携帯機器向けGUI環境「式神」の実装

大谷 浩司
株式会社 アックス

1 はじめに

我々は、携帯機器向けの GUI 環境「式神」を開発している。[1] 式神は、ハンドヘルド PC や PDA で Linux を利用するユーザに、GUI 環境を提供するもので、以下の特徴を持つ。

1. 操作の手間を省く簡潔な「デスクトップ」
2. 手書き文字認識システム「布目」
3. 知的情報処理を補助する「人工知能機能」
4. GNOME API 互換

プロジェクトでは、7月24日現在、アルファ2・バージョンの公開を行ない、図1と図2に示すように COMPAQ iPAQ と NEC Mobile Gear II で動作が確認されている。

図 1: iPAQ 上の式神

実現予定の機能や特徴については、「Linux Conference 2000 Fall」[1] にて、発表しており、これらの詳細については、そちらを参照して欲しい。PDF データが、式神のホームページ [2] から入手可能である。

図 2: Mobile Gear II 上の式神

本論文では、現状と、実装の詳細について述べる。

以下では、まず、全体のプログラムの概略を述べる。次に、「デスクトップ」と「布目」の実装について述べ、「布目」については、認識のアルゴリズムとその評価も行なう。

2 概略

式神は、ハンドヘルド PC と PDA では、画面構成が異なる。図3と図4にハンドヘルド PC 向けと PDA 向けの画面例を示す。

ハンドヘルド PC 向けでは、画面の主な部分をメインとサブの二つの領域に分け、各領域を満たすようにアプリケーションを表示する。パネルを右横に表示し、メニュー・バー、ツール・バーは、アクティブなアプリケーションのもののみを画面上部に表示する。二つの領域で動作中のアプリケーションの表示は、必要に応じて瞬時に交換できる。また、広い画面を利用したい

図 3: ハンドヘルド向け画面

図 4: PDA 向け画面

場合は、簡単にひとつの画面にすることも可能である。

PDA 向けでは、画面を縦長に使う。アプリケーションの表示される領域は、原則ひとつであるが、必要ならば二つの領域を使うこともできる。パネルを画面上部に配置し、メニュー・バー、ツール・バーは、下部に配置する。メニュー・バーについては、画面の幅に収めるために、階層を増やして、常時表示しているのは「メニュー」項目だけにする。

プログラムの構成は、ハンドヘルド PC 向けと PDA 向けともに同じである。図 5 にプログラム構成を示す。

ウィンドウの基本部分には X Window サーバを利用しており、その上で式神ウィンドウ・マネージャが画面を支配する。その上には、メニューの表示やプログラムの起動を行なうパネル、ファイルマネージャ、人工知能がユーザを補助する。布目は、アプリケーションに文字入力を提供する。

構成は、GNOME を元にしているが、セッション・マネージャは利用していない。パネルは、GNOME のパネルを変更して使っている。ファイル・マネージャは、GMC(GNOME Midnight commander) をやはり

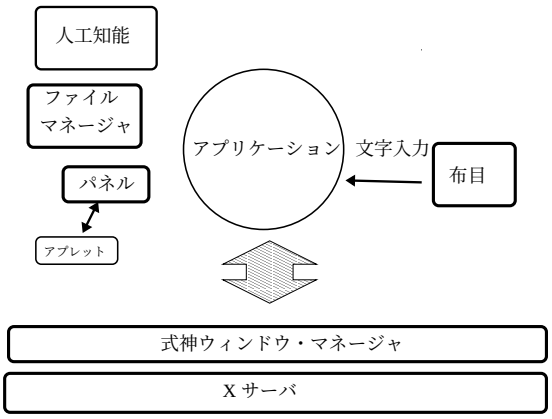

図 5: プログラム構成

変更して使っている。

3 デスクトップ・システム

デスクトップ・システムのプログラムのライブラリ構成は、ハンドヘルド PC と PDA 向けで同じで、図 6 の様になっている。

<table>
<tr><td rowspan="2">式神ウィンドウ
マネージャ</td><td>GNOME
アプリ</td><td>パネル</td><td>ファイル
マネージャ</td></tr>
<tr><td colspan="3">GNOME libs/式神</td></tr>
<tr><td colspan="4">GTK+/式神</td></tr>
<tr><td colspan="4">Xlib</td></tr>
</table>

図 6: デスクトップのライブラリ構成

GNOME lib は、式神用に改造している。GTK+に関しては、改造を予定しているが、現在は変更していない。以下で、各プログラム/ライブラリについて述べる。

3.1 式神ウィンドウ・マネージャ

X Window では、ウィンドウ・マネージャがアプリケーションのウィンドウの表示場所、大きさ等を制御する。従って、式神においてアプリケーションのウィンドウを小さな画面に如何に配置するかは、ウィンド

ウ・マネージャの役割となる。ここでは、今回新規に作成した「式神」ウィンドウ・マネージャについて、詳細に説明する。

3.1.1 GNOME 互換

「式神」は、GNOME 互換を目指している。そのため、式神ウィンドウ・マネージャも、GNOME 互換ウィンドウ・マネージャ[3] となっている。「式神」におけるメインとサブの二つの画面領域は、GNOME のワークスペースに対応している。すなわち、「式神」では、二つのワークスペースが標準で存在し、メインとサブという名前がついているのである。メインがワークスペース 0、サブは、ワークスペース 1 である。

GNOME 互換であるので、GNOME のタスク・リスト・アプレットなどが、ほとんど変更せずに利用可能である。

3.1.2 ウィンドウ・マネージメント

X Window では、ウィンドウはサーバが管理する矩形領域で、ボタン等にも利用される。しかし、ウィンドウ・マネージャの管理の対象となるのは、アプリケーションがルート・ウィンドウにマップしようとするウィンドウだけである。

そのようなウィンドウは、ウィンドウ・マネージャからみて、以下の 3 種類に分類される。

1. トップレベル・ウィンドウ

   アプリケーションの主な表示を行なうウィンドウである。アプリケーションにひとつの場合が多いが、ウェブ・ブラウザなどでは、複数の場合もある。

2. 一時的ウィンドウ

   ダイアログなど一時的に表示されるウィンドウである。プロパティとして WM_TRANSIENT_FOR を持つ。

3. 臨時ウィンドウ

   プロパティに関わらず属性 overide_redirect が True になっているウィンドウで、メニューなどに使われる。

このうち、臨時ウィンドウはウィンドウ・マネージャが制御することはできない。従って、管理の対象になるのはトップレベル・ウィンドウと、一時的ウィンドウである。これらのウィンドウは、アプリケーションを表しているために、タスクと呼ばれることがある。

ウィンドウのマップ

式神ウィンドウ・マネージャは、一つのワークスペースについて、一つのウィンドウしかマップしない。あるウィンドウがマップされた時、以前マップされていたウィンドウはアンマップされる。そこで、以下の点が問題となる。

1. 新規にマップ要求のあったウィンドウをどのワークスペースにマップするか？
2. ウィンドウ・マネージャがアンマップしたウィンドウをどのように管理するか？
3. アンマップ要求によってウィンドウをアンマップしたとき、空いたワークスペースにどのウィンドウをマップするか、あるいはしないか?

これらのための方式を決定するには、ユーザの要求を考える必要がある。我々内部で議論した際には、以下のような意見があった。

i 電卓や時計のような補助的なアプリケーションは、一般にはサブ・ワークスペースに表示して欲しい。

ii しかし、時には、メインに表示して欲しい場合もある。

iii 作業中のアプリケーションのウィンドウは、なるべく隠さないで欲しい。

iv ダイアログを閉じたら直前に表示していたウィンドウを表示して欲しい。

v 一旦非表示にされたウィンドウが、再び表示される場合は、以前と同じワークスペースに、表示して欲しい。

vi しかし、それでは、ユーザは、以前のワークスペースを覚えているわけではないので、意図したウィンドウがどこに表示されるか予想しにくい。

vii なるべくたくさんのウィンドウを表示して欲しい。

これらの意見は、簡単には実現不可能と思えるもの、互いに矛盾しているものが存在する。そこで、我々は、以下のような方式を採用した。

- i,ii の意見は、ユーザが意図しているものを意図した場所に出すことを要求している。しかし、ユーザが今、どのような意図であるかを推測するのは、困難である。そのため、トップレベル・ウィンドウを新規にワークスペースにマップする場合は、ユーザに尋ねることにした。
- 一時的ウィンドウについては、関連するトップレベル・ウィンドウとは異なるワークスペースに表示する。これにより、作業中のトップレベル・ウィンドウをなるべく隠さないようにして、意見 iii に沿うようにした。
- iv の意見から、ウィンドウ・マネージャがアンマップしたウィンドウは、スタックに管理することとした。アンマップ要求によりウィンドウをアンマップした場合は、後にはスタックのトップのウィンドウをマップする。これにより、ダイアログを閉じた場合には、直前に表示していたウィンドウを表示する。また、ウィンドウが二つ以上ある場合には、必ず両方のワークスペースに表示されていることになるので vii の意見に従っている。
- マップ要求の際にウィンドウに_WIN_WORKSPACE プロパティが存在する場合は、その値が示すワークスペースにマップする。これは、GNOME 互換のために必要となる。これにより、タスクリスト・アプレットによってマップする場合は、以前のワークスペースにマップされる。従って、意見 v に部分的に沿っている。この動作は、意見 vi にあるように、少々分かりにくいかも知れない。

図 7 に、新規作成ウィンドウをマップしようとした場合を示す。

1. アプリケーションがウィンドウ E を新規に作成し、マップ要求を出すと、ウィンドウ・マネージャは、どちらのワークスペースにマップするかユーザに問い合わせる。

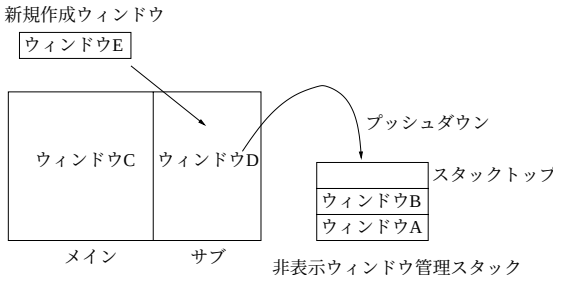

図 7: 新規作成ウィンドウのマップ

2. ユーザがサブ・ワークスペースを指定すると、そこに既に表示していたウィンドウ D は、アンマップされて、非表示ウィンドウ管理スタックのスタックトップに入れられる。
3. その後に、ウィンドウ E がマップされる。

図 8 は、サブ・ワークスペースに表示しているウィンドウ E を削除した場合を示す。

1. アプリケーションがウィンドウ E を削除しようとして、アンマップする。
2. ウィンドウ・マネージャは、非表示ウィンドウ管理スタックのトップにあるウィンドウ D をとりだして、サブ・ワークスペースにマップする。

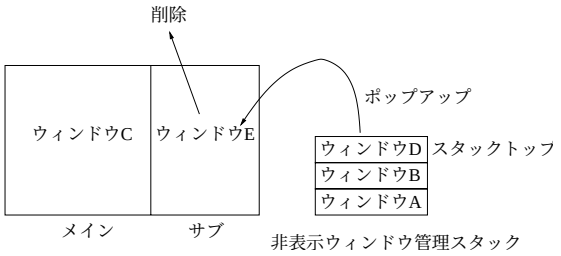

図 8: 表示ウィンドウの削除

ウィンドウのリサイズ

ウィンドウ・マネージャが扱わなければいけない、もうひとつの問題はウィンドウのサイズの問題である。アプリケーションが、画面のサイズに合わせて最適化されていれば、問題は少ない。しかし、既存の多くのアプリケーションは、小さな画面に表示すると画面か

ら、はみ出して非常に使いにくくなる。また、小さな画面に合わせて設計されていても、ワークスペースのサイズの変更が起こるとやはり、はみ出てしまう。そこで、式神では、以下のようにウィンドウの大きさを制御する。

式神ウィンドウ・マネージャは、小さな画面を有効に使うために、アプリケーションのウィンドウはワークスペースの表示サイズ一杯にリサイズする。しかし、アプリケーションが最大サイズ、最小サイズをプロパティで指定している場合には、それに従う。最小サイズがワークスペースの表示サイズよりも大きい場合は、スクロール・バーが現れる。スクロール・バーにより、ワークスペースをスクロールして、隠れている部分を表示することができる。

これにより、画面の大きさに合わせて最適化されているアプリケーションは、最適に表示される一方、既存の多くのアプリケーションを利用することが可能になる。また、新たにアプリケーションを作成する場合でも、画面の大きさをあまり気にしない簡易な方法をとることもできる。

3.1.3 メニュー・バー、ツール・バーの配置

先に述べたように、式神ウィンドウ・マネージャは、メニュー・バー、ツール・バーを、アプリケーションから切り離し、上部あるいは下部に配置する。この実装は、libgnomeui の改造をともない少々トリッキーである。実装法を、以下に簡単に述べる。

1. GNOME のメニュー・バー、ツール・バーは元々アプリケーションのトップレベル・ウィンドウから切り離すことができる。
2. そこで、libgnomeui において、式神ウィンドウ・マネージャが動作していると判断した場合には、メニュー・バー、ツール・バーをトップレベル・ウィンドウから切り離す。ただし、マップしないでおく。
3. トップレベル・ウィンドウのプロパティにメニュー・バー、ツール・バーのウィンドウ ID を設定しておく。
4. 式神ウィンドウ・マネージャでは、ワークスペースがアクティブになった時に、トップレベル・ウィンドウのプロパティからメニュー・バー、ツール・バーのウィンドウ ID を知り、指定の場所に配置してマップする。

3.1.4 布目連係

文字入力に、手書き文字認識を利用する場合、手書き文字を書く領域と制御用のウィンドウが必要となる。問題は、このウィンドウをどのように画面に配置するかである。任意の位置に表示した場合は、アプリケーションの必要な箇所を隠してしまう可能性が高く、頻繁に移動する必要が出てくる。そこで、式神では、以下のように布目のウィンドウを表示する。

1. 布目のウィンドウは、作業中のウィンドウが表示されていないワークスペースすなわちアクティブでないワークスペースにマップする。ワークスペースがひとつしか有効になっていない場合でも特別に、ふたつ目を有効にしてからマップする。
2. 布目のウィンドウを表示しているワークスペースは、布目のウィンドウを表示するのに必要十分なだけの大きさに変更する。したがって、布目のウィンドウはスクロールなしで操作できる。
3. 布目のウィンドウが閉じられた場合には、ワークスペースの状態を元に戻す。
4. 布目のウィンドウが表示されている間は、そのワークスペースには、決して他のウィンドウが表示されることはない。また、無効になることもない。

アプリケーションの表示領域は、布目のウィンドウの出現により狭くなる。しかし、式神のワークスペースの機能により、リサイズされるか、あるいはスクロール・バーが出現する。そのため、アプリケーションの全エリアを操作することができる。

式神ウィンドウマネージャが布目のウィンドウを認識する方法は、ウィンドウのプロパティである。ウィンドウにプロパティ_AX_SG_NUNOME がついていれば、そのウィンドウを布目のウィンドウだと認識する。

3.1.5 デスクトップ連係

これまで、GUI 全体のシステムを表す言葉として、デスクトップを用いてきたが、この項では、狭義のデ

スクトップとの連係について述べる。

ファイルやデバイス、ランチャなどのアイコンを置ける、いわゆるデスクトップを式神は提供する。このデスクトップは、簡単に表示できる方が便利である。そこで、式神ではデスクトップのウィンドウを認識し、ボタン等でデスクトップ・ウィンドウをアクティブ・ワークスペースに呼び出すことができるようにしている。

式神ウィンドウ・マネージャがデスクトップのウィンドウを認識する方法は、ウィンドウのプロパティである。ウィンドウにプロパティ_AX_SG_DESKTOP がついていれば、そのウィンドウをデスクトップのウィンドウだと認識する。

3.1.6 パネル連係

パネルが表示されている場合は、パネルの部分を避けてワークスペースを配置する必要がある。そのため、式神ウィンドウ・マネージャは、パネルのウィンドウを認識し、それを除いた領域にワークスペースを配置する。

式神ウィンドウ・マネージャがパネルのウィンドウを認識する方法は、布目やデスクトップの場合と同様、ウィンドウのプロパティである。ウィンドウにプロパティ_AX_SG_PANEL がついていれば、そのウィンドウをデスクトップのウィンドウだと認識する。

3.1.7 マウスボタン・シミュレーション

既存の GNOME アプリケーションなどの X Window のアプリケーションでは、マウスの中ボタン、右ボタンを利用するものがある。しかし、ハンドヘルド PC や PDA では、左ボタンのみがサポートされている。そこで、式神ウィンドウ・マネージャは、中ボタンや右ボタンをシミュレーションする機能を持っている。これは、以下の方法で実現されている。

- あるキー (あるいはボタン) が押された時に、XSetPointerMapping を利用して、左ボタンを中ボタンや右ボタンにマッピングする。
- キーが離されると、マッピングを元に戻す。

式神が利用している GNOME パネル、GMC において右ボタンが利用されている。これらでは、右ボタンは、カスタマイズなどのメニューを表示するために使われており、頻繁には利用しないために不便さは感じない。

3.1.8 クライアント・メッセージ

式神ウィンドウ・マネージャの以下の機能は、X のクライアント・メッセージを利用して、外部プロセスから制御できる。

1. ワークスペース間のタスクの入れ換え
2. ワークスペースの無効/有効の切替え
3. アクティブ・ワークスペースの切替え
4. ワークスペースの大きさの変更
5. デスクトップの呼び出し
6. 指定のワークスペースへのウィンドウのマップ

3.2 パネル

パネルは、以下の改造を行なっている。

1. 式神ウィンドウ・マネージャとの連係のためのプロパティ操作
2. デフォルトのメニューの変更
3. デフォルトの属性の変更

   表示位置、大きさ、HIDE ボタン、アイコン・イメージの縮小方式などを変更している。

3.3 GMC

式神では、ファイル・ブラウザ、デスクトップ・マネージャとして GMC を利用している。しかし、そのままでは、デスクトップのウィンドウはルート・ウィンドウとなってしまう。そのため、以下の改造を行なっている。

1. デスクトップ用のウィンドウを新たに作成する。
2. このウィンドウをルート・ウィンドウの代わりに使用する。
3. ウィンドウには、式神ウィンドウ・マネージャ連係用のプロパティを設定する。

4 布目

4.1 布目の概要

布目は、漢字を含めた手書き文字を認識して、文字コードにしてアプリケーションに渡すシステムである。キーボードを持たない PDA 等で、漢字を入力するには必須のシステムといえる。入力画面例を、図 9 に示す。

図 9: 布目の画面例

布目は、手書き文字認識の他に、仮想キーボード、JIS コード入力を備え、文字入力全般の機能を提供する。

4.2 プログラム構成

布目のプログラム構成を図 10 に示す。

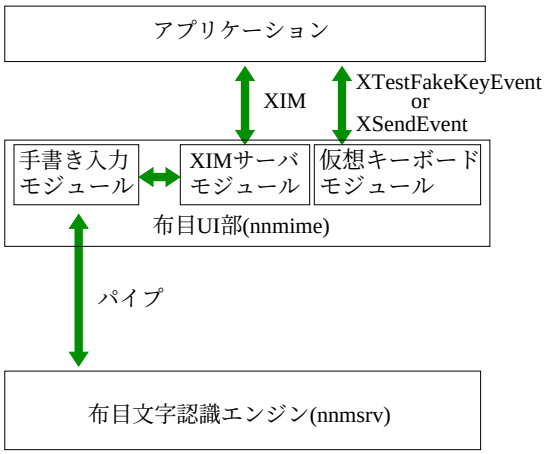

図 10: 布目のプログラム構成

布目は、ユーザインタフェース部の nnmime と認識エンジン nnmsrv の二つのプロセスで構成される。nnmime と nnmsrv の間は、パイプを利用して通信をする。

nnmime は、以下の三つのモジュールから構成される。

1. 手書き入力モジュール

   手書き文字入力のインタフェース部を担当する。

2. XIM モジュール

   XIM プロトコルのサーバ側を実装するモジュール。

3. 仮想キーボード・モジュール

   仮想キーボードを実現しているモジュール。

手書き入力モジュールは、入力された手書き文字データを、認識エンジンに送る。認識エンジンは、データを認識して文字コードを手書き入力モジュールに返す。ユーザが入力を確定すると、手書き入力モジュールは、確定した文字コードを XIM モジュールを利用して XIM を使ってアプリケーションに送る。

仮想キーボードの場合は、文字コードは、XTestFakeKeyEvent または XSendEvent を利用してアプリケーションに送る。

4.3 二つのプロセス

先に述べたように、布目は二つのプロセスで動作する。ひとつの認識エンジンを複数の UI 部から利用するという事は行なっておらず、一見不合理に思える。しかし、以下のような理由によりこのような方式を採用した。

布目の認識処理は、少なからぬ処理量があり、数十ミリ秒から数百ミリ秒を要する。そのため、認識処理の終了を UI 部が待つようにすると次のような問題が生じた。

- 連続して文字を入力すると、次の手書き文字の最初の方のストロークをただちに表示しない。そのため、ユーザは非常にぎこちない感覚を覚える。また、処理を遅く感じる。マシンによっては、ストロークを失うこともある。

これを解決するためには、認識処理と UI の処理を並行に処理する必要がある。並行処理をおこなう方式は、以下の 3 種類が考えられる。

(1) プログラミング・テクニックのみによる方法

認識処理を十分小さな複数の処理に分割し、各処理毎に UI 部に制御を戻す。UI 部は、処理すべきものがない場合は、認識処理を続行させる。

(2) マルチスレッド処理

認識処理と UI 処理を異なるスレッドで動作させる。

(3) マルチプロセス処理

認識処理と UI 処理を異なるプロセスで動作させる。

方式 (1) は、プログラムが複雑になる上、非効率的になる。リソースの利用効率を考えると (2) が良さそうである。しかし、Linux では、マルチスレッドは全ての CPU においてうまく動作するわけではない。従って、(3) のマルチプロセス処理を行なうことにした。

4.4 コード送出

布目のような文字入力プログラムでは、認識結果の文字や仮想キーボードの文字のコードを、どのようにしてアプリケーションに送るかが問題となる。

式神は X Window の上に実現しているので、X Window において最も一般的な XIM を利用するのは、当然であろう。そうすれば、既存の多くの XIM 対応プログラムに入力することが可能になる。しかし、XIM では、一部のコントロール・コードを送ることができない。そのため、仮想キーボードでは、XIM を利用できないのである。XIM を利用しないで既存の X Window のプログラムにコードを送る方法は、以下の二つ存在する。

(1) XSendEvent を利用する方法

XSendEvent 関数を利用して KeyPress イベントをアプリケーションのウィンドウに明示的に送る。キー・フォーカスがどのウィンドウにあるかを送信側が認識しないければならない。

(2) XTestFakeKeyEvent を利用する方法

X サーバの XTest 拡張に存在する XTestFakeKeyEvent 関数を利用して、サーバにキー・イベントを送る。後は、サーバが、キー・フォーカスのあるウィンドウに KeyPress イベントを送ってくれる。

これらは、マルチバイト・コードを送ることができないが、キーボードに存在するコードを送るのには十分である。一方、それぞれ次のような欠点がある。

(1) XSendEvent を利用する方法

- 受信側のイベント構造体には、send_event フラグが立つ。アプリケーションによっては、send_event フラグの立ったイベントを受け付けない場合がある。例えば、kterm などでは、allowSendEvent 属性を true にしなければ、受け付けない。
- ウィンドウ・マネージャによっては、キー・フォーカスのあるウィンドウを知ることができない。

(2) XTestFakeKeyEvent を利用する方法

- XTest 拡張がすべてのサーバに備わっているわけではない。
- XTest 拡張は、暫定的なものとされており、なくなるかも知れない。

そこで、布目では、認識結果の文字コードを送るのには XIM を利用し、仮想キーボードのコードを送るのには、以下の基準で XSendEvent あるいは XTestFakeKeyEvent を利用する。

1. 布目のコンパイル時に XTestFakeKeyEvent を利用しないようになっていれば、XSendEvent を使う。
2. 現在のサーバに XTest 拡張が存在すれば、XTestFakeKeyEvent を使う。
3. そうでなければ、XSendEvent を使う。

4.5 認識アルゴリズム

布目では、入力された手書きデータから特徴情報を抽出し、辞書中の標準特徴情報と比較し最も近いデータの示す文字種別を認識結果とする。認識時間を短縮するために、図 11 に示すように、比較は 2 段階で行なう。

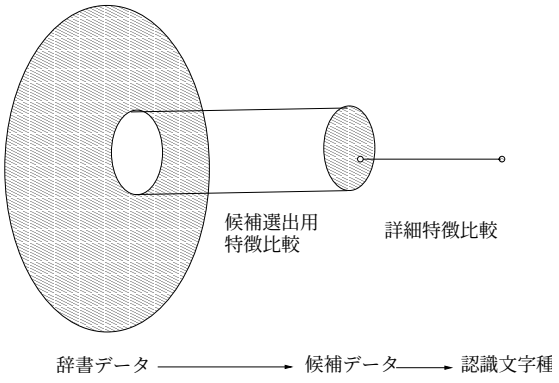

図 11: 2 段階文字認識

第 1 段階で、候補選出用の簡単な特徴を利用して、辞書中の全データから 50 個の文字種に候補を絞る。第 2 段階では、この候補だけを対象にして詳細な特徴比較を行ない最も近い文字種を認識結果とする。

4.5.1 候補選出用特徴

第 1 段階の認識で利用する特徴は、以下のものである。

1. 全ストローク数
2. 縦横の大きさ
3. 領域毎の各方向の長いストロークの数
4. 領域毎の各方向の短いストロークの数

方向は、図 12 に示す 8 方向に分類している。

4.5.2 詳細比較用特徴

第 2 段階の認識では、文字を 16 × 16 の領域に分割し、各領域に各方向のストロークがどの程度の長さ含まれているかを数値化したものを文字データの特徴として利用している。

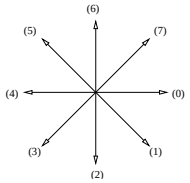

図 12: 方向の分類

例えば、図 13 を見て欲しい。ただし、説明の簡明さのために、分割数は 4X4 にしてある。ここでは、「谷」という字が書かれている。上から 2 行目の左から 3 番目の領域を抜き出したものが、図 14 である。この領域では、左上から右下への斜めのストロークが、ほぼ全域を横切っている。領域の幅が 4 であるとすると、この領域の特徴を表す数値は、方向 (1) が 4 で、他の方向は 0 となる。

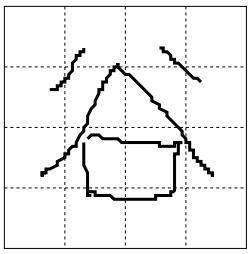

図 13: 詳細比較用特徴 (a)

図 14: 詳細比較用特徴 (b)

ただし、16x16 のままでは、データ量が大きくなり過ぎるのと、特徴が鋭敏になり過ぎるため、後処理により重ね合わせおよびぼかし処理を行ない 7 × 7 の領域のデータに変換する。まず、16x16 の領域の上に図 15 に示すように、7x7 の領域を設定する。新しい領域の特徴量は、元の領域の値を加えたものとする。このとき、図 16 に示すように、周辺部の値を含める。図 16 の色の濃さは、値への寄与の大きさを示している。

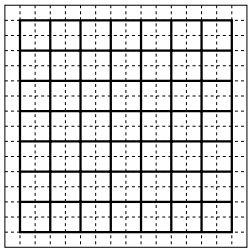

図 15: 重ね合わせ／ぼかし処理 (a)

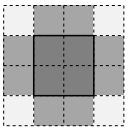

図 16: 重ね合わせ／ぼかし処理 (b)

4.5.3 特徴抽出処理

手書きデータから特徴を抽出する処理は図 17 のように行なう。

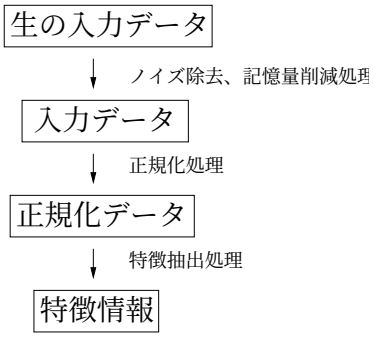

図 17: 特徴の抽出

まず、生の入力データからノイズ除去、平滑化を行なった後、近接している点の除去、再サンプリングを行い記憶量の削減を行なう。次に正規化を行なう。正規化では、入力データを全て同じ大きさまで拡大する。このとき、多くのストロークが存在する部分をより大きく拡大する非線形正規化を行なっている。図 18 に例を示す。左側の枠が元のデータで文字「整」を書いている。右が正規化後のデータである。単に大きくなっているだけでなく、より複雑な上部の方がより大きくなっていることに注意して欲しい。

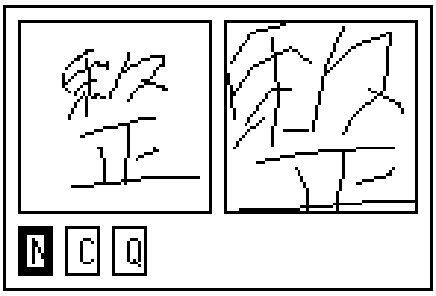

図 18: 非線形正規化

4.5.4 辞書

辞書は、文字種の特徴情報を集めたものである。各文字種のデータは、複数のユーザの手書きデータの特徴を平均したものである。

4.6 評価

認識性能の評価のために認識率を導入する。以下で使用する認識率とは、次の通りである。

- 認識率

  ひらがな、カタカナ、アルファベット、第 1 水準漢字を認識させて、意図した文字種が出た数の全体の文字数に対する割合である。

以下では、あるユーザ A のデータを利用しないで作成した辞書を用いて、A の手書きデータを認識させた結果を用いている。複数のユーザの認識率の平均をとるのが良いと思われるが、データが少ないために一人のユーザのものを用いている。結果として、このユーザの癖を反映したものとなっていると思われるが、認識率向上の目安としては、役立つと思われる。

まず、認識に用いた処理の効果を見るために、図 19 に幾つかの条件下での認識率を示す。辞書は、6 人分のデータから作成されたものを用いている。

図 19 から、1 段階目の認識だけに比べ、2 段階目の詳細な認識処理を行なう事により、30%以上の大幅な認識率の向上が実現できていることがわかる。

また、1 段目の処理による候補の絞り込みより、認識率の低下が生じることが心配されたが、表を見ると 1 段目の処理を行なわない場合の方が、約 5%認識率が低下している。これは、第 1 段処理で落されるべき文

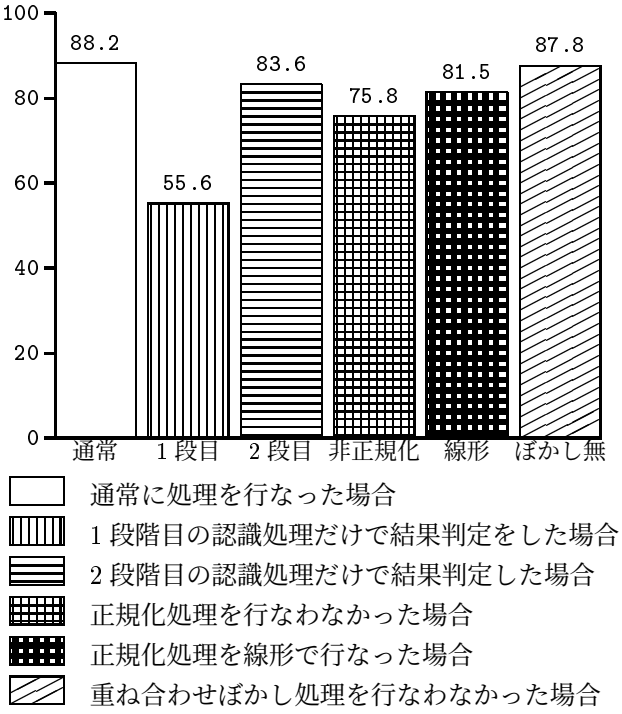

図 19: 幾つかの条件下の認識率

字種が残ってしまっていることにより、誤った認識を行なうことがあるためだと思われる。

正規化処理についても、行なったことにより約12%向上していることが分かる。また、正規化を非線形にしたことにより、約 7%の向上が見られる。

特徴量の重ね合わせ/ぼかし処理は、向上は 0.4%とわずかなものに留まっている。今後、見直しが必要となるかも知れない。

処理時間については、表 1 に示す通り、第 1 段階の処理を行なうことにより、大幅に短縮されている。

条件	1 文字の認識に要した時間 (msec)
第 1 段階認識を行なう	60
第 1 段階認識を行なわない	5900

表 1: 処理時間

次に、辞書にデータを利用したユーザの人数と認識率の関係を見てみる。図 20 に、辞書にデータを利用した人数と、あるユーザの認識率のグラフを示す。

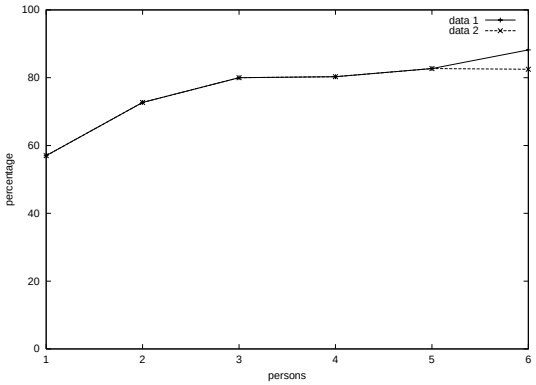

図 20: 辞書への利用人数と認識率

data 1 と data 2 では、利用したユーザのうち 6 人目が異なる。

図 20 をみると、一般に辞書にデータを利用した人数が多いほど、認識率が高くなっている。しかし、data 2 では、6 人目でわずかに低下しており、必ずしも、単純に向上するとは限らないことが分かる。従って、認識率を向上させるためには、多くのデータから適切なデータを選択して、辞書を作成することが必要となるといえる。

4.7 課題

布目を利用してみると、文字によっては丁寧に書かないと認識してくれない。そのひとつの原因として、考えられるのは以下の点である。

- 布目では、文字の書き順は、見ていない。そのため、ユーザの自由度が高くなっている。しかし、ストロークを 8 方向に分類している。そのため、左から右に書いた場合と右から左に書いた場合では、認識に大きな影響を及ぼす。これにより、ユーザの自由度が減少している。例えば、図 21 を見て欲しい。「口」の下辺は、(A) に示すように正しく、左から右に書かないといけない。(B) のように、右の縦棒部分から続けて右から左に書くと認識されないのである。

これに対する解決方法として、単純にストロークの分類を 4 方向に減らすことが考えられるが、単純に減

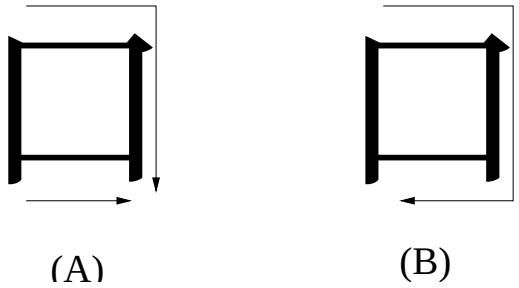

図 21: ストロークの方向

少させた場合は、約 5%の認識率低下が生じた。他のパラメータやアルゴリズムと合わせて変更する必要があると思われる。あるいは、辞書に両方のパターンを登録するのが良いかもしれない。今後の検討課題である。

また、本論文を記述している段階では、個人の癖の学習機能を持たない。この機能を加えることにより、ユーザの自由度はさらに増し、認識率も向上すると考えられるので、合わせて考える必要がある。

5 おわりに

本稿では、式神の現状と実装について述べた。デスクトップは、GNOME を元にライブラリやプログラムの改造と、新規プログラムの作成により実現していた。布目については、その認識アルゴリズムについても述べ、評価を行なった。2 段階認識、正規化など布目の行なっている処理は、認識率の向上や認識時間の短縮に効果があることが分かった。また、辞書のデータの改良が認識率の向上に結び付くことも判明した。

今後は、以下の事項について、作業をしていきたいと考えている。

1. デスクトップのメモリ消費量の削減
2. 布目の辞書の整備
3. 布目の認識アルゴリズムの改良
4. 布目の認識アルゴリズムのパラメータの調整
5. ユーザインタフェースの改良

6 謝辞

布目の設計開発を行なった故 橋本 圭介氏に感謝する。現在の布目の認識アルゴリズム等主な部分は、すべて橋本氏によるものである。また、共同開発者の近藤 政雄氏、福居 宏和氏、竹岡 尚三氏、横川 龍雄氏に感謝する。その他、多くの方々にコーディング、手書き文字データの入力／整理を手伝っていただいた。感謝したい。最後に、GNOME プロジェクトを始め、オープン・ソース・ソフトウェアに関わっている方々に感謝する。

参考文献

[1] 大谷 浩司, 福居 宏和, 橋本 圭介, 渦原 茂, 竹岡 尚三: 携帯機器向けのオープンな GUI 環境「式神」, Proceedings of Linux Conference 2000 Fall pp.403-410

[2] 式神プロジェクト・ホーム・ページ, http://www.sikigami.com/

[3] GNOME Window Manager Compliance - How to write a GNOME compliant Window Manager, http://developer.gnome.org/doc/standards/wm/book1.html