ヒューマンインタフェース 58－6
（1995．1．19）

GUIを持つシステムとユーザ間のダイアログを形式的に記述する試み

松林弘治　辻野嘉宏　都倉信樹
大阪大学基礎工学部

HCIの分野で，インタラクションの様々な側面を形式的にモデル化して評価や設計に援用しようという要求が高まっている．本稿では，コンピュータシステムとそれを利用するユーザとの間でダイアログ（対話）が行われるというインタラクションのモデルを提案する．このモデルでは，システムとユーザはそれぞれ内部状態を持っていると考え，このとき，相手からの入力記号の意味は，入力される前の内部状態から，入力された後の内部状態及び相手への出力記号への写像として与えられる．
本稿では，GUIベースのアプリケーションについて，その具体的な適用例を説明する．

AN APPLICATION OF THE DENOTATIONAL SPECIFICATION OF HUMAN-COMPUTER DIALOGUE TO GUI-BASED SYSTEMS

Kohji MATSUBAYASHI　Yoshihiro TSUJINO　Nobuki TOKURA
Faculty of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560, Japan
matubays@ics.es.osaka-u.ac.jp

In the field of HCI, it has been a growing concern to give a formal model of interaction which is usable for the evaluation and development of computer system. In this paper, we propose the interaction model which specifies the dialogue between computer system and user. Both system and user are defined as have internal entities, and the input symbols to the opponent are strictly defined as a mapping from the current value of entities to those of the next and the output symbols from the opponent.. We also present the application of this specification technique to Macintosh Finder™, a typical GUI-based application.

1　はじめに

ソフトウェア／ハードウェアの進化によってコンピュータシステムが高機能化され，よりユーザの細かい要求に応えるようになってくると，必然的にその両者の間で取り交わされる情報およびその方法はより複雑に，より多義性を帯びたものになってくる．「ユーザ・フレンドリ」と謳われるアプリケーションの開発であればあるほど，そのインタフェース部分の開発に多くの時間が費やされていることからも明らかである．

ユーザとシステムとの間でどのような情報が交換されていて，それらの振舞いや性質はどのようになっているか，すなわちインタラクションの本質を明らかにすることは，ユーザにとってより使いやすいシステムを作るためには必要不可欠であると考えられる．しかしながら，システムの設計や評価における指針となるべきインタラクションのモデルがほとんど存在しない[3]　ために，インタフェース／インタラクション設計の方法論については，思うように進んでいないのが現実である．

この問題の原因としては，人間の認知的行動でまだ不明瞭な部分が多いことも考えられるが，各モデルがそれぞれ別の視点または目的でインタラクション全体および部分を捉え，その結果，各目的に特化した手法が提唱され，インタラクションの様々な側面を統一的に捉えることが困難になっていることも挙げられる．

そこで，本稿では，ユーザとシステムとの間で行われているインタラクションを，対話部分であるダイアログを軸に，数学的に厳密なモデル化を行う手法を提案し，その第一歩として，ユーザの入力記号の厳密な意味記述を行なってみる．[5] では CUI ベースのシステムに対して記述を行ったが，今回はＧＵＩベースのシステムに対して記述を試みる．

2　従来のインタラクションのモデルと形式化について

2.1　既存の形式的手法について

インタラクションをモデル化する方法論およびその実際の設計／評価への適用は，以前から少なからず行われている．最も初期の研究である CLG[7] や GOMS[2]　に始まり，現在までに様々な形式的表記法が提唱されている[4].

インタラクションやインタフェース，対話言語を形式的にモデル化することによって，

- 開発においては，実際にシステムを作る前に，記述上で何らかの評価や予測を行える可能性がある
- 形式的なモデルを元にすることで，定量的な評価尺度を提示しやすい

などといった利点が得られると考えられている．

それらの記述の対象，利用目的は様々に異なるが，代表的なモデルについては表１のようになる．

2.2　これらの問題点

このように，今まで様々な観点からインタラクションを形式化して設計／評価に援用するための試みが行われてきているが，実際には幾つかの重要な問題点を列挙することができる．

(1)　各モデル化手法や記述法は，実際に評価したい対象をそのモデル化及び記述の対象とする．そのため，対象に直接関係ない記述は省かれ，簡潔になる半面，厳密さは失われてしまう．

	モデル化する対象	モデル化の目的
GOMS	ユーザの行動モデル，ユーザの持つ目標	ユーザの行動予測，ユーザビリティ評価
CCT	システム利用に必要なユーザの知識	インタフェースの複雑さの定量的評価
TAL TAG	対話言語（ユーザの入力記号列の文法）	対話言語の一貫性チェック，ユーザの能力モデル生成
CLG	対話の階層的／分解的モデル	インタフェース記述，初期段階での設計補助

表１：代表的なモデル化技法とその特徴

(2) それに伴い，各モデル間の統一をはかるのが困難であり，あくまで異なる側面として表現される．そのため，評価したい側面の数だけモデル／表記法を用意してやる必要がある．

そこで，本稿では，

・インタラクションの様々な側面を表現することが可能であり，
・様々な抽象化のレベルを扱うことができ，
・かつ数学的に充分に厳密な，

インタラクションのモデルおよびその形式的記述法について考えてゆくことにする．

3 インタラクションのモデル

3.1 ダイアログの定義

まず最初に，ユーザとシステムとの間に行き交う情報を記号列であるとみなしてダイアログ（対話）を形式的に捉えることにする．ダイアログDを，ユーザの入力とシステムの出力が交互に行われるものであると考えると，次のように定義することができる．

[定義] ユーザとシステムが与えられたとき、両者の間のダイアログDの集合 $\mathcal{D}$ は

$$\mathcal{D} = \{\beta_0 \alpha_1 \beta_1 \alpha_2 \beta_2 \cdots \alpha_n \beta_n \mid \alpha_i \in \Sigma^*,\ \beta_i \in \Gamma^*,\ \Sigma = \Sigma_s \cup \Sigma_c,\ \Gamma = \Gamma_s \cup \Gamma_c,\ \Sigma \cap \Gamma = \phi\}$$

である。ただし、

Σ_s ：ユーザがシステムに与える記号の集合
Σ_c ：ユーザからシステムに制御が移るきっかけとなるコントロール記号の集合
Γ_s ：システムがユーザに与える記号の集合
Γ_c ：システムからユーザに制御が移るきっかけとなるコントロール記号の集合

とする． □

これは，入力と出力を表面的な並びだけでみたものであるが，ここで表したいのは，様々な抽象化のレベル，階層のレベルでも，交互に記号をやり取りするという対話が行われている，と捉えることができるということである．例えば，ユーザのマウス移動によって，画面上でのマウスカーソルの位置が変化するのも入力と出力の対として考えられるし，あるコマンドの入力と，その結果の出力も同じ様に対として捉えることができる．この場合，それぞれのレベルにおける入出力記号のドメインを決定してやることが必要となる．

3.2 インタラクション全体のモデル化

ダイアログの字面．つまり入出力記号列だけで何らかの評価を行ったりすることも可能であろうが，ここではその意味を厳密に指定できるようにしたいのであり，そのためには，入力／出力によってそのダイアログを行っているユーザとシステムにどのような意味があるか，できるだけ両者の要素を加味して表現してやらなければならない．ここでは，インタラクションの全体像を図１の様に捉えている．

これは，ダイアログにおける各入力記号の意味が，相手からの出力記号と自分自身の内部状態，及びその他の環境などに依存することを表している．

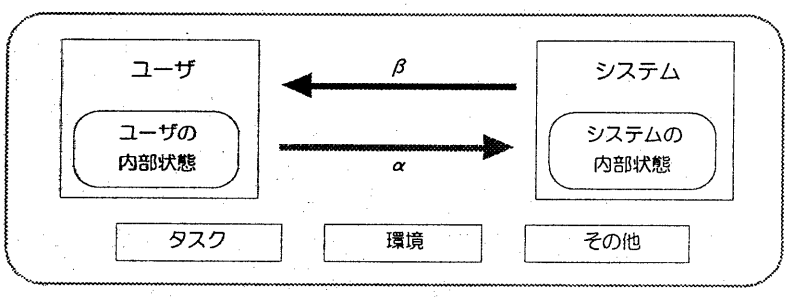

図１：インタラクション全体のモデル

3.3 意味の指定方法

インタラクションに対する意味の指定とは，ここではダイアログで使われる記号に対して意味を指定することを指す．具体的には，ユーザからシステムへの入力記号 α の意味は，α を受け取る前のシステムの内部状態から変化したあとの内部状態への，またユーザへの出力記号 β への写像として定義される．システムからユーザへの出力記号 β の意味も，同様に写像として定義されるものとする．

[定義] ユーザからシステムへの入力記号を α，対応するシステムからの出力記号を β とする．α を受け取る前のシステムの内部状態を E_s，受け取った後の E_s' とすると，**入力記号 α の意味**は，E_s から E_s' および β への写像として与えられる．

同様に，システムからユーザへのの出力記号を β'，対応するシステムへの次の入力記号を α' とする．β' を受け取る前のユーザの内部状態を E_u，受け取った後の E_u' とすると，**入力記号 β' の意味**は，E_u から E_u' および α への写像として与えられる． □

4 実際の適用例：Macintosh Finder™[注1]

4.1 Finder™ の機能の概要

本節では，前節で述べた形式的モデル化の枠組に基づき，ＧＵＩベースの既存のアプリケーションの例として，ファイルブラウジング／操作ソフトウェアである Macintosh Finder™ を取り上げ，そのダイアログの形式的記述について述べる．実際の Finder™ は，各種ファイル操作の他にもシステムのリブート／シャットダウン，ネットワーク上のファイル共有の各種設定など，ログインシェル／システム管理ツールとしての役割も果たしているが，ここでは純粋にファイル操作に関わる操作に限定して記述を行った．本稿では，そのうち幾つかの例について説明していくことにする（詳細は[6]を参照のこと）．

ここで説明する例は，以下のものである．

- マウス移動／クリック
- フォルダのダブルクリック
- "New Folder" コマンド
- "Find..." コマンド

4.2 システムの内部状態の記述

まず，ユーザの入力記号 α の意味を決定するのに必要なものの１つである，システムの内部状態を記述する必要がある．記述上では，領域 (*domain*) を指定することに相当する．

例えば，デスクトップ全体を表すエンティティ Desktop は，次のように，他のエンティティに対応する領域による積領域として表現される．

Desktop : Size × (Disk)* × Trash × (N → Window) × Selection × MenuBar
画面サイズ×ディスク×ゴミ箱×ウィンドウ×選択項目×メニューバー

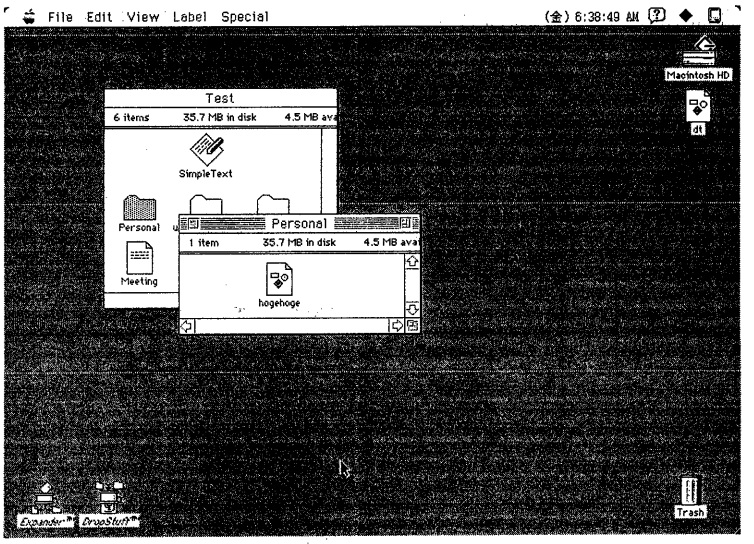

図２：Macintosh Finder™ の画面

注1 Version 7.1.4 (U.S.)

この Desktop を構成する各エンティティも，（通常は）より基本的なエンティティで定義される．この場合であれば，画面サイズを表す Size は，

```
Size      : (N × N)
            (縦サイズ×横サイズ)
```

と表現されるし，ウィンドウを表すエンティティ

```
Window    : Rect × WContent × Path
            サイズ×表示内容×対応するパス
```

は更に，

```
Rect      : (N × N) × (N × N)
            (左上の位置) × (右下の位置)
WContent  : Icon* + Bitmap
            (アイコン) またはビットマップ
Path      : Str
            文字列
```

のように，より詳細に表現することができる．このようにして，システムの内部状態を細部に渡って記述することができる．

4.3 ダイアログの構造の指定

次に，入力と出力の並びの構造を抽象構文として与える．ここで，機能としてのコマンドを入力単位として，ダイアログの並びを考える．

```
Dialoguecom ::=      Initcom Dcom
Dcom ::=  ε | ( NewFolder <NewName> Resp
            | OpenSelection Resp
            | CloseWindow Resp
            | DuplicateSelection Resp
            | FindFile <FindDialog> Resp
            | FindAgain Resp
            | SelectFile Resp
            | SelectAll Resp
            | EmptyTrash <TrashDialog> Resp
            | RenameFile <NewName> Resp
            | MoveSelection <MoveDialog> Resp
            | MoveWindow Resp
            | ResizeWindow Resp ) Dcom
```

ここで，下線を引いてあるのはシステムからの出力記号である．また，<NewName> のようにかぎ括弧で囲まれているものは，1つ下のレベルのダイアログを表し，このレベルでのダイアログの入力記号の一部として機能する．

4.4 意味の具体的な指定：: マウスの移動など

[5] において記述を行ったＣＵＩベースのテキストエディタなどでは，機能としてのコマンドを実行するためにはある特定のキー入力を行うことで実現しているものが多い．そのため，テキストファイルを読み込むコマンド InputFile は，'command' キーを押しながら 'I' キーを押す，という風に物理的な入力シーケンスに直接対応づけることができたが，ＧＵＩのアプリケーションの多くでは，マウスを使って，プルダウンメニューを選択したり，ボタンをクリックしたりしてコマンドを実行することが多い．この場合，マウスの軌跡はいわば自由移動であり，これとコマンドを単純に対応づけるのは困難であると考えられる．

そこで，マウスやキーボードの入力と，機能としてのコマンドとを別のドメインとして定義し，その対応関係を指定することで表記することにする．

マウスやキーボードのレベルでの入力とその出力の並びは次の様に書き表すことができる．

```
DialogueMouseKey ::=   InitMouseKey DMouseKey
DMouseKey ::=   ε | ( MC
            | PressMB Resp MC ReleaseMB Resp
            | DblClick Resp
            | Char Resp | Bs Resp
            | Return Resp | Esc Resp
            | Up Resp | Down Resp
            | Left Resp | Right Resp ) DMouseKey
MC ::=      ε ( MCUp Resp | MCDown Resp
            | MCLeft Resp | MCRight Resp ) MC
```

この入出力の構造を元に，入力記号の意味を与える次の意味関数を考える．*action* は，システムの内部状態から次の内部状態への写像として意味を与え，*echo* は，システムの内部状態及び外部出力から次の外部出力を与えるものである．

```
action : MK → Desktop → Mouse → Clip → FindStr
            → (Desktop × Mouse × Clip × FindStr)

echo : MK
    → Desktop → Mouse → Clip → FindStr → Disp
                                → Disp
```

ここで MK は，マウスやキーボードのレベルでの入力記号列のドメインを表し，Mouse はマウス位置とその形状を，Clip はクリップボード（コピー／ペースト用の一時用領域）を，FindStr は検索に使われる文字列を，それぞれ表すドメインである．Disp は，ディスプレイの表示，すなわちユーザへの外部出力である．

例えば，マウスカーソルの上移動による内部状態の変化は，次の様に記述することができる．

```
action [ MCUp Resp MC ] dt m c f =
    let (mx, my) = m in
        if my = 0 then action [ MC ] dt m c f
        else let m' = ( mx, my - 1 ) in
            action [ MC ] dt m' c f
```

これは，現在のマウス位置のｙ座標が０でなければ，次のマウスのｙ座標の位置が１少なくなり，その他のドメインには変化がないことを表している．

同様にして，外部出力の変化は，次の様に記述することができる．

```
echo [ MCUp ] dt m f c d =
    let (dt', m', f', c') = action [ MCUp ] dt m c f in
        redraw-mouse-cursor d m'
```

4.5 意味の具体的な指定：操作と，機能としてのコマンドとの対応指定

このようにして，マウスやキーの物理的な入力単位の意味を写像として記述することができるが，機能としてのコマンドを入力単位として捉えた場合のダイアログとの関係を示さなければならない．

例えば，フォルダをダブルクリックする（図３）ことにより，フォルダを開く（図４）場合を考えてみる．

図３：フォルダのダブルクリック

この場合，ドメイン MK ではユーザの入力記号は DblClick に相当する．この結果，フォルダが開かれるわけであるから，ドメイン Com でのユーザの入力記号 OpenSelection が実行されたことになる．この場合，入力記号 DblClick の意味は次の様に書き表すことができる（一部）．

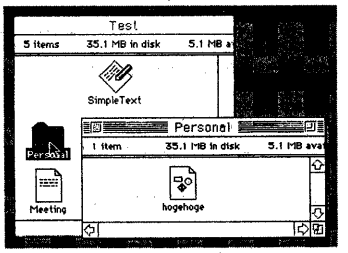

図４：ダブルクリックの結果

```
action [ DblClick ] dt m c f =
    case whereMB dt m in
    onWindow :
        let dt' = front-window (dt, whichWindow dt
m) in
        let i = whichItem dt' m in
        if i = ε then
            select-parent-folder (dt', m, c, f)
        else let dt'' = select-file (dt', i) in
            execute [ OpenSelection ] dt'' m c f
    default :
        ....
    end case
```

これは，「ダブルクリックした場所がウィンドウ内であれば，まずそのウィンドウをフロントにして，マウスカーソルが何の項目の上にもなければ，単にそのウィンドウが選択された状態にする．もし何かの項目の上ならば，その項目を選択してから開く．」ということを表している．

別の例で考えてみる．プルダウンメニューを選択して，カレントディレクトリに新規フォルダを作成するコマンドを選んだとする．この場合，マウスやキーボードのレベルでのダイアログは，DMouseKey における

PressMB Resp MC ReleaseMB Resp

に対応する．このとき，最後の ReleaseMB の時の位置により，どのコマンドが実行されるかが決定

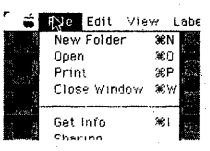

図５：PressMB Resp

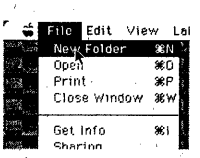

図６：MC

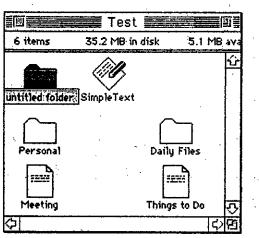

図7：NewFolderに対応するResp

される．このことをふまえて意味を記述すると次の様になる（一部）．

```
action [ PressMB Resp MC ReleaseMB ] dt m cf =
    case whereMB dt m in
    onMenuBar :
        let ( m', com ) =
                selectMenu [ MC ReleaseMB ] d m
        in execute com dt m' c f
    .... (以下略)
```

ここで現れる意味関数 **selectMenu** は，メニューがプルダウンされてから最終的にメニュー項目が選択されるまでの内部状態の変化を表すものである．この例の場合では，マウスボタンを押した後のマウスカーソルの移動 **MB** ReleaseMB の結果，**selectMenu** が対応するコマンド *com* を返してくる．その *com* に従って **execute** が行われる．

このようにして，マウスやキーボードのレベルでのダイアログと，機能としてのコマンドのダイアログとの対応関係を記述することができる．

4.6 意味の具体的な指定：階層的な構造を持ったダイアログの意味指定

プルダウンメニューを使ったコマンド選択では，その引数となるべきものは，続いて表示されるダイアログボックスで指定する場合が多い．例えば，「何々という名前のファイルを検索する」というコマンドは，まず "Find" という項目をメニューから選び，そこで表示されるダイアログボックスに検索するファイル名を入力して，その文字列に基づき検索が実行される．このとき，コマンドの構造

FindFile <FindString> Resp

に対応する意味記述は，次のようになる．

```
execute [ FindFile <FindString> ] dt
m cf =
    let ( m', c', f' ) = eval [ <FindString> ] m cf in
        if μ [ <FindString> ] = no then ( dt, m', c', f' )
        else ( find-file (dt, f), m', c', f' )
```

このとき，<FindString> は，ダイアログボックスが表示されてから，ユーザが "Find" ボタン "Cancel" ボタンを押してダイアログボックスが画面上から消えるまでの一連の入出力列を表す．

```
<FindString> ::= FindDialog FindStr
                 (OK | Cancel) DismissDialog
FindStr ::= ε | ( Char Resp | CutStr Resp
    | CopyStr Resp | PasteStr Resp
    | SelectStr Resp | MoveWindow Resp
    ) FindStr
```

このレベルでは，ダイアログボックス上に対する操作しか定義されていない．ここで行われた対話の結果，すなわち，この場合では検索対象となる文字列と，検索をする／しないの情報が，機能としてのコマンドのレベルへと渡されると捉えているわけである．

```
eval [ FindDialog FindStr OK ] m cf = eval [ FindStr ] m cf
eval [ FindDialog FindStr Cancel ] m cf =
                                    eval [ FindStr ] m cf
eval [ TypeStr FindStr ] m cf =
    eval [ FindStr ] eval [ TypeStr ]
..... (以下略)
```

このようにして，階層的な構造をもったダイアログの意味を記述することができる．

5 特徴，および問題点

今回提唱したモデル化，記述方法の特徴としては以下のものが考えられる．

(1) ダイアログを階層的に捉えて記述できるので，各レベル毎に理解することができる
(2) それに伴い，細部まで厳密に記述することが可能である

また，問題点としては以下のものが挙げられる．

(1) 考え得る全ての状態をエンティティとして写

像の引数にとらなければならないので記述が繁雑になる恐れがある.

(2) 新しい機能／状態を考えたときに，記述全体に渡って書き直さなければならない可能性がある.

(3) 記述内容は，それを記述する人に依存するため，ある既存のシステムを評価しようとする際に，評価尺度によっては，同じシステムであるのに記述が異なるために評価結果が異なる可能性がある.

（1）と（2）に関しては，表示的意味論の表記法を使ったところからくる問題点である．しかしながら，これはあくまで記述上の問題であり，表示的な意味の指定法そのものに問題があるわけではないと考えられる．実際，プログラム言語の意味論においても，より "Practical" な意味記述のためにAction Semantics などが考えられている [8].

（3）に関しては，形式的な記述法には必ず現れる問題であるが，この記法を実際にどのように評価に適用するか，どのような側面を表現可能か，という問題と密接に関わっているために，今後の課題となる.

6 おわりに

本稿では，ユーザとシステムとの間で行われるダイアログを，内部状態をもった2主体間で行われる記号のやり取りと定義し，その全体をインタラクションとして捉えるというモデルを提唱した．また，そのモデルに基づき，やり取りされる各記号の表示的な意味を数学的に厳密に記述する手法を紹介した．その例として，ＣＵＩに比べてより複雑なインタラクションが行われているであろうと思われるＧＵＩベースのアプリケーションについて，その入力記号の意味と，インタラクションの特徴を含んだ記述を行った [6]. この記述法は，インタラクション全体を統一的に記述するための第一歩であり，様々な階層や抽象レベルを踏まえて記述することができた．また，細部にわたって厳密な記述を行うことができた.

システムからの出力記号の意味は，ユーザの内部状態と次のシステムへの入力記号に依って決まるが，その写像を決定するためにはもちろん適切なユーザモデルを慎重に考え，導入する必要がある.

また，実際の設計に援用するためには，プログラミング言語におけるセマンティクスと同様，実用的なものにするためにより記述を分かりやすくしたり，可搬性に優れたものにする必要がある．その場合には，実際に行う記述は読むものにとってより平易な別の表記法で行い，その表記法と数学的に厳密な表記との間での写像を定義することが考えられる.

参考文献

[1] Apple Computer Inc.: *Macintosh Human Interface Guidelines*, Addison-Wesley, NJ. (1989).

[2] Card, S.K., Moran, T.P. and Newell, A. : *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale NJ. (1983)

[3] チグネル，M.H.，ハンコック，P.A.，ローエンサル，A.：*"知的インタフェース - 人とマシンの知的相互作用 - "*，辻 敏夫 訳，海文堂．(1991)

[4] Kieras, D.E. & Polson, P.: "An approach to the formal analysis of user complexity", *International Journal of Man-Machine Studies*, 22, pp. 365-394. (1985)

[5] 松林 弘治，辻野 嘉宏，都倉 信樹：*"ＨＣＩにおけるダイアログを階層的に捉えた形式的記述の試み"*，*情報処理学会研究報告*，HI-53. (1994)

[6] Matsubayashi, K., Tsujino, Y. & Tokura N. : A *Denotational Specification of Human-Computer Interaction and Its Application to Macintosh Finder™*, 大阪大学基礎工学部情報工学科テクニカルレポート（登録予定）.

[7] Moran, T.P.: "The Command Language Grammer: A representation for the user interface of interactive cimputer systems", *International Journal of Man-Machine Studies*, 15, pp.3-50. (1981)

[8] Mosses, P. D. : Denotational Semantics, in *Handbook of Theoretical Computer Science, Vol.B*, ed. van Leeuwen, J., North-Holland, Amsterdum (1990)

[9] Payne, S.J. & Green, T.R.G. : Task Action Grammers, *Human Computer Interaction*, 2, pp.93-133. (1981)

[10] Reisner, P. : Formal Grammers and Human Factors design of an interactive graphics system, in *IEEE Transactions on Software Engineering*, 5, pp.229-240. (1981)