ヒューマンインタフェース 35－7
（1991. 3. 7）

手書きヒューマンインタフェースの構築支援環境

荒井俊史　正嶋博　福永泰
（株）日立製作所　日立研究所

手書きヒューマンインタフェース（手書きＨＩ）とは、計算機をペン一本で操作するための技術である。筆者らは、ワークステーション上で手書きＨＩを実現する環境を開発している。手書きＨＩのための表示／入力装置として、高精細液晶と透明タブレットを重ねた入力一体化ディスプレイを試作した。また、手書きＨＩを支えるソフトウェアとして、手書き文字を認識するストローク認識サーバ、画面設計や動作指定を支援するユーザインタフェース管理システムを開発した。これらにより、手書きＨＩを開発する手間を大幅に削減できる。

DEVELOPMENT ENVIRONMENT FOR A HANDWRITTEN HUMAN INTERFACE

Toshifumi Arai, Hiroshi Shojima, Yasushi Fukunaga
Hitachi Research Laboratory, Hitachi Ltd.
4026 Kuji-cho, Hitachi-shi, Ibaraki-ken 319-12 Japan

The Handwritten Human Interface is a technology which allows its user to operate a computer system using a "pen" device. We have developed the hardware and software environments to support the building Handwritten Human Interface. The hardware environment is based on a high resolution LCD, and a transparent tablet device set on the LCD. The software environment provides stroke recognition facilities based on a client-server model, and a UIMS (User Interface Management System) supporting the development of the human interface software.

1. はじめに

日常の業務では、案内文書に地図を付けたり、文書にコメントをつけたりするなど、イメージ情報が自然に活用されている。すなわち、コード情報しか扱えないと人間にとっての効率が悪いという処理も存在する。筆者らは、計算機で日常業務を支援するには、イメージ情報を容易に扱えるＨＩが必要と考え、ペンだけで計算機を操作できる手書きＨＩのための基盤技術を開発している。

ハードウェアとしては、高精細液晶ディスプレイに透明タブレットを重ねた装置を試作した。この表示と入力の機能を兼ね備えた装置を、「入力一体化ディスプレイ」と呼んでいる。入力一体化ディスプレイでは、テキストやイメージなどが表示される位置と、手（ペン）で指示する位置を一致させられるため、紙に字や絵を書く感覚で計算機を操作できる。例えば、文書編集システムで校正記号を編集コマンドに使うような応用も可能である。手書き文字認識によるテキスト入力も、もちろん可能である。

手書きＨＩを支えるソフトウェアとしては、手書きストロークのオンライン認識機能と、グラフィカルユーザインタフェース（ＧＵＩ）の構築支援機能が重要である。前者は、手書き文字などをその場で認識してコード情報に変換する技術で、従来のキーボードと仮名漢字変換機能の代りとなる。この認識技術は従来から研究されており、現時点でも実用に耐えるレベルに達している［1，2］。ＧＵＩ構築支援環境としては、ユーザインタフェース（ＵＩ）管理システム（ＵＩＭＳ）と呼ばれるものが、広く研究／実用化されている［3-7］。

本稿では、筆者らが開発したソフトウェアのうち、アプリケーションプログラム（ＡＰ）へのストローク認識機能の組み込み方、手書きＨＩ実現のために必要だったＵＩＭＳへの機能拡張ついて報告する。

2. 手書きＨＩ構築時の問題点

2.1 文字入力時の応答性の確保

利用者のストローク入力に対するエコーの応答性は極めて重要である。ペン先の移動に対して、ストローク表示が遅れると、利用者に大きな違和感を与えてしまう。エコーの応答性の確保は、特に文字を連続して入力している時に重要である。図１は利用者が「日本」の「日」を書き終り、「本」の第１画目の横棒を書き始めたところである。横棒の書き始めの時点で文字枠が移ったこと、すなわち前の文字（「日」）が書き終ったことが分かるので、システムは前の文字の認識を開始する。筆者らのシステムでは、１文字当りの認識時間は200～500ミリ秒であるが、この間にエコーが遅れるだけでも利用者に大きな違和感を与えてしまう。したがって、システムはストローク認識中であっても、利用者のストローク入力に対して十分な応答性を確保しなければならない。

上記の課題を満たすため、筆者らのシステムでは、ストローク認識モジュールをＡＰとは独立に実行されるサーバプロセスとして実現した。これについては第３章で詳しく説明する。

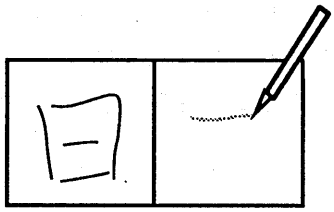

図１．ストローク認識中の応答性の確保

2.2 テキスト挿入ポイントの整合性

図２は、利用者が文字「本」を書き終った直後にテキスト挿入位置を動かした状況を示している。このような操作は、前に書いた文字に認識誤りを発見した場合によく現れる。この場合、ペンがテキスト表示フィールドに触れた時点で、今まで書いていた文字（「本」）が書き終ったものと見なせるので、システムはその文字の認識を開始する。

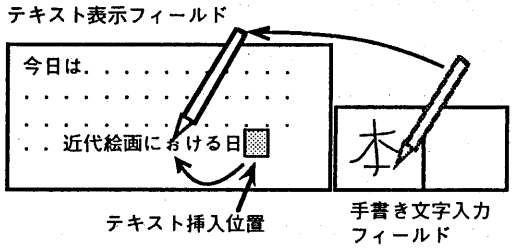

図２．文字認識中のテキスト挿入ポイントの移動

ここで問題となるのは、認識し終った文字を何処に挿入するかである。認識が終るのは200～500ミリ秒後であり、その時点では挿入位置は移動されているので、単純に現カーソル位置に挿入すると利用者

が意図していた結果と食い違ってしまう。認識結果を挿入すべき位置は、その文字を書き始めた時点でのテキスト挿入位置である。

上記のような処理を、ＡＰを作る毎に考えるのは効率が悪い。そこで筆者らは、手書きＨＩの構築基盤の一部として、上記処理を支援する機能を組み込むことにした。具体的には、筆者らが開発してきたＭＵという名のＵＩＭＳ［5 - 7］を、手書きＨＩ実現のために機能拡張し、ＭＵ／Ｓと呼ばれるシステムを開発した。どのような機能を追加したかは、第４章で詳しく説明する。

3. ストローク認識機能の組み込み方

3.1 クライアントサーバモデル

第２章で述べたように、筆者らのシステムではストローク認識機能は、サーバ形式のプロセスで実現されている。筆者らはそれをSserverと名付けた。Sserverは、クライアントであるＡＰに対してストローク認識機能と辞書管理機能を提供する。ストローク認識が必要な応用プログラムは、Slibというライブラリを介してSserverとの接続を確立する。Slibは、Sserverとのネットワーク透過な接続を提供するＣ言語から使用できるライブラリである。

ソフトウェア開発者は、Slibを介してSserverが提供する次の６つの機能を利用し、手書きＨＩを活用したＡＰを作成できる。

1. 認識に用いる辞書の追加／削除
2. パターンの登録／削除
3. ストロークデータの正規化*
4. 正規化後のデータをSserverに送る
5. 認識結果をSserverから受け取る
6. Sserverの状態を調べる

*補足説明：

デバイスからの入力座標列は、座標系の向きや尺度がデバイスやウィンドウシステムに依存しており、ストロークの大きさ（文字枠の大きさなど）は応用プログラムの仕様に依存している。そのようなデータは、Sserverに渡す前に決められた形式に変換しなければならない。この変換をするのがストロークデータの正規化である。

3.2 クライアントサーバモデルの利点

クライアントサーバモデルは、ネットワークで接続された計算機間で、負荷を分散したり資源を共有できるなどの利点を持つ。このため、システムの応答性を向上させたり管理の手間を低減するために有効である。さらに、手書きＨＩの構築支援環境に適用した場合には、ＡＰの制御構造を単純にできるという利点もある。以下では、ＡＰの制御構造を単純にできることについて説明する。

第２章で述べたように、システムがストローク認識中でもペン入力に対して即座にエコーを返さねばならない。このため、認識プログラムをＡＰの一部として実現すると、エコーの応答性を確保するために定期的な入力検査や割込処理が必要となる。しかし、定期的な入力検査や割込処理を使うと、ＡＰの構造が複雑になってしまうという問題がある。

一方、クライアントサーバモデルを用いると、認識処理が、入力とエコーを処理する応用プログラムとは独立に動作可能となるため、応答性確保のための特別な工夫は不要となる。以下でその例を説明する。

次のＣ言語風の疑似コードは、クライアントサーバモデルを用いた場合の、典型的なＡＰの制御構造を示している。ＡＰは、初期化終了後にタブレットデバイスとSserverとの通信用のストリームを開く。そしてメインループの中で、それらのストリームを多重入力待ちする。手続き 多重入力待ち は、複数のストリームからの入力を監視し、入力があったストリームの識別子を返すものとする。ＡＰは、利用者の操作（ペン入力）があった場合はその利用者イベントを処理し、Sserverから通信があった場合は送られてきた認識結果を表示する。

このように、クライアントサーバモデルを用いることで、ＡＰの制御を単純で理解し易いループ構造とすることができる。

```
main()
{
    int fd, tfd, sfd;

    ＡＰの初期化;
    tfd = タブレットデバイスを開く;
    sfd = Sserverとの通信を開く;
    while (TRUE) {
        fd = 多重入力待ち(tfd, sfd);
        if (fd == tfd) {
```

```
            利用者イベント処理(tfd);
        } else if (fd == sfd) {
            認識結果表示(sfd);
        }
    }
    後処理;
}
```

4. 手書きＨＩ対応ＵＩＭＳ

この章では、筆者らが開発したＭＵ／Ｓについて説明する。ＭＵ／Ｓは、筆者らが手書きＨＩとは独立に開発してきたＭＵという名のＵＩＭＳに、手書きＨＩ実現に必要な機能を追加したものである。まずＭＵの概要を述べ、次にＭＵ／Ｓへの機能拡張について述べる。

4.1 ＭＵの概要

ＭＵではＵＩを定義するためのＵＩＤＬという言語を提供しており、この言語での記述はシステムに組み込まれているインタプリタによって解釈／実行される。インタプリタであるため、ＵＩ変更後のＡＰの再リンクは不要であり、少しずつ動かしながらＵＩの定義を追加して行くこともできる。

以下で、例題をもとにＭＵの概要説明を試みる。次例は、ＵＩＤＬによる典型的なＵＩ構成要素（ＵＩ部品）の記述である。

```
object SampleButton (PushButton)
{
    base = 'MainWindow
    pos_x = 100
    pos_y = 100
    title = "Push Here"
    event PenDown() {
        HighLight()
        call AppFunction()
        UnHighLight()
    }
}
```

この記述からは、SampleButtonというプッシュボタンが生成される。PushButtonsは、ＵＩ部品を生成する際の雛型であり、プロトタイプと呼ばれる。プッシュボタンの他に、メニュー（Menu）、アイコン（Icon）、テキスト表示領域（TextField）などが用意されている。ボタンが表示される土台はbaseという属性で指定され、この例ではMainWindowが指定されている。また、表示位置はpos_xとpos_yという属性で指定され(100, 100)、表示される文字列はtitleで指定され"Push Here"である。

プッシュボタンがペンで指示された時には、そのボタンに対してPenDownというイベントが送られる。この例では、PenDownを受信した時の動作は、強調表示し（HighLight）、AppFunctionというＡＰ中の手続きを呼び出し（call）、強調表示をやめる（UnHighLight）ように指定されている。それぞれの動作は、アクションと呼ばれている。PenDown以外のイベントとして、ペンを離した時に発生するPenUp、キー入力時に発生するCharInput、一定時間他のイベントが発生しなかったときに発生するTimeoutなどがある。

PenDownのように座標に関係があるイベントは、その座標を含むＵＩ部品に送られる。一方、KeyInputやTimeoutのように座標とは無関係に発生するイベントの送り先は、イベントフォーカスという考え方で決められる。例えば次のような記述がなされているＵＩ部品は、ペンで指示された時にキー入力を受け付けるようになる。

```
event PenDown() {
    Get_event_focus('CharInput)
}
event CharInput() {
    /* do something */
}
```

また、プログラムでイベントを発生できるように、Send_eventというアクションが用意されている。発生させるイベントの種類は、イベント名で指定する。

4.2 ＭＵ／Ｓで拡張された機能

ＭＵ／Ｓでは、ストローク処理を定義し易くするため、次の４つのイベントを使用する。

(1) Timeout
タイムアウトが発生したことを示すイベント。

(2) Leave
イベントを受け取るＵＩ部品が切り替わるときに発生するイベント。あるイベントが、これまでイベントを受け取っていたＵＩ部品とは別のＵＩ部品に送られるべきものである時、これまでイベントを受け取っていたＵＩ部品に対しLeaveが送られる。

上記２種類のイベントは、手書き文字を一文字分書き終ったことを判定するために使われる。したがって、これらのイベントを受信するＵＩ部品は、ほとんどの場合ストローク入力フィールドである。イベントを受け取った時のストローク入力フィールドの動作は、ストロークの座標列を正規化し、Sserverに送信することである。

(3) NewChar

このイベントは、ＵＩ部品がSend_eventを使って発生させる。ほとんどの場合、このイベントを発生させるのはストローク入力フィールドであり、受信するのはテキスト表示フィールドである。ストローク入力フィールドは、手書き文字入力が開始されたことを示すためにNewCharを発生させる。テキスト表示フィールドは、NewCharを受信することで後にSserverイベントが発生することを知り、Sserverイベントのイベントフォーカスを手に入れると共に、その時点でのテキスト挿入位置を記憶する。

(4) Sserver

このイベントは、ストローク認識サーバから認識結果が届いた時点で発生する。ほとんどの場合、受信するのはテキスト表示フィールドである。テキスト表示フィールドは、認識結果を読み出し、NewChar受信時に記憶しておいたテキスト挿入位置に挿入する。

次節で、以上のイベントを使ったストローク処理の具体的な定義例を示す。ＭＵ／Ｓのストローク処理では、各ＵＩ部品の記述が他のＵＩ部品の記述と独立にできるため、ストローク処理に関するＵＩ部品を容易に追加／削除できるという利点がある。さらに、ストローク処理に関する定義を、それぞれのプロトタイプにあらかじめ組み込んでおけるので、ＡＰの開発者は単にＵＩ部品を生成して表示に関する属性を設定するだけでよいという利点もある。

4.3 ストローク処理の定義例

次のリストは、２つの手書き文字入力領域と、１つのテキスト表示領域のためのＵＩＤＬによる記述である（ただし、簡単のために各ＵＩ部品の初期化と表示に関する記述は省略してある）。

文字入力領域

```
object CharDrawingArea1 (StrokeField)
{
    event PenDown {
        if (Buffer_empty()) {
            Send_event('NewChar)
            Get_event_focus('Leave)
            Get_event_focus('Timeout)
        }
        Echo_and_store_stroke()
    }
    event Leave {
        Normalize_and_send_data()
        Clear_buffer()
    }
    event Timeout {
        Normalize_and_send_data()
        Clear_buffer()
    }
}

object CharDrawingArea2 (StrokeField)
{
    /* completely same as
                CharDrawingArea1 */
}
```

テキスト表示領域

```
object TextDisplayArea (TextField)
{
    event PenDown {
        Cursor_on()
        Get_event_focus('NewChar)
    }
    event NewChar {
        Remember_current_position()
        Get_event_focus('Sserver)
    }
    event Sserver {
        var s:string
        s = Get_result_string()
        Inset_string(s)
    }
}
```

StrokeFieldは手書きストロークを入力するためのＵＩ部品を生成するためのプロトタイプである。ストロークのエコーを表示し座標列をバッファに記録するアクション（Echo_and_store_stroke）、表示されたストロークのエコーとバッファ中のデータを消去するアクション（Clear_buffer）、バッファ中の座標

列を正規化しSserverに送るアクション（Normalize_and_send_data）、バッファが空かどうかを確かめるアクション（Buffer_empty）が組み込まれている。

TextFieldはテキストを表示／編集するＵＩ部品を生成するためのプロトタイプである。カーソルを表示するアクション（Cursor_on）、現在の挿入位置を記憶するアクション（Remember_current_position）、文字列を記憶されている挿入位置に挿入するアクション（Insert_string）が組み込まれている。Get_result_stringは認識結果を取り出すアクションで、全てのＵＩ部品が実行できる。

5. システム構成と適用例

入力一体化ディスプレイの外観を写真１に示す。液晶の解像度は1120×780ドットであり、１ミリメートルあたり約５本の線密度を持つ。タブレットからの座標データは、9600bpsのRS-232C回線を介して、ワークステーション本体に入力される。タブレットの座標検出精度は、１ミリメートルあたり約１０本である。

図３に、手書きＨＩの応用例であるメールシステムの実行画面を示す。右下の２つの矩形は手書き文字を入力するための領域であり、ここに書かれた手書き文字は、認識された後に左側のテキスト入力／表示領域のテキスト挿入位置に挿入される。この図のように、メール本文に、地図などのイメージ情報を重ねることもできる。

このＨＩは本稿で述べたシステムを用いて構築されており、新規開発したプログラム量は、メール配送系とのつなぎにＣ言語で870行、画面設計と動作の定義にＵＩＤＬで650行であった。

6. まとめ

手書きＨＩを開発する際の問題点は、ストローク認識中にエコーの応答性が低下すること、ストローク認識中のユーザ操作の許容に伴い文字挿入位置がずれることの２点である。それらの解決策として、クライアントサーバモデルに基づくストローク認識モジュール（Sserver、Slib）、テキスト処理機能とイベント処理機能を強化したＵＩＭＳ（MU／S）を開発した。以上により、手書きＨＩを活用したＡＰの開発を大幅に効率化できるものと考えている。

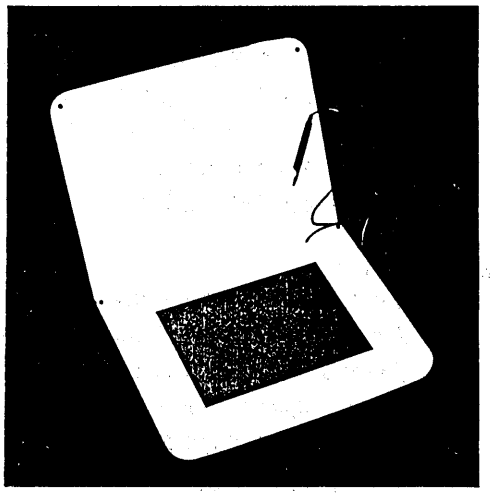

写真１．入力一体化ディスプレイ

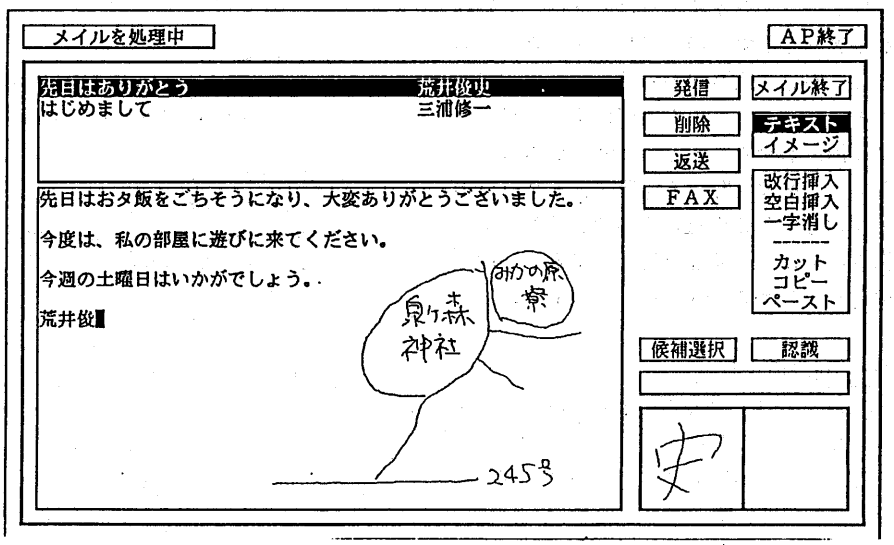

図３．手書きＨＩの例（メールシステムの画面）

参考文献

[1] 横田他、ひらがな文字詳細識別アルゴリズムの検討と評価、信学技報 PRU88-82、pp57-64（1988）

[2] 正嶋他、手書きマンマシンインタフェースの開発、信学技報 OS88-48、pp.51-56（1989）

[3] PFAFF, G.E., ed.、User Interface Management System、Springer-Verlag（1985）

[4] 暦本他、エディタを部品としたユーザインタフェース構築基盤：鼎、情報処理（創立30周年記念論文特集） Vol.31, No.5, pp.602-611（1990）

[5] 谷他、メタユーザインタフェースを有するユーザインタフェース構築支援システム、情処論文誌 Vol.23, No.3, pp137-146（1987）

[6] 谷他、手書きスケッチによるユーザインタフェースのプロトタイピング、情処研報Vol.90, No.3（1990）

[7] 荒井他、メタレベルユーザインタフェースを構築支援できるユーザインタフェース管理システム、情処研報Vol.90, No.3（1990）