ICOT Technical Memorandom: TM-0960

TM-0960

ユーザモデルを利用した
ユーザインタフェースのカスタマイズ

阪田　企弘（日本電気）
上田　賢省（神奈川大学）

October, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

ユーザモデルを利用したユーザインタフェースのカスタマイズ

○坂田 金弘 (日本電気(株) ソフトウェア生産技術開発本部)
*土田 賢省 (神奈川大学 工学部)

User Interface Customization Using User Models

Masahiro SAKATA
Software Engineering Development Laboratory, NEC Corporation
2-11-5, Shibaura, Minato-ku, Tokyo 108, Japan

*Kensei TSUCHIDA
KANAGAWA University
3-27, Rokkakubashi, Kanagawa-ku, Yokohama, 221, Japan

Abstract.

This paper describes a method for user interface customization using User Models. This method consists of two steps;

(1) Modeling each user's knowledge and recognition of user interface ,which we call User Models, through examples.

(2) Customizing user interface for each user using the User Models.

With the change of user's recognition by experience or practice, the User Models are changed and user interface is customized.

Keywords. User Models; User Interface Customization;

1 はじめに

ソフトウェアの評価において, ユーザインタフェースの善し悪しは非常に大きなウェイトを占める. 言い替えれば, ユーザインタフェースのできがそのソフトウェアの評価を決定すると言っても過言ではない. そのため, ユーザインタフェース作成のためのガイドライン (UIGL) [1] や, ユーザインタフェース構築ツール [2] 等の良いユーザインタフェースを構築するための手段が用意されている.

しかし, ガイドラインに沿ったユーザインタフェースを設計し, ユーザインタフェース構築ツールを用いたとしても, それが全ての人に使いやすいと感じられるとは限らない. なぜならば, このようにして作られたものは, 設計者, 構築者が使いやすいと感じたユーザインタフェースで, それ以外の人 (ユーザ) が必ずしもそのユーザインタフェースで満足するとは限らない. これは, 個人個人でユーザインタフェースの使いやすさに対する評価基準が異なるからであると考えられる. そこで, ユーザ自身でユーザインタフェースを変更し, 自分にあったものにカスタマイズできることがどうしても必要となる.

本稿では, 各個人の持つユーザインタフェースに関する認識や使いやすさの評価基準をモデル化 (ユーザモデル) し, それを参照してユーザインタフェースをカスタマイズする方式を提案する.

2 ユーザモデル

2.1 ユーザインタフェースの評価

前章で述べたように, ユーザインタフェースの使いやすさの評価基準は, 各個人で異なるものである.

例えば, エディタ等における文字列消去の命令を実行する際, あるエディタでは消去する文字列をまず選択し, それから消去コマンドを発生させ実際にその文字列を消

去させるインタフェースでこれを実現しており，また別のエディタでは，まず消去コマンドを選択し，それから消去したい文字列を選択することでインタフェースを実現している．これは，それぞれのエディタの設計者が，どのようなインタフェースが使いやすいかを検討した結果から得られた設計方針が異なるためで，すなわち評価基準の違いがあらわれたものといえる．

また，Multi-Window システムにおいて，Window をタイリングで配置するか，オーバーラップにするか，ということは開発者側の評価で決定されるもので，やはり，その評価の違いがあらわれている．

そのうえ，ユーザインタフェースの評価は利用者の習熟度によっても異なるものである．例えば，HELP 等のメッセージでは，初心者のユーザに対しては詳しい説明が必要であるが，そのシステムを習熟したユーザに対しては，メッセージは簡潔なもので済むようになる．

そして，そのユーザが今までにどのようなユーザインタフェースを経験したか，どのような環境で作業してきたかによっても，評価基準は異なってくる．例えば，UNIX の vi のようなモードを持ったエディタを常に使っている人にとっては，モードを持った他のエディタをあまり使いにくいとは思わないかも知れないが，モードのないエディタを使っている人にとっては，vi のようなモードを持ったエディタを使いにくいと感じるであろう．また，Macintosh[1]のように，ユーザインタフェースガイドラインが明確に打ち出されているようなシステムでは，異なるアプリケーションでも同じ作業を行なうときの操作が同じようになっているので，ユーザは新しいアプリケーションを使う場合でも，基本的な機能については，他のマシンの新しいアプリケーションよりも比較的簡単に使いこなせるようになる．

2.2 ユーザインタフェースのカスタマイズ

すべてのユーザの評価基準にあったユーザインタフェースを構築することはできない．そこで，これを解決する手段としてはそのユーザ自身がユーザインタフェースを変更し，自分の使いやすいと感じるユーザインタフェースを構築する方法が考えられる．

ユーザが，まず与えられたユーザインタフェースで作業を行ない，その中でユーザインタフェースのどの部分を変更すれば良いかを判断し，ユーザ自身でその部分をカスタマイズすることで，自分にあったユーザインタフェースが実現できる．こうすれば初心者は初心者なりの，熟練者は熟練者なりのユーザインタフェースで作業が可能となる．

実際のカスタマイズの方法として，環境変数や環境設定ファイルを利用する方法が現在用いられている．例えば，エディタに自分にあったコマンドのキーバインディングを設定したり，フォントや色を環境設定ファイルから指定することができる．また，alias(別名) 機能なども自分にあったコマンド名の設定であるから，これも一種のユーザインタフェースのカスタマイズといえる．

2.3 ユーザモデル

一般的にユーザモデルとは，システム利用中のユーザがおかれている状況や，ユーザの理解の度合等を表現するものとして利用されている．これは，ユーザの状況や理解の度合などは個々のユーザ毎に異なり，システム側ではそれに応じた応答が必要となるからである．

本章で述べてきたように，ユーザはそれぞれにユーザインタフェースに関するモデルを持っており，そのモデルにあったユーザインタフェースを使いやすいと感じ，それに相反するユーザインタフェースについては使いにくいと感じるものと考えられる．

また，このユーザの持つユーザインタフェースに関するモデルは，ユーザの経験によって変化するものと考えられる．前述のように今まではわかりやすく思っていたメッセージが冗長だと感じるようになるのは，理解度の高まりにつれてモデルが変化したためであると考えられる [5]．

逆に見ると，これらのことから，このモデルに沿ったユーザインタフェースが個々のユーザに与えられれば，それはそのユーザにとって非常に使いやすいと評価されるものである，と言える．そこで，前述のようなユーザインタフェースに関するモデルをユーザモデルとし，ユーザモデルを利用したユーザインタフェースのカスタマイズを行なうことを考える．

2.4 ユーザモデルの階層性

ユーザモデルには，個々のアプリケーションに対応するための部分と，全てのアプリケーションに共通な部分とに分かれる．

個々のアプリケーションに対応する部分のユーザモデルは，そのアプリケーション特有のユーザインタフェースに関する特徴を示しており，例えば前述の vi のようなエディタなら，その終了方法や入力モードへの移行のた

[1] Macintosh is a trademark of Apple Computer, Inc.

めのコマンドなどは，viのユーザモデルに記述されるものである．

このモデルがアプリケーション起動時に参照されれば，そのモデルに沿ったユーザインタフェースの環境でユーザは作業を行える．ただし，このモデルはそのアプリケーションのユーザインタフェースでだけ有効である．これは，環境設定ファイルを読み込んでそのユーザに適した環境を設定するのと同じ方式である．

また，全アプリケーションのユーザインタフェースに共通な記述の部分のユーザモデルも考えられる．これは，そのユーザが持つユーザインタフェースに関する一般的な概念であると考えられる．また，前述の個々のアプリケーションに対するユーザモデルの上位概念ととることもできる．すなわち，アプリケーションに対するユーザモデルの共通部分から取り出した上位概念が共通部分のモデルとなる．

例えば，文字Fontは，個々のアプリケーション毎にも設定できるべきであるが，すべてのアプリケーションで使われるFontをそのユーザが読みやすいと思うFontに統一することも考えられる．これは，各アプリケーション毎のFontという概念の上にあるもので，これは上位概念と捉えることができる．

ただし，これは対象となるアプリケーションのユーザインタフェースがこの上位概念と同じ枠組みの概念に沿って構築されているときにのみ有効で，そのときにはユーザインタフェースに対して，カスタマイズが可能となる．すなわち，アプリケーションがある上位概念に沿って構築されており，ユーザが同じ枠組みの異なる上位概念を持つときには，そこでカスタマイズが行われ，ユーザが持つ上位概念に沿ったアプリケーションのユーザインタフェースのモデルが獲得される．

3　ユーザモデルを利用したカスタマイズ方式

3.1　例示からの一般化手法の応用

本章では，ユーザインタフェースをカスタマイズする方式として，例示からの一般化手法 [3],[4] を取り入れた方式を提案する．

例示からの一般化手法の利点の1つに，全ての部分について個々に定義する必要がなく，ある程度の部分を例として与えることで，それを一般化し，そこから全ての部分を決定することができる，と言うことが挙げられる．

この手法をユーザインタフェースに応用すると，例示からの一般化によってユーザモデルを獲得し，それに従ってユーザインタフェースをカスタマイズするという方式が考えられる．この方式では，例えば，3つ以上の場面を持つようなユーザインタフェースに対して，2つの場面の部分をカスタマイズすることで，新しいユーザモデルが獲得され，その結果，残りの場面へ反映させることで，その他の場面のカスタマイズの必要がなくなる．

すなわち，例示からの一般化手法によるユーザインタフェースのカスタマイズでは，ある部分についてのカスタマイズからそれを一般化し，全体に反映させることができる．カスタマイズしたい部分をすべて定義する必要が無いので，これによってユーザはカスタマイズの手間を大幅に省くことができる．

前章で述べたような環境設定ファイルによる読み込みでは，その設定はあくまでも1つのアプリケーションのユーザインタフェースの設定であった．しかし本方式では，例示からの一般化によってユーザモデルを獲得し，ユーザモデルの階層性から，それを上位概念のモデルへと反映させることで，他のアプリケーションのユーザインタフェースも変更させることができる．

3.2　コマンド操作系列のカスタマイズ

本節では，アプリケーション(テキストエディタ)におけるコマンドの操作系列のカスタマイズを例に挙げ，前述のカスタマイズ方式について説明し，同時に，その結果獲得されたユーザモデルから，他のアプリケーションへモデルを反映させる方式について説明する．

ここでは，話を簡単にするためにコマンドとして，COPY(文字列の複写)，MOVE(文字列の移動)，CUT(文字列の削除)の3つの文字列処理のコマンドだけを考える．そして，これらのコマンドの操作系列として，

COPY: 文字列選択 → コマンド選択 → カーソル移動 → 実行確認

MOVE: 文字列選択 → コマンド選択 → カーソル移動 → 実行確認

CUT: 文字列選択 → コマンド選択 → 実行確認

がシステム側からデフォールトとして与えられているものとする．コマンド実行の為の各操作を，コマンド名とその引き数の決定のための操作と考えると，これらは各コマンドの引数が，

COPY: 複写範囲(選択)，複写先(カーソル移動)

MOVE: 移動範囲(選択)，移動先(カーソル移動)

CUT: 削除範囲(選択)

を持つものとして，それを決定するための操作の系列で

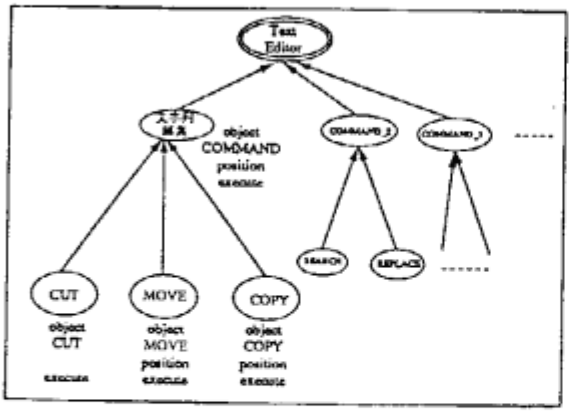

図 1: テキストエディタのコマンド群 (抜粋)

あると見ることができる．また，これはシステム開発者が 3 つのコマンドの上位概念である '文字列編集' において，

文字列編集: 対象の選択 → コマンド選択 → (目的の選択) → 実行確認

の操作系列モデルを持ち，これに基づいて各コマンドの操作系列を決定しているものと考えられる (図.1)．

以下で，ユーザがユーザモデルを持たない時 (与えられたデフォールトのインタフェースで操作) に，操作系列を変更する場合を考える．

(1) CUT コマンドの操作系列の変更

テキストエディタ実行中に，ユーザが CUT コマンドの操作系列を，

CUT: コマンド (CUT) 選択 → 文字列選択 → 実行確認

に変更したい場合を考える

ユーザは，カスタマイズモードに入りコマンド操作系列の変更を以下の手順で行う．

1. 操作系列を変更したいコマンド CUT を選択
2. 現在の操作系列での CUT コマンドを実行
3. CUT の操作系列 (コマンド選択, 文字列選択, 実行確認) を入力
4. 変更を指示

これによって，CUT コマンドの操作系列の変更が指示される．

また，システムでは以下の操作が行われる．

1. テキストエディタの CUT コマンドの系列が変更される．
2. ユーザモデルに現在変更した CUT コマンドの操作系列を登録する．
3. ユーザモデル内で一般化可能なものがあれば一般化をおこない，ユーザモ デルに付け加える．

ここでは，ユーザモデルにそれまでに登録されているものがないので一般化は起こらない．

(2) MOVE コマンドの操作系列の変更

次に，MOVE コマンドの操作系列を，

MOVE: コマンド (MOVE) 選択 → 文字列選択 → カーソル移動 → 実行確認

に変更する場合を考える．

ユーザは，(1) と同様の操作でコマンド操作系列の変更を行うことで，MOVE コマンドの操作系列の変更が指示される．

また，システムでは以下の操作が行われる．

1. テキストエディタの MOVE コマンドの系列が変更される．
2. ユーザモデルに現在変更した MOVE コマンドの系列を登録する．
3. 一般化を行う．

   (2) で CUT の操作系列がユーザモデルに登録されているため，ここで '文字列編集' という概念の一般化が行われる．

   2 つの登録された操作系列 (CUT と MOVE) を一般化し，

   文字列編集: コマンド選択 → 対象の選択 → (目的位置の選択) → 実行確認

   という概念が獲得される．
4. 下位概念への反映

   CUT, MOVE の上位概念である '文字列編集' のモデルが獲得された事により，'文字列編集' を上位概念に持つ他のコマンド (ここでは COPY) にこの獲得を反 映させるかどうか，操作系列の変更をユーザに促す．

   ユーザに対して COPY コマンドを変更するかを問い合わせ，もし変更するな ら，COPY コマンドの操作系列を，

   COPY: コマンド (COPY) 選択 → 文字列選択 → カーソル移動 → 実行確認

   に変更し，これをユーザモデルに登録する．
5. '文字列編集' をユーザモデルに登録

   3. で獲得された '文字列編集' の操作系列をユーザモデルに登録するかどうかを 決める．

ユーザモデルは, CUT,MOVE等のコマンドそのもの (そのアプリケーション固有) のモデルを登録しておく部分と, 上位概念 (ここでは '文字列編集') を登録しておく部分とに分かれる.

上位概念は, そのユーザ自身のモデルとして用いられ, 他のツール実行時にも参照されて, そのユーザにあったユーザインタフェースが提供される.

3.3 上位概念のアプリケーションへの反映

前節で獲得されたユーザモデルの上位概念は, 獲得の際に実行されたアプリケーション以外のものにも反映される.

前節で用いたアプリケーションとは別のものを考える. これには, テキストエディタと同様に, 上位概念 '文字列編集' に基づいて作成されたコマンドがあるとする. このアプリケーションが起動されると, まず, そのユーザのユーザモデルの内のそのアプリケーション固有の部分を調べる. その記述が無い場合, 上位概念部分を調べる.

前節でテキストエディタから獲得された上位概念 '文字列編集' がユーザモデルに登録されているとすると, '文字列編集' に基づいて作成されたコマンド群に対して下位概念への反映を実行する. そしてコマンドを変更した場合には, これを新たにユーザモデルのアプリケーション固有部分の方に登録する.

これは, 異なるアプリケーションが同じ上位概念にしたがってコマンドを作成している場合のみ有効である. そのためには, コマンドのための上位概念がいくつか用意されており, アプリケーション作成時にはこれにしたがってコマンドを作成する. その時, どの上位概念を継承して作成したかを明確にしておく (コマンド自体が持っている) 必要がある. 本章の例では, COPY, MOVE, CUT の 3 つのコマンドのみについて考察したので, 上位概念も '文字列編集' 1 つだけであったが, 引数の数や種類が異なるような他のコマンドは, 当然他の上位概念を継承して作成されるものである.

4 ユーザモデル獲得実験

現在, 出力系, 特に木構造データの表示を例にとって, その表示のためのユーザモデルと, そのモデルの獲得方式について検討およびその実験を行なっている. なお, 木構造データはノード (丸) とアーク (矢印) で表し, それを XY の座標系に表示するものとする.

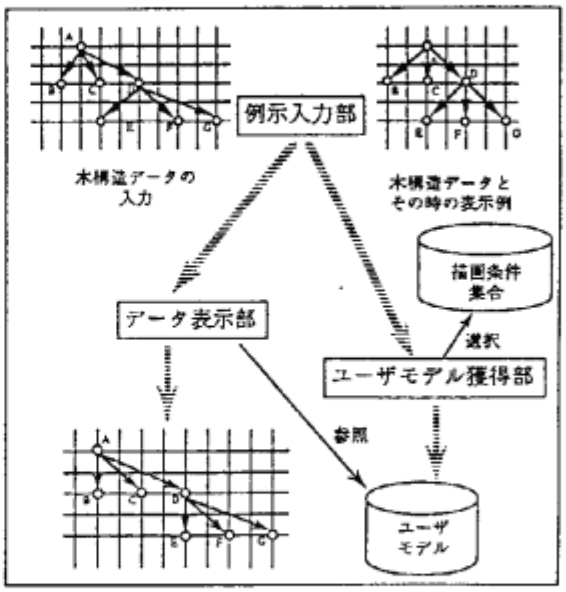

図 2: ユーザモデル獲得実験システム

4.1 システム概要

本実験システムは, 図.2 に示すように, 例示入力部, ユーザモデルの獲得部と, ユーザモデルに基づいたデータの表示部からなる.

例示入力部は, XY の座標軸上にノードとアークを描画することで木構造データを例示入力する部分である. ユーザモデル獲得部は, 与えられた例示からユーザモデルを取り出す部分である. データ表示部は例示入力部より入力された木構造データをユーザモデルにより描画座標を計算し, 描画する部分である.

ここでは, データ表示のための条件をユーザモデルとして利用した. 表示の条件とは, 例えば,

- ノードの座標は必ず格子点 (XY 座標が共に整数)
- あるノードの子ノード (アークの終点ノード) は すべて等間隔で表示する.
- 同じレベル (ルートノードからの距離が同じ) のノードは 同じ X 座標を持つ.

である.

描画条件集合は, ユーザモデルの母集合をなすもので, 例示入力について, この集合の中で条件を満たすもののみを取り出しこれをユーザモデルとする. すなわち, ユーザモデルは描画条件集合の部分集合となる.

4.2 ユーザモデル獲得部

ユーザモデル獲得部では与えられた例から，描画条件集合で満たすものを取り出し，それをユーザモデルとして登録する．このとき，当然いくつもの条件が満たされるような場合がある．例えば，

1. 親ノードのＹ座標と，1番目の子ノードのＹ座標は等しい．
2. 親ノードのＹ座標は，子ノードのＹ座標の平均値に等しい．

という２つの条件が描画条件集合に属しているとする．

与えられた例示が子ノードを高々１つしか持たない場合，上記の２つの条件は両方とも満たされることとなる．しかし，この両方をユーザモデルに登録すると，子ノードが２つ以上ある場合，２つの条件を満たすようにすることができなくなる．

そこで，各条件に優先順位を割り当てる．そして，描画条件集合をいくつかのグループに分け，その中からは条件を高々１つしか取り出さないとしておく．前述のような矛盾する場合を生じる条件は，同じグループに属し，例示を満たすものが同じグループに複数ある場合，優先度に従って制約を取り出すようにする．これによって，ユーザモデル内の矛盾は解消される．

5 考察

実験システムによって，データ表示に関するユーザモデルの獲得，及びユーザモデルを利用したデータ表示が可能であることが確認できた．しかし，本実験によって，いくつかの検討すべき項目も明らかになった．

- 条件の優先順位
  描画条件集合から取り出す条件の優先順位は，現在あらかじめつけておいた順位でおこなっている．しかし，優先順位についても当然ユーザ毎に 異なると考えられるので，ユーザモデルの一部として，この優先順位を定められる方式を検討する必要がある．
- モデルの更新
  本システムでは，ユーザモデルを更新する時にも 例示からの一般化手法を用いている．ただし，この時の 条件の母集合は現在のユーザモデルとなり，その中で 新しい例示が満たす条件のみをユーザモデル内に 残す方式をとっている．
  しかし，優先順位の関係でユーザモデルに登録されなかった 条件がユーザの要求しているものの場合，この条件は 最初の例示でユーザモデルから落とされるので，あとから獲得されることはない．そこで，例示が満たす条件の 集合をユーザモデル以外に持ち，更新の際の母集合を この集合とすることで，上記のような場合にも対応 できるようにする．

6 まとめ

例示からの一般化手法を用いてユーザモデルを獲得し，そのユーザモデルにそってユーザインタフェースをカスタマイズする方式について述べた．また，実験システムでは，表示系についての部分実験を行ない，ユーザモデルが獲得でき，それにそった表示が可能であることが確認された．また，ユーザモデルの枠組での検討すべき課題についても明らかとなった．

今後の研究課題として，前述のモデルの枠組の問題点を解消できるようなモデルの構築がまずあげられる．また，今回の実験では表示系のみに絞って行なったが，この枠組を広げ，入力系についても検討をしていきたいと思う．さらに，実際のアプリケーション実行時にこのプロセスをどのように組み込むべきかということも今後は検討していきたいと思う．

謝辞 本研究は，新世代コンピュータ技術開発機構 (ICOT) の委託研究として進められました．御支援頂きました長谷川 隆三 ICOT 研究部長代理，相場 亮 第４研究室室長代理を初めとする方々に深く感謝いたします．また，研究にあたり，御指導下さったソフトウェア生産技術開発本部川越課長に感謝いたします．

参考文献

[1] Smith ,S.L., Monsier, J.N. , 田村訳：“利用者インタフェース・ソフトウェア設計ガイドライン”, インタフェース・ソフトウェア研究会, 1986

[2] Thompson, T., “The NeXT Step”, BYTE, March 1989, pp. 265-269

[3] 松沢, 鈴木, 町田, 土田, 阪田, “仕様獲得実験システムにおける仕様獲得技法”, 第 39 回情処全大 pp. 1517-1518, 1989.10

[4] 土田, 阪田, 鈴木, 町田, “仕様獲得実験システム”, 情処研報 Vol.89, No.101(89-SE-69), 1989.11

[5] Card, S.K., Moran, T.P., Newell, A., “The keystroke-level model for user performance time with interactive system”, CACM, Vol.23 No.7, 1980