を開封した後，ユーザは1番目のメールでの操作を再利用できる可能性があることに気付く．そこでユーザは，先ほどシステムにより生成されたMacro1が利用可能であるかを調べるために，Macro1ボタンの上にマウスカーソルを移動させる．図2dに示すように，チップヘルプによる簡単な説明が現れ，Table 1にはマクロの動作結果が示される☆☆．これはユーザが期待したとおりの結果であるため，ユーザはボタンをクリックし実際にマクロを実行する（図2e）．

3番目の返信メール（図2f）は指定したフォーマットに従っておらずMacro1を使用できない．そのため，ユーザは自ら操作して表の3行目を作成する．まず，ユーザは，出欠の返事“No”を表に直接入力し，さらに，差入れの“5 bottles of beer”およびメールアドレスを表へコピーする（図2f）．システムはこの行を作成した操作が有用であると判断し，新しいマクロMacro2を生成する．

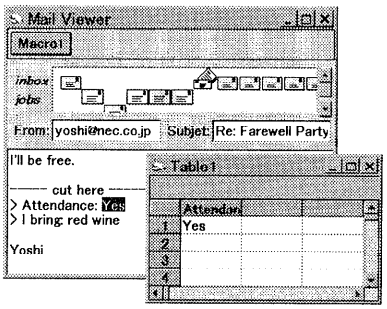
4番目の返信メール（図2g）も3番目のメール同様にフォーマットに従っていない．そのため，ユーザは

☆ 表のフィールド名“Attendance:”は，システムにより自動的にコピーされる．

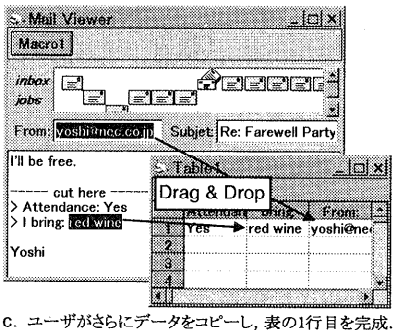
☆☆ マウスカーソルをボタンから外せば，元の状態に戻る．



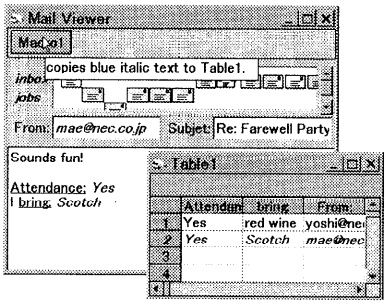
a. ユーザが1番目の返信メールで"Yes"を選択し，drag&dropで表へコピー．



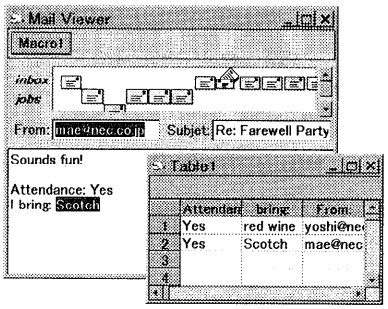
b. システムは，aでのユーザの操作に相当するマクロ"Macro1"を生成．



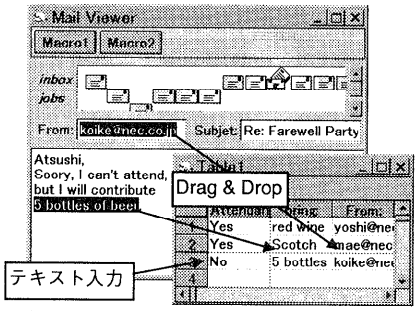
c. ユーザがさらにデータをコピーし，表の1行目を完成．システムがMacro1の内容を更新．



d. 2番目のメールにMacro1を適用可能かを調べるために，ユーザはMacro1ボタン上にマウスカーソルを移動．システムはマクロの動作を説明．



e. dのシステムの説明により所望のマクロが生成されていることが確認できたため，ユーザはMacro1ボタンをクリックしてマクロを実行．



f. 3番目のメールにおいては，ユーザはキーボードから"No"を入力し，メールから表へデータをコピー．システムは新たにMacro2を生成．

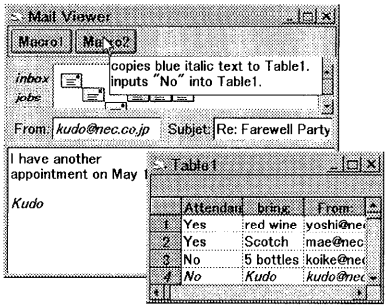
図2 例示のトレース（続く）
Fig. 2 Trace of demonstration (continued).

Macro2 が利用できる可能性があると考え，Macro2 の動作検証を行う．しかし，図 2g に示されるように，Macro2 をそのまま適用することはできない．1 列目と 3 列目は期待どおりの結果であるが，2 列目（bring: の列）の "Kudo" は余分である．そこでユーザは，Macro2 の機能（表の 3 行目を作成した操作）を部分的に利用するために，自らマクロを定義する．具体的には図 2h に示すように，表の (3, 1)，(3, 3) の 2 つのセルを選択し，メールウィンドウのマクロバーに drag&drop する☆．システムは，これら 2 つのセルの作成に関連する操作のみを抽出し，マクロ Macro3 を生成する．ユーザは Macro3 を実行し，表の 4 行目を作成する（図 2i）．

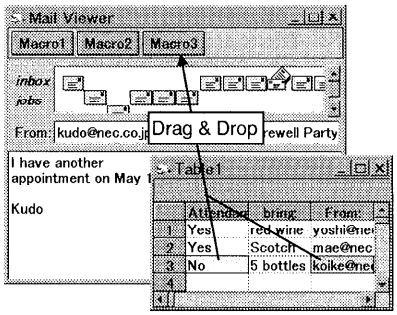
3.3 操作記録

マクロ生成のための最初のステップはユーザ操作の

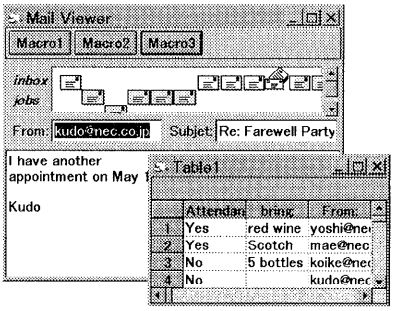
☆ 本例題ではユーザはメールウィンドウにマクロを生成したが，任意のウィンドウに生成可能である．



g. ユーザは4番目のメールにおいて，Macro2の動作を検証．しかし，このマクロは不適切．



h. 表のセル(3, 1)と (3, 3)の作成において行った操作をマクロとして再利用するために，ユーザはこれらのセルを選択しマクロバーへdrag&drop．



i. ユーザはMacro3を実行し，表の4番目の行を作成．

図 2 例示のトレース（続き）
Fig. 2 Trace of demonstration.

```
1:  openTable("Table1")
2:  openMail("Mail Viewer", 155)
3:  text = selectText("Mail Viewer", Body, 68, 71)
4:  drag("Mail Viewer", Body, text)
5:  drop("Table1", 1, 1)
6:  text = selectText("Mail Viewer", Body, 84, 92)
7:  drag("Mail Viewer", Body, text)
8:  drop("Table1", 1, 2)
9:  text = selectText("Mail Viewer", From, 0, 19)
10: drag("Mail Viewer" From, text)
11: drop("Table1", 1, 3)
12: openMail("Mail Viewer", 156)
13: execute("Macro1")
14: openMail("Mail Viewer", 157)
15: cell = selectCell("Table1", 3, 1)
16: inputText("Table1", cell, "No")
17: text = selectText("Mail Viewer", Body, 51, 68)
18: drag("Mail Viewer", Body, text)
19: drop("Table1", 3, 2)
20: text = selectText("Mail Viewer", From, 0, 18)
21: drag("Mail Viewer", From, text)
22: drop("Table1", 3, 3)
23: openMail("Mail Viewer", 158)
24: execute("Macro3")
```

a. 操作履歴．

スライシング

```
1:  openTable("Table1")
14: openMail("Mail Viewer", 157)
15: cell = selectCell("Table1", 3, 1)
16: inputText("Table1", cell, "No")
20: text = selectText("Mail Viewer", From, 0, 18)
21: drag("Mail Viewer", From, text)
22: drop("Table1", 3, 3)
```

b. スライシングにより切り出された操作．

一般化

```
Macro2(currentMail) {
    openTable("Table1")
    openMail("Mail Viewer", currentMail)
    cell = selectCell("Table1", LAST_ROW + 1, 1)
    inputText("Table1", cell, "No")
    text = selectText("Mail Viewer", From,
                      TEXT_HEAD, TEXT_END)
    drag("Mail Viewer", From, text)
    drop("Table1", ACTIVE_ROW, 3)
}
```

c. 生成されるマクロ．

図 3 例題の Macro2 の生成
Fig. 3 Generation of Macro2 in the example task.

記録である．システムはユーザが行ったすべての操作を記録する．DemoOffice はデータの直接操作で編集作業を行うシステムであるため，記録される操作は，操作と操作対象となったデータの組で表現される．図 3 a は図 2 の例題でのユーザの操作記録である．たとえば，図 3 a の 3 行目は，“Mail Viewer” ウィンドウの Body（本文）フィールドの 68 番目から 71 番目の文字を選択した操作を表している．また，5 行目は Table 1 のセル (1, 1) にデータをドロップしたことを表している．

3.4 操作抽出

マクロ生成のための2番目のステップは，操作履歴からマクロとして必要な操作のみを抽出することである．

PBDシステムおいて，ユーザの通常の編集作業を極力妨げることなく有用な操作のみを抽出するにはどうすればよいであろうか? この問いに対する筆者らの解答は，(1) 必要な操作を検索するための検索キーとして作業中のウィンドウ上にあるデータを利用すること，(2) ユーザの代わりにシステムが自動的に操作を抽出することである．

3.4.1 操作列スライシング

操作列スライシングは注目するデータの作成に影響を及ぼした操作のみを操作履歴から切り出す手法である．以下，操作列スライシングで注目するデータをスライス基準と呼ぶ．

この考え方は，プログラムスライシング[8]に類似している．プログラムスライシングは，プログラム中のある文の実行に影響を与える可能性のある文だけを切り出す技術である．概念的には類似しているものの，スライシングの手続きは操作列スライシングの方がはるかに単純なものですむ．プログラムスライシングでは，ループや条件分岐などの制御構造，ポインタなどの複雑なデータ形式，複数手続き間にまたがる影響などを考慮しなければならない．これに対し操作列スライシングでは，データ間の依存関係を調べるだけでよい．

表やメールの内容が変更されるのは，テキストが入力されるか，データがドロップされた場合である☆．そこで，操作列スライシングではそれらの操作（inputText, drop）を起点とし，操作履歴を遡って関連する操作を抽出する．たとえば，図2の表Table1のセル(3, 1)がスライス基準である場合，セル(3, 1)へのテキスト入力である図3の操作16を切り出し，さらに変数cellによる依存関係から操作15を切り出す．

drop操作が起点となる場合は，変数による依存関係に加え，再帰的な操作抽出処理が必要となる．drop操作に対しては，その直前のdrag操作およびdrag元のデータの選択操作が，必ずスライシングの対象となる．また，これらの操作以前に，drag元のデータの内容を変更した操作が操作履歴に存在する可能性があるため，drag元のデータを新たにスライス基準として再帰的にスライシングを行わなければならない．

☆ これはエディタの機能に依存する．一般には，コピー&ペーストや削除などの編集操作も対象となる．

たとえば，スライス基準が表のセル(3, 3)である場合，セル(3, 3)へのdrop操作である操作22を切り出した後，直前のdrag操作21とdragするデータを選択した操作20を切り出す．さらに，drag元のデータの作成に影響を与えた他の操作を切り出すために，“Mail Viewer”のFromフィールド（メールアドレス）のテキストを新たなスライス基準として操作20以前の履歴を検索する．ただし，図3aの操作履歴にはメールアドレスの内容を変更する操作はなく，これ以上抽出されない．

したがって，例題の図2hのようにユーザが表のセル(3, 1)および(3, 3)をスライス基準として選択した場合，図3bに示す操作が抽出される．

3.4.2 自動スライシング

自動スライシングは将来繰り返される可能性の高い操作を操作履歴から検出し，それら一連の操作を切り出す手法である．操作抽出は操作列スライシングを利用して行われる．したがって，このプロセスは，(1) ユーザ操作の再利用性判定，(2) 操作列スライシングのためのスライス基準の設定，という2つのステップからなる．

再利用性判定：DemoOfficeは，ユーザに操作されたデータが特定の集合に属する集合データであるかに着目して，操作の再利用性を判定する．集合の1要素に対する操作は同じスキーマからなる他の要素に対しても行われる可能性が高い．そのため，集合データを扱ったユーザ操作をマクロ化の対象とする．DemoOfficeでは受信電子メール，データベース表が集合データである．受信メールは，フォルダごとに分類されている集合データである．また，表は同一スキーマで構成されるレコード（行）の集合からなる．

したがって，図2の例題のように，集合の1要素である受信メールのデータを用いて表の1行を新規に作成した操作は，マクロ化の対象となる．また，表の1レコードのデータを利用して送信メール作成するといった操作もマクロに変換される．

しかしながら，表の行や送信メールが新たに作成されるたびに新たなマクロが生成されては，ユーザはマクロの管理が困難になる．そのため，DemoOfficeではマクロ生成を一部制限している．

まず，新規作成のための操作がテキスト入力のみの場合は，マクロを新規に生成しない．入力されたテキストはアプリケーション上に表示され，操作そのものがデータとして残るからである．また，ユーザが行った操作を既存のマクロで代用できる場合はマクロを生成しない．さらに，自動生成したマクロの中で利用頻

度の低いものを削除してマクロが一定の数を超えないようにしている．

スライス基準の設定：再利用性判定後に，システムはスライス基準を設定し操作列スライシングを行う．ここでは，再利用性の高い操作をユーザにとって意味のある単位でまとめてマクロに変換することが重要である．たとえば，すべてのdrag&drop操作を1つずつマクロに変換してもユーザにとっては好ましくない．DemoOfficeでは，集合の要素単位（表の1行全体やメール全体）で操作をまとめてマクロを生成する．すなわち，集合データの1要素が編集された場合，その要素全体をスライス基準とする．

例題では，ユーザが最初に表の1行目にデータをコピーした操作（図2b，図3の操作5）に対し，システムがマクロMacro1を生成している．この場合，システムは表の1行目をスライス基準とする．その後，表の1行目にデータがコピーされるごとに（図2c，図3の操作8および11），システムは1行目全体をスライス基準としてスライシングを行い，Macro1の内容を動的に更新する☆．結果として，出欠の返事，差入れの飲物，メールアドレスをコピーした操作が1つのマクロに変換される．

3.5 一般化

マクロ生成のための最後のステップは，抽出した操作の一般化である．特定のデータに対して行われた操作を他のデータに適用するためには，ユーザの操作を抽象レベルで解釈しなければならない．

DemoOfficeで一般化の対象とするのはテキスト選択と表のセル選択の2種類の操作である．テキスト選択に関しては，行の先頭/末尾，テキストの先頭/末尾，選択個所の前後の単語，といった情報を用いて一般化を行う．たとえば，図2aでメールウィンドウ中のテキスト“Yes”を選択した操作（図3操作3）に対しては，「メールBodyの文字列“Attendance:”の次の文字から行末までを選択」というように一般化を行う．

また，表のセル選択に関しては，最終行/列，フォーカスがあるアクティブな行/列といった一般化を行う．たとえば，図2fでテキスト“No”を入力したセル(3, 1)を選択した操作（図3操作15）には，「表の最終行の次の行を選択」という解釈を与える．

3.6 マクロの検証

上記のようにDemoOfficeはマクロ生成のためにいくつかの推論を行う．そのため，生成されたマクロがユーザの期待どおりに動作する保証はない．マクロの実行に先立ちその動作を検証できることが重要である．

DemoOfficeでは，マクロボタンの上にマウスカーソルを移動させることにより，マクロの検証を行うことができる．このとき，システムは次の4種類の方法でマクロの動作をユーザに説明する（図2d，g参照）．

(1) チップヘルプによりマクロの概要を文章で説明する．システムは，ユーザ操作に対応する説明文のテンプレートを有しており，テンプレート中のパラメータを実データで置き換えることによりこの文章を生成する．たとえば，テキスト入力操作に関しては，次の説明文テンプレートを有している．

inputs *String* into *WindowName*

テンプレート中でイタリックで記述された箇所が変数である．この場合システムは，ユーザにより入力された文字列，およびテキスト入力が行われたウィンドウ名を変数に割り当て，チップヘルプの説明文を生成する．

(2) マクロ実行で操作対象となるデータをハイライトする．図2dのメール中のイタリックフォントのテキスト“mae@nec.co.jp”，“Yes”，“Scotch”がこれにあたる．

(3) テキスト選択操作に関しては，選択の基準となった単語に下線を引く．これは，システムの一般化結果をユーザに提示することを目的としている．図2dではMacro1は“Attendance:”および“bring:”に着目してテキスト選択を行うことを示している．

(4) プログラムの実行結果を前もって示す．図2d)の表2行目のイタリックのテキストがこれにあたる．

4. 評価

DemoOfficeでのマクロ定義手法の有効性を確認するために，1人の被験者により簡単な実験を行った．実験での課題は，28通の返信メールから必要なデータを抜き出し表に登録するという，アンケート集計作業である．

実験の結果，作業を通して6個のマクロが自動的に生成され，被験者はそのうちの2個のマクロを実際に使用して22通のメールを処理を行った．被験者自らはマクロを定義しなかった．筆者らの観察と被験者へのインタビューによれば，不必要なマクロが生成されることに関しては多少苛立つものの，被験者は状況に応じて利用すべきマクロを容易に選択することがで

☆ マクロの内容が変更されたことをユーザに知らせるためにDemoOfficeはマクロボタンを点滅させる．

きた．マクロの使い分けには，チップヘルプやマクロの実行結果の表示などのマクロ検証機能が特に有効であった．

また，ユーザ自身がマクロ定義可能であるか（操作列スライシングのためのスライス基準データを選択可能か）を検証するために，マクロ自動定義機能を外して上記の作業を行ってもらった．被験者は適切にスライス基準を指定することができ，3個の有用なマクロを作成することができた．

5. 制　　限

本稿で提案した操作列スライシング手法により，マクロ定義に必要な作業がほとんど不要ではあるものの，定義できるマクロ（抽出できる操作）には制限がある．

操作列スライシングでは，データの内容に何らかの変更があった場合に，その変更に関連する操作のみを抽出する．したがって，データの内容に影響を及ぼさない操作のみをマクロとして定義することはできない．たとえば，単にウィンドウを開くだけのマクロを定義することはできない．

また，操作列スライシングでは，スライス基準であるデータの作成に関連するすべての操作が切り出される．そのため，過去のある時点から現時点までの差分や，データの作成に関連する一部の操作のみをマクロとして定義することはできない．たとえば，テキストエディタで，(1) 文字列“コンピュータ”を入力，(2) その文字列をボールドフォントに変更，(3) さらに文字サイズを24 pt に変更，といった操作を行ったとする．操作列スライシングにより，(1)～(3) のすべての操作を抽出することは可能であるが，(2)，(3) の操作のみを抽出することはできない．

テキストエディタの場合，(1)～(3) と同様の作業を行いたければ文字列を選択してコピーすればよく，これらすべての操作をマクロ化する必要性は少ないと考えられる．むしろ，既存の文字列のフォント属性を変更する (2)，(3) の操作の方が有用であるといえる．再利用性が高く，有用なマクロを生成するためには，対象アプリケーションや利用されたコマンドの性質などを考慮してスライシングを行う必要がある．

6. おわりに

本稿では，操作列スライシングと自動マクロ定義というPBDでのマクロ定義を簡略化する2つの手法について述べた．これらの手法により，PBD機能の有効な利用が促進されると期待できる．一般に，PBDシステムで生成されるプログラムはユーザの期待どおりに動作する保証がない．そのため従来は，ユーザは手間をかけてマクロを作成しようとはしなかったのである．しかし，DemoOfficeのようにマクロが自動的に作られるならば，仮に生成されたマクロが期待どおりに動作しなくとも，そのマクロを使わなければそれですむ．ユーザ自らマクロを定義した場合でも，定義に必要な作業コストが少ないためユーザの受ける被害は少ない．そのために，過去の操作が再利用されるケースが増えると思われる．

DemoOfficeでは，マクロバーを導入することにより，複数のマクロの使い分けを可能にしている．これはPBDシステムの実用性を向上させるためには重要である．一般にPBDでは，様々な状況に対応できる汎用性の高いプログラムの作成は困難である．たとえば，例題のような電子メールでの集計操作では，指定したフォーマットに従っていないデータやノイズを含んだデータを扱う必要がある．また，例題の3番目のメールのように文書の内容にまで踏み込まないと行うべき処理を決められない場合がある．それらの複雑な処理をすべてプログラム化して1つのマクロを生成するのは難しい．例題で示したように，状況に応じてユーザ自身が利用すべきマクロを決めることにより，効率良く作業を行うことが可能となる．マクロバーは必要なときにのみ表示すればよく，ユーザの作業の妨げになるものではないと考えている．

自動スライシングには，マクロ定義に必要なコストを最小化するという利点がある反面，ユーザの意志とは関係なくマクロを生成してしまうという側面もある．3.4.2項で述べたように，DemoOfficeではマクロ生成に制限を設けているが，それでも利用されることのない不要なマクロが生成されてしまう場合がある．ただし，不要なマクロの削除などマクロバーの編集機能を充実させれば，実用上は問題ないと考えている．

今後は，前章で述べた制限事項を解決したうえで，操作列スライシング手法をテキストエディタ，グラフィカルエディタ，表計算ソフトなどにも適用していく．

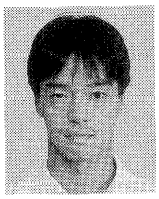
謝辞　本研究を行う機会を与えてくださったNEC C&Cメディア研究所後藤敏所長および阪田史郎統括部長に感謝いたします．

参考文献

1) Cypher, A.: Eager: Programming Repetitive Tasks by Example, *Proc. CHI '91*, pp.33–39, ACM (1991).
2) Cypher, A. (Ed.): *Watch What I Do: Programming by Demonstration*, MIT Press

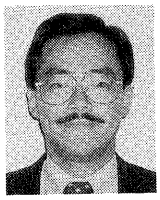
(1993).
3) Kurlander, D. and Feiner, S.: A History-Based Macro by Example System, *Proc. UIST '92*, pp.99–106, ACM (1992).
4) Masui, T. and Nakayama, K.: Repeat and Predict – Two Keys to Efficient Text Editing, *Proc. CHI '94*, pp.118–123, ACM (1994).
5) Maulsby, D.L., Witten, I.H. and Kittlitz, K.A.: Metamouse: Specifying Graphical Procedures by Example, *Proc. SIGGRAPH '89*, Vol.23, No.3, ACM (1989).
6) Myers, B.A.: *Peridot: Creating User Interfaces by Demonstration*, Chapter 6, pp.125–153, MIT Press (1993).
7) Potter, R.: *Just-in-Time Programming*, Chapter 27, pp.513–526, MIT Press (1993).
8) Weiser, M.: Program Slicing, *IEEE Trans. Softw. Eng.*, Vol.SE-10, No.4, pp.352–357 (1984).

(平成 9 年 6 月 26 日受付)
(平成 10 年 1 月 16 日採録)



杉浦　淳（正会員）

1988 年大阪大学工学部電気工学科卒業．1990 年大阪大学大学院工学部電気工学専攻修士課程修了．同年日本電気（株）に入社．現在，同社 C&C メディア研究所所属．エキスパートシステム構築ツール，ビジュアルソフトウェア，例示プログラミングに関する研究開発に従事．1995 年度人工知能学会論文賞．人工知能学会，ソフトウェア科学会各会員．



古関　義幸（正会員）

1979 年東京工業大学工学部情報工学科卒業．1981 年 University of California, Los Angeles（UCLA）において Master of Science in Computer Science 取得．同年日本電気（株）に入社．1987 年 Stanford 大学 Computer Science 学科客員研究員．現在，同社 C&C メディア研究所研究課長．工学博士．人工知能の応用，知識獲得，ビジュアルソフトウェアに関する研究開発に従事．1995 年度人工知能学会論文賞．人工知能学会，IEEE，AAAI 各会員．