ウィンドウマネージャによる
アクセシビリティの改善について

平成 25 年 2 月 15 日

情報電子工学科
工藤 覚

目 次

1 はじめに **1**

2 アクセシビリティ **1**

2.1 支援技術 1

3 X ウィンドウシステム **5**

3.1 概要 5

3.2 X Window の設計構築 5

4 ウィンドウマネージャ **7**

4.1 各種ウィンドウマネージャ 7

4.2 各種ウィンドウマネージャの比較 8

5 Fvwm2 カスタマイズ **9**

5.1 デスクトップ上のフォーカス 10

6 デスクトップ上の位置を対象としたフォーカス **12**

6.1 改修に必要なリソース設定 12

6.2 改善した結果 1 17

6.3 他に考えられる位置の対象 18

6.4 改善した結果 2 20

6.5 考察 21

7 まとめ **21**

参考文献 **23**

概要

パソコンをマウスやキーボードを使って操作することは, 普通の人には問題はないが, 特に高齢者や目, 腕などの障害を持つ人にとって使いにくいことが多い。それを解決するには, 利用しやすさ, アクセシビリティを改善することである。

X ウィンドウシステムに含まれるグラフィカル・ユーザ・インターフェースは多数のウィンドウマネージャが備えており, 各種類ごとにデスクトップ環境の外観, マウス・キーボードなどの操作が異なっている。その環境で障害者がパソコンを使用する場合, 操作性と使いやすさを考慮しなければならない。

本研究ではウィンドウマネージャFvwm2 のデスクトップ環境を選び, キーボードナビゲーションのフォーカス操作にウィンドウの位置対象として左上, 左下, 右上, 右下の条件を 追加する改良を行なった。改良前と改良後を比較するとウィンドウをフォーカスする巡回が少なくなり, ボタンの押す回数を減らすことができた。

しかし, 問題点として, 位置対象を四つのキーの組合せで押すため覚えるのに時間がかかることと, パソコンに慣れた熟練者だけ扱える欠点があったため, 位置対象を上, 下の二つに少なくした初心者が扱いやすい改良も行なった。

工藤 覚

1　はじめに

人がパソコンを使用してマウスやキーボードを操作する際, マウスから手を離してキーボードを打ったり, 再びマウスで動かすのはデスクトップ (画面) 上で必要不可欠な事だが, 身体に障害がある人はその操作をするのに不慣れな環境になってしまう。

その対策として高齢者や障害を持つ人にとって, パソコン操作し易い環境はどうあるべきか, どう改善するか考えなければならない。例えば, 手や腕の障害でマウスを使えない場合, ソフトウェアはキーボードだけの使用に限定し, 弱視や老眼の人にとってはデスクトップ (画面) の拡大や配色を用意に行なう操作や目がまったく見えない人は読み上げソフトを使うので, それに適したレイアウトや記述方法が必要と挙げられる。

これらの問題を改善するには X ウィンドウシステムを利用し, アクセシビリティ内のプログラムをカスタマイズすることが考えられる。普通カスタマイズするのに OS が MS-Windows の場合, 容易に実行することはできない。だが,OS が UNIX の場合, X ウィンドウシステムに含まれている GUI(Grafical User Interface) 環境で独自な操作性を構築できる。

元々,GUI は文字やアイコンなどの絵ができ, マウスでの操作は対象物 (文字, アイコン) がそれぞれわかりやすくコンピュータに不慣れな初心者でも扱いやすくアクセシビリティの改善が図れます。

2　アクセシビリティ

2.1　支援技術

この研究室で使われている Solaris のデスクトップは障害を持つ人がそのデスクトップ上のパソコン操作を支援するソフトウェアが備わっており機能をカスタマイズすることができる。支援技術に使用されるソフトウェアはスクリーンリーダー, 拡大鏡など障害者がコンピュータを操作しやすくするためインストールされている。これら支援するソフトウェアはどの機能があるか特徴と障害に適している対象と主なソフトウェアをまとめた。

- スクリーンリーダー

  コンピュータ上で動作する合成音声の読み上げとテキスト出力を読むための点字ディスプレイがそれぞれ利用される。ユーザは音声を聞いたり出力された点字を読むことでコンピュータに表示されている画面の情報を知ることができる。主に視力が全くない人が使用され,web サイトをマウスとキーボードの変わりにアクセスするのに使われる。

主に目が全く見えない全盲の障害者が利用している。

- winVoice
  かな分かち書き・六点漢字・漢点字・情報処理点字など, 各種点字に表示されるようそれぞれ対応している。また, タッチカーソルが備えられており, カーソル位置表示とカーソルの追跡ができるなど, 点字ディスプレイを存分に生かせる機能になっている。
- PC-Talker XP
  パソコンに内蔵されている PCM 音源は人の肉声に近い各種アプリケーションでも使われ, その音声で聞き慣れた音声をメインにして使い分けること, ガイド音声も好みの設定に変更でき, さらに視覚障害者がキーボードだけで点字入力 (六つのキー) もできる。
- JAws
  欧米のスクリーンリーダーとして使われている。特徴はスクリプト (簡易プログラム) を記述することで複数のアプリケーションに対応できる。しかし初心者には難しく価格が高価なため利用している人は少ない。
- CatWalk
  他のスクリーンリーダーと異なり, 読めない部分を読みやすくする目的で開発されている。また, 単体で使うより PC-Talker XP と winVoice など併用することで大きな効果が期待される。しかし, 初心者には操作が少し難しく, 単体で使う場合ドキュメントメーカーとプロトメーカーなどの音声エンジンを用意しなければならない。

- 音声認識

  マウスやキーボードを動かす操作の必要性を軽減し, 人の音声でコンピュータ操作ができる。その人の音声で声を出してドキュメントや電子メールを入力したり音声コマンドを使用したプログラムの起動・切り替え,OS の制御と Web フォーム等の入力も実行できる。

  主に腕や手の障害でマウスを動かせない人に適している。

  - Amivoice
    PC にソフトウェアをインストールすれば, すぐに音声入力が可能で音声認識の向上を実現した初心者でも扱いやすいソフトウェアである。同音異義語を検出・修正でき, 利用回数が多いほど音声認識精度を上昇させる学習機能を持っている。また, 登録・修正された単語を自動で学習し, 人の肉声による認識をより高くすることができる。
  - Julius

フリーの高性能音声認識ソフトウェアで, 数万語の語彙を対象とした文章発声認識を持っている。また, 高音声認識率は二万語彙の読み上げ音声は 90 パーセント以上で文章を記述する文法でも認識され, 文のパターンを人手で記述した認識用文用法を使用することで小語彙の音声対話システムと音声コマンド入力ができる。

- ドラゴンスピーチ

  議事録と報告書などの文章を作成するのに時間がかかってしまう。そこでドラゴンスピーチを使用することでそれらの文章を音声だけ作成でき, キーボード入力でかかった時間より短縮できる。その他, 録音することもでき, そのデータを自動的に文字化する機能を持っている。また, この機能は病院で訪問診療の移動時間中に音声で問診結果を録音し, その録音したデータを音声認識で文字化することで電子カルテへの効率的な入力ができるようになる。

- 拡大鏡

  人がデスクトップ (画面) を見やすくするため指定された画面の一部を別のウィンドウに拡大されたものが表示される。また, マウスカーソルに合わせた拡大表示および全画面表示が実装されている。主に弱視や老眼で画面が見えづらい障害の人が利用するのに適し, 実際に拡大する位置を指定するにはマウスカーソル・キーボード入力, またはテキスト編集で決めることができる。

  - xzoom

    表示の一部を拡大でき, スクロール操作で画面が切り替わる場合, 連続して拡大画面が追従し UNIX 上のコマンド入力に -mag x y というオプションを同時入力することで拡大率の指定ができ, キーボードで x と y 方向の拡大率を増やしたり減らすこともできる。

  - xlupe

    xzoom と同様に画面の一部を拡大表示でき, 拡大された画面はマウスカーソルを動かした位置に追従する。xzoom と違いキーボードによる操作は無く三つのボタン (stop はカーソルの追従を停止・再起動,stats はコマンド入力内のウィンドウにデバッグ表示, quit はアプリケーションの終了) をマウスクリックすることで実行される。

  - lupe

    xlupe と同じマウスカーソル追従で, どの位置に画面が拡大されているかがわかる。しかし,xzoom と xlupe は長方形のウィンドウで拡大表示されているが,lupe は円の形で拡大表示されているため全体的に小さい。

- キーボードナビゲーション

パネルやウィンドウ上で使用するユーザが, キーボードからデスクトップ (画面) をナビゲート (パネルとウィンドウ等の操作する指示を与える) でき, パネルとウィンドウの操作をナビゲートするにはあらかじめ設定されたキーボードショートカット (キーボードの組合せを同時に入力すること) で実行される。

主に腕や手の障害でマウスが動かしづらい人が利用されている。

- ウィンドウとパネルのナビゲート
  ウィンドウ上の操作と特別なプログラム, アプリケーション等を実行でき最下部に広がるパネルをそれぞれナビゲートを実行するキーボードショートカットについて紹介する。
  - Alt + Tab
    各ウィンドウを表すアイコン付きのポップアップウィンドウを表示し, フォーカス (ウィンドウにキーボードやマウス等の入力装置から操作できる状態) したいウィンドウに到達するまでウィンドウ間を移動するには Alt キーを押したまま,Tab キーで押し続けることでフォーカスしたウィンドウにたどり着くことができる。
  - Alt +space
    ウィンドウにフォーカスされているとき (ウィンドウのタイトルバーがハイライト), そのウィンドウに指定されたキーを押すことで, ウィンドウの制御を操作でき, このキーはウィンドウ上のメニューを開くことができる。
  - Alt +F7
    ウィンドウにフォーカスがある状態でこのキーを押すとそのウィンドウを画面中に移動する操作ができ, マウスカーソルは十字の形に変わる。
  - Ctrl +Alt +Tab
    パネルにフォーカスを与える操作が実行される。このショートカットはデスクトップ全体の壁紙, パネル, アイコン付のポップアップウィンドウが表示される。また, それらの間でフォーカスを切り替えるには,Ctrl + Alt キーを押したまま,Tab キーを押し続けることでフォーカスさせたい位置にたどり着くことができる。
  - Shift + F10
    パネル上のメニューをナビゲートするために使用可能なキーボードショートカットで, このキーを押すとメニュー項目に関連するポップアップメニューを開くことができる。
  - 二次元メニューのナビゲーション
    一次元に並ぶドロップダウン式メニューと違い, ナビゲーションの選択肢をグループ化 (並列) することでスクロールの操作は不要になり, ユーザの選択できる内容を一目で見渡せることができる。

3　X ウィンドウシステム

3.1　概要

X ウィンドウシステムは MIT(マサチューセツ工科大学) で開発された UNIX ワークステーション上のウィンドウシステムである。利用されるワークステーションは一般に使われるパソコンより高性能化され, 多人数で情報を共有するスーパーコンピュータと同じ処理能力を持っている。またこのワークステーションは少人数で情報を共有でき, 価格は安価であること,OS は UNIX に適したウィンドウ環境を構築できる。その環境を構築する際, ウィンドウの外観とマウス, キーボードの操作は OS が MS-Windows の場合変更できなかった。しかし OS が UNIX だとそのシステム上にあるサーバクライアント・モデルが利用され, これによってウィンドウ環境をプログラミング技術がなくても簡単にユーザが扱いやすいように変更することが可能である。このウィンドウシステムについて以下のような特徴が見られる。

- サーバ/クライアント間の画面表示を他の端末 (ワークステーション) で見られる透過性
- 各種 GUI をベースにサポートできる
- 豊富なカスタマイズで利用者が提供したい操作性を構築できる
- デバイス (入出力装置) の直接操作はすべてサーバで処理されるので, アプリケーションはサーバ側のデバイスとは独立に設計できる
- 日本語入力が可能な国際化機能
- 多くのプラットフォーム (ワークステーションおよびコンピュータ環境) に対応できる

3.2　X Window の設計構築

X ウィンドウシステムではマウス・キーボード等の操作性を変更できることが挙げられた。では一体どのような仕組みでウィンドウ環境を構築されているか説明する。

先程紹介したサーバクライアント・モデルは X サーバとクライアントに分けられ, それらをネットワークでお互いに通信でやり取りしながらウィンドウ環境を構築できる. 図 3.1 のようにそれぞれクライアントはパソコンの操作性とウィンドウ環境の変更を要求するもので, それらの要求は X サーバに通してディスプレイ上で出力される。また,X サーバとク

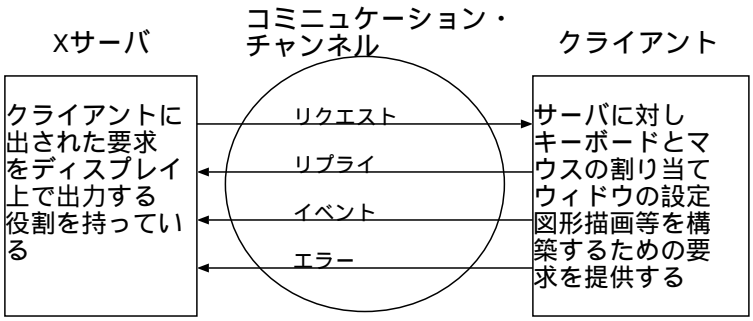

Fig. 3.1 X サーバ・クライアントモデル

ライアント間で使われる通信はコミニュケーション・チャンネルが利用されこの通信を実現するには図のように四つ,X プロトコルと呼ばれる役割を持っている。

- リクエスト

クライアントからサーバに送られる複数のメッセージで, サーバからの返事を全くない要求 (ウィンドウの構築要求, キーボードの割り当て等) と, サーバからの返事がある問い合わせ要求 (エラー報告) が分かれる。

- リプライ

クライアントからの問い合わせ要求 (エラー報告) に対し, 必要な情報 (修正点) をサーバが返信するメッセージ

- イベント

サーバ側で発生した X プロトコルをクライアントにレポートするのに必要なメッセージ

- エラー

イベントの一種だったが, クライアントからの要求に対してサーバ側でエラーが発見されたとき, そのエラーをクライアントに報告するためのメッセージ

以上のように X プロトコルによって X サーバとクライアント間の通信でウィンドウ環境の設定を実現する役割を持っている。しかし, それらの設定は専用ファイル内に変更し

たい内容を修正すればマウスやキーボードのカスタマイズができるが, ウィンドウマネージャの種類によって, ウィンドウの外観が異なったり, 設定内容も他のと違ったり, 設定できたのにできなかったりするのもある。次にそれについて紹介する。

4　ウィンドウマネージャ

X ウィンドウシステムを構成しているクライアント内のプログラムでユーザが会話形式で画面中のウィンドウ画面の位置や大きさ, フォーカスの指定等, 利用者が変更するときの処理に使われ, デスクトップの操作環境をカスタマイズすることが挙げられる。

またユーザから見ると, それらウィンドウ画面の雰囲気とマウスやキーボードの操作具合はウィンドウマネージャのカスタマイズによって左右されることがあり, これらのウィンドウ環境でうまく扱える人と扱えない人が分かれてしまう。しかしウィンドウマネージャは複数に分類されており, 利用者に合わせたウィンドウマネージャを選ばせることができる。それらの外観とカスタマイズはウィンドウマネージャの種類によって異なるが, 本研究はアクセビリティに適し, 障害者が利用しやすくするにはどのウィンドウマネージャを選んだほうが使いやすいか以下の図に合わせて紹介する。

4.1　各種ウィンドウマネージャ

主に代表的なウィンドウマネージャを以下のように紹介する。

- uwm

  1989 年に X11R4 が発表されるまでは一般的なマネージャ。

- mwm

  単独で機能するように設計されたマネージャで GNME や KDE と併用できない。

- ctwm

  twm の拡張版で, 複数の仮想スクリーンや数多くの機能をサポートできる。

- Fvwm

  立体的なウィンドウ表示, 仮想デスクトップ機能を備えたマネージャ。

- Fvwm2

  Fvwm をベースにしたバージョンで 2.2,2.4,2.6 など改良された物もある。

- Fvwm95

  Fvwm2 をベースに,MS-Windows 風の外観・操作にあわせたマネージャ

- Enlightenment

  最もデスクトップの外観が美しいマネージャで, さらにデスクトップのテーマを切り替えることができる。

- Window Maker

  Linux で標準に採用されているマネージャで, グラフィックとサウンド効果が高い。

- qvwm

  Windows95/98/NT の操作に熟練した人が扱えるマネージャ。

- sawfish

  Emacs で使われている Lisp 言語または GUI でカスタマイズ可能なマネージャ。

- AfterStep

  Fvwm1.24r をベースに, ウィンドウ形式を NeXT 風に変更したマネージャ。

- mlvwm

  カスタマイズすれば,MacOS8 風に構築できるマネージャ。

- IceWM

  Windows のようなインターフェースを提供する軽いマネージャ。

- Blackbox

  シンプルで見た目が良く, 他では表現できないサイトバッチな画面を生みだせる。

4.2 各種ウィンドウマネージャの比較

ウィンドウマネージャの特徴を述べたが, 各種類ごとに持っている機能は共通した機能と他のウィンドウマネージャだけの機能がそれぞれ挙げられる。以下のように二つのウィンドウマネージャを比較する。

- twm と ctwm の比較
  - 平面的な twm と異なり,ctwm はオプションとウィンドウのタイトルと枠線を持つ立体的な外観。

  - ctwm は壁紙 (ルートウィンドウ) を変えられるボタンツールが備わっており,twm の壁紙設定と違い複数のルートウィンドウ上の外観を変更できる。

- AfterStep と WindowMaker の比較
  - 共通として,NeXTSTEP 風なデスクトップ環境で WindowMaker は Next の優れた点を受け継いでおり, 画像形式をマウスでドラッグすることで, 画面上に固定可能なスライド式のアプリケーションメニューを開くことができる。
  - AfterStep は WindowMaker と異なり, ルックアンドフィール (パソコンの操作画面の見た目や操作) を細かく設定できることが挙げれられ, 主にウィンドウのタイトルバーを水平・垂直方向に配置する機能を持っている。

- twm と Fvwm の比較
  - ウィンドウのタイトルバー上に付属されているボタンは twm が 2 個だったのに対し,Fvwm は 10 個機能を複数に割り当てることができる。
  - twm のメニュー操作はマウスを押し続けることで表示できたが,Fvwm はマウスをクリックするだけで固定的なメニューを表示できる。

- Fvwm2 と Fvwm95 の比較
  - Fvwm95 はウィンドウの外観, メニューの形状, マウス操作は Fvwm2 と異なり MS-Win dows 風に近いデスクトップ環境で構築されている。また,MS-Windows 寄りのユーザが UNIX 上で使用する目的で作られている。
  - Fvmw95 は MS-Windows 風に GUI が構築されているため,Fvwm2 のような自由度の高い GUI を構築・追加することは難しい。

5 Fvwm2 カスタマイズ

私が選んだ中で最もアクセシビリティに適するウィンドウマネージャは Fvwm2 で, UNIX 上で動作する CPU の負担は少なく軽いウィンドウマネージャとして挙げられる。また, カスタマイズの自由度が高く, タイトルバー, フォント, キーボード等の設定は簡易にできることが考えられる。元々, バージョン 1.24r だったものが機能の拡張・改修されたものでそれ以降はバージョン 2.4,2.6 など機能だけでなくカスタマイズが追加されたものも作られた。これからカスタマイズ例を紹介する。さらに, カスタマイズに必要な設定ファイル (fvwm2rc) を以下の通りに記述した。

```
# some simple default key bindings:
```

```
key Tab  R    S   Exec xcalc
key F1   TW   C   Exec jnetscape
key F2   A    M   Exec kterm
```

これはキーボードの機能を割り当てるカスタマイズで, マウスクリックやコマンド入力しなくてもアプリケーションを開くことができる。一列目の設定はマウスカーソルがルートウィンドウ (コンピュータ画面全体 (R)) の位置にある状態で Tab キーと Shift キーを同時に押すことで xcalc(電卓) が起動できる。また, 二列目はマウスカーソルがウィンドウ (W) 内もしくはウィンドウのタイトルバー (T) の位置で F1 キーと Ctrl キーを押すことで jnetscape(netscape の日本版) が起動できる。最後はどの位置でもマウスカーソルを置いた状態で F2 キーと Alt キーを押すことで kterm(x の端末エミュレータ) が起動でき, なお,A のマウスカーソルの位置は一列, 二列の R(ルートウィンドウ) と T(ウィンドウタイトルバー),W(ウィンドウ内) いずれかの位置にマウスカーソルを置いても問題なく起動できる。

これらの操作をキーボードショートカットと呼び, マウスクリックとコマンド入力でアプリケーションを起動できる。しかし手間がかかるためキーボードに置き換えて起動できるようになり, 上記の設定でアプリケーション名の前に Exec(コマンドを実行する) を記述しなければならない。よって, 同時に押すボタンキーとマウスカーソルの位置を組み合わせることでキーボードだけで複数のアプリケーションの割り当てができると考えられる。しかし, 割り当てたキーとマウスカーソルの位置を設定し過ぎるとどの位置にボタンを押したら良いか迷ってしまうことが挙げられる。

5.1 デスクトップ上のフォーカス

キーボードのカスタマイズはアプリケーションを開く・ウィンドウを操作する作業をマウスの変わりにできることから, 腕や手の障害で全くマウス操作できない人に有効なカスタマイズと考えられる。アクセシビリティで紹介したキーボードベースのナビゲーションに該当する。普段, 開いたアプリケーションを作業するにはウィンドウ上でマウスクリックしアクティブ (ウィンドウタイトルバーの色が変わる状態) にすることでウィンドウ内にキーボード入力を受け取る状態になる。この操作をフォーカスと呼び, これをキーボードでも実行できる。以下のカスタマイズ例を紹介する。

```
# some simple default key bindings:

Key TAB A S Next [CurrentPage !Iconic !Sticky] focus
```

```
Key TAB A M Prev [CurrentPage !Iconic !Sticky] focus
```

これらを実行するには Next と Prev の [ ] 内の条件を全て一致することで, ウィンドウをフォーカスすることができる。Next は現在フォーカスされているウィンドウの次に,Prev は現在フォーカスされているウィンドウの前にフォーカスされる。また,[] 内の条件に含まれている内容の CurrentPage は現在表示されているページにウィンドウが表示されているか,!Iconic はウィンドウがアイコン化 (小さな長方形) になっていないか,!Sticky はウィンドウタイトルバーに複数の横線が引かれていないかが, それぞれの条件として満たさないとフォーカス操作を実行することはできない。

マウス操作の代わりにキーボードでフォーカスできるため, マウスをクリックしなくてもキーボードでウィンドウをフォーカスさせてから, そのウィンドウ上で文章を入力する利点が挙げられる。

しかし, 図 5.1 のようにデスクトップ上に複数のウィンドウが配置された場合, キーボードを押し続けることと, フォーカスさせたいウィンドウを順にキーボードを押し続けなければならない。つまり, ウィンドウを巡回にフォーカスする問題が生じてしまう。このような問題を改善するにはウィンドウマネージャFvwm2 に含まれているクライアント内のリソースを改修した上で, この問題が改善できているか本研究の課題として考察する。

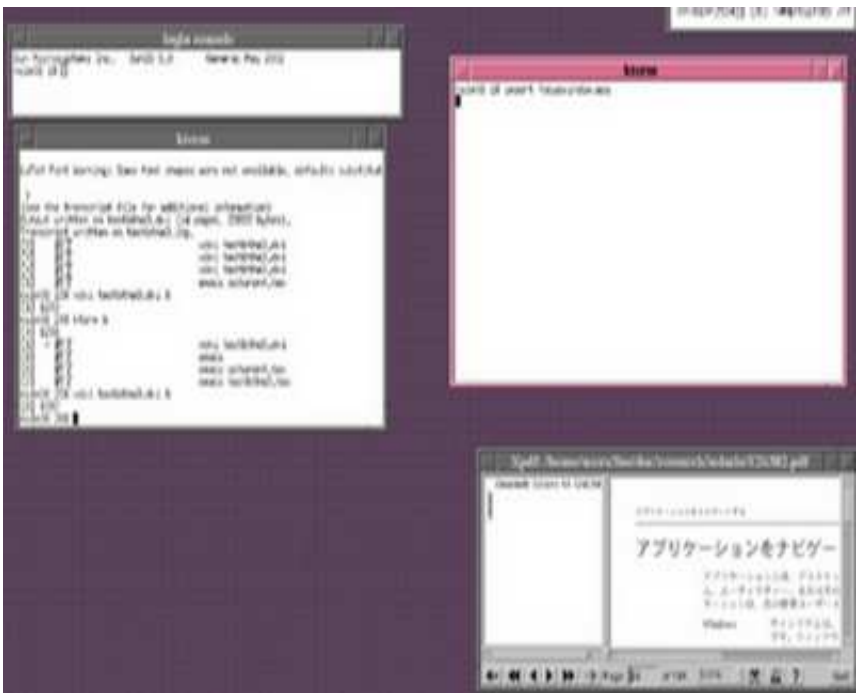

Fig. 5.1 適当な位置に配置されている複数のウィンドウ

6　デスクトップ上の位置を対象としたフォーカス

複数に配置されたウィンドウを巡回にフォーカスせず任意の位置にフォーカスさせるには, デスクトップ上の領域を図 6.1 のような左上, 左下, 右上, 右下の位置それぞれ Next と Prev の条件の中に組みこまなければならない。Fvwm2 のクライアント内リソースを改修し, それらの条件を対象とした巡回によるフォーカスの改善と腕や手の障害者がキーボード操作の負担 (キーボード押す回数, キーボード操作のしやすさ) を軽減できるか考察する。

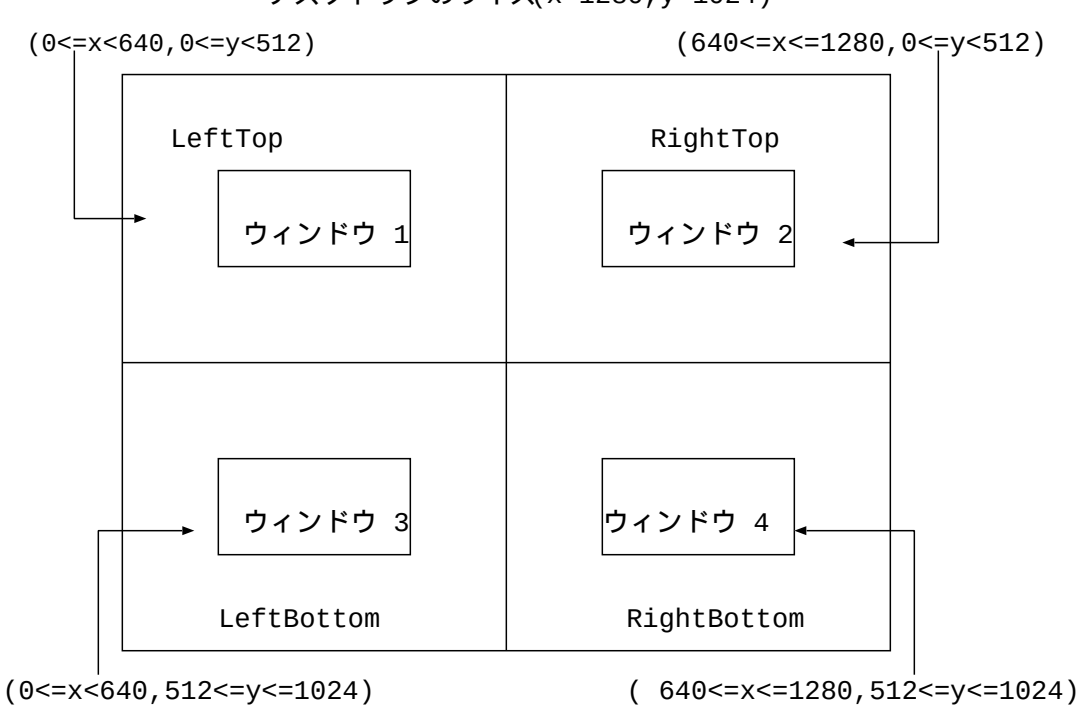

Fig. 6.1　対象にされるウィンドウ (左上, 左下, 右上, 右下)

6.1　改修に必要なリソース設定

図 6.1 で表した図のように Fvwm2 のカスタマイズを追加する。ここでは Fvwm2 の設定ファイル (fvwm2rc) を書き込むのではなく,Fvwm2 内のリソースは複数のプログラムで構成されており, 条件で働いているプログラムを修正しなければならない。これらを修正することで図 6.1 のように任意の位置に対象するフォーカス操作が実行できる。なお, 以下の通りに条件内に含まれるプログラムを記述する。

```
#file:fvwm.h
```

```
・
・
・
typedef struct WindowConditionMask
{
  struct
  {
・
・
・
unsigned needs_pos_rt : 1;
unsigned needs_pos_rb : 1;
unsigned needs_pos_lt : 1;
unsigned needs_pos_lb : 1;
   }my_flags;
```

まず,fvwm.h を typedef(新しいデータ型を作るのに利用する変数) で新しく定義される WindowConditionMask の構造体 (struct) に位置の名前 (右上:rt, 右下:rb, 左上:lt, 左下:lb) と 1 を四つそれぞれ記述して初期化する。

```
#file:conditional.c
・
・
・
void CreateConditionMask(char *flags, WindowConditionMask *mask)
{
  char *condition;
・
・
・
  while (condition)
 {
    else if (StrEquals(condition,"PositionRightTop"))
    {
    mask->my_flags.needs_pos_rt = 1;
    }
    else if (StrEquals(condition,"PositionLeftTop"))
    {
    mask->my_flags.needs_pos_lt = 1;
```

```
    }

    else if (StrEquals(condition,"PositionRightBottom"))
    {
    mask->my_flags.needs_pos_rb = 1;
    }
    else if (StrEquals(condition,"PositionLeftBottom"))
    {
       mask->my_flags.needs_pos_lb = 1;
    }
```

Conditional.c は fvwm.h に新しく定義された WindowConditionMask は CreateCondition Mask のポインターに使われる。また,fvwm.h で初期化した変数は if 文の条件を返すのに使われ,if 文の条件に『“ “』に区切られている部分の名前 (利用条件) を記述する。その名前は設定ファイル (fvwm2rc) に使用する条件として組みこませることができる。しかし, デスクトップ (画面) 上の位置を対象にしているため MatchesConditionMask 内に if 文を記述しなければならない。

```
#file:conditional.c
Bool MatchesConditionMask(FvwmWindow *fw, WindowConditionMask *mask)
{
・
・
・
 if(mask->my_flags.needs_pos_rt) {
    /* debug */
      fprintf(stderr, "DEBUG: (name,W,H,x,y) = (%s,%d,%d,%d,%d)\n",
      fw->name, Scr.MyDisplayWidth, Scr.MyDisplayHeight,
      fw->frame_g.x, fw->frame_g.y);
    if (fw->frame_g.x > Scr.MyDisplayWidth
  || fw->frame_g.x < Scr.MyDisplayWidth / 2
  || fw->frame_g.y >= Scr.MyDisplayHeight / 2
  || fw->frame_g.y < 0) {
    fprintf(stderr, "DEBUG: return 0\n");
      return 0;
    }else{ fprintf(stderr, "DEBUG: return 1\n");}
  }
```

```
if(mask->my_flags.needs_pos_lt) {
   /* debug */
     fprintf(stderr, "DEBUG: (name,W,H,x,y) = (%s,%d,%d,%d,%d)\n",
     fw->name, Scr.MyDisplayWidth, Scr.MyDisplayHeight,
     fw->frame_g.x, fw->frame_g.y);
   if (fw->frame_g.x < 0
 || fw->frame_g.x >= Scr.MyDisplayWidth / 2
 || fw->frame_g.y >= Scr.MyDisplayHeight / 2
 || fw->frame_g.y < 0) {
   fprintf(stderr, "DEBUG: return 0\n");
     return 0;
   }else{ fprintf(stderr, "DEBUG: return 1\n");}
 }


  if(mask->my_flags.needs_pos_rb) {
   /* debug */
     fprintf(stderr, "DEBUG: (name,W,H,x,y) = (%s,%d,%d,%d,%d)\n",
     fw->name, Scr.MyDisplayWidth, Scr.MyDisplayHeight,
     fw->frame_g.x, fw->frame_g.y);
    if (fw->frame_g.x > Scr.MyDisplayWidth
 || fw->frame_g.x < Scr.MyDisplayWidth / 2
 || fw->frame_g.y < Scr.MyDisplayHeight / 2
 || fw->frame_g.y > Scr.MyDisplayHeight) {
   fprintf(stderr, "DEBUG: return 0\n");
     return 0;
   }else{ fprintf(stderr, "DEBUG: return 1\n");}
 }


 if(mask->my_flags.needs_pos_lb) {
   /* debug */
     fprintf(stderr, "DEBUG: (name,W,H,x,y) = (%s,%d,%d,%d,%d)\n",
     fw->name, Scr.MyDisplayWidth, Scr.MyDisplayHeight,
     fw->frame_g.x, fw->frame_g.y);
   if (fw->frame_g.x < 0
 || fw->frame_g.x >= Scr.MyDisplayWidth / 2
 || fw->frame_g.y < Scr.MyDisplayHeight / 2
 || fw->frame_g.y > Scr.MyDisplayHeight) {
```

```
    fprintf(stderr, "DEBUG: return 0\n");
      return 0;
    }else{ fprintf(stderr, "DEBUG: return 1\n");}
  }
```

MatchesConditionMask は WindowConditionMask 内の構造体に初期化した変数を返すポインターを使用する。if の () にその変数を条件文に使われ, の内容を元に debug 内の name(利用条件),W(デスクトップの横幅=1280),H(デスクトップの縦幅= 1024),x(デスクトップ内右方向の位置),y(デスクトップ内下方向の位置) は fwvm2 の設定ファイル内 (fvwm2rc) の利用条件が設定されたキーを押すことで,kterm(x 端末エミュレータ) に出力される。

それらを出力させるため,g.x(x),g.y(y),MyDisplayWidyh(W),MyDisplayHeight(H) を図 6 .1 に表す対象範囲にしなければならない。まず, 不等号と OR 文が並んでいるが, 本来 OR はいずれか不等号に表す条件を複数ある場合, そのいずれか一つ当てはめることで成立する。しかし, このプログラム内は逆に出力されているため,AND として使われている。また, 不等号も逆方向に向き,=が付属・未付属しているものも逆に出力される。

まず, 例を rt(右上) の x,y,W,H で表すと

```
x<W/2||x>W||y<0||y>=H/2 → W/2<=x && x<=W && 0<=y && y<H/2
```

このようなプログラム文を含めて四つ記述し,fvwm.h と conditional.c をコンパイルさせてインストールすると fvwm2rc に conditional.c で定義した条件が使われる。

```
# some simple default key bindings:
key Tab  A   L  Next[CurrentPage PositionLeftTop]focus

Key Tab  A   S  prev[CurrentPage PositionLeftBottom] focus

Key Tab  A   M  Next[CurrentPage PositionRightTop]focus

key Tab  A   C  Prev[CurrentPage PositionRightBottom] focus
```

上記の設定ファイル内に任意の位置を対象としたウィンドウだけフォーカスさせる操作を実行できる。

それぞれの条件を説明すると, 一列目の PositionLeftTop は Tab+L(Caps] Lock) を押すと左上の位置だけフォーカスされ, デスクトップ (画面) は (x=1280,y=1024) で x の半分 (640) 未満かつ y の半分 (512) 未満の中にウィンドウがあればフォーカスの対象に当たる。

二列目の PositionLeftBottom は Tab+S(Shift) を押すと左下の位置だけフォーカスされ,x の半分 (640) 未満かつ y の半分 (512) 以上から y(1024) 以下の中にウィンドウがあればフォーカスの対象に当たる。

三列目の PositionRightTop は Tab+M(Alt) を押すと右上の位置だけフォーカスされ, 右下は x の半分 (640 以上) から x(1280) 以下かつ y の半分 (512) 以上から y(1024) 以下の中にウィンドウがあればフォーカスの対象に当たる。

最後の四列目の PositionRightBottom は Tab+C(Ctrl) を押すと右下の位置だけフォーカスされ, 右下は x の半分 (640 以上) から x(1280) 以下かつ y の半分 (512) 以上から y(1024) 以下の中にウィンドウがあればフォーカスの対象に当たる。

6.2 改善した結果 1

左上, 左下, 右上, 右下を対象としたウィンドウにそれぞれフォーカスすることができた。下記の通りに実行したキーボードの操作性に関して良い点と悪い点をまとめた。

- 良い点
  - 対象にされるウィンドウだけフォーカスできるので, キーを押しつづける操作は少なくなった。
  - キーボードでフォーカス操作できるため, 離れたウィンドウを巡回せずフォーカスしやすくなった。
  - 対象とされるウィンドウが 1 個ずつそれぞれ配置された場合, 割り当てられたキーの押す回数は少なくなる。

- 悪い点
  - ウィンドウサイズを大きくしてしまうと, どの位置にフォーカスしたかわからなくなってしまう。
  - ウィンドウを真ん中に配置すると, キーボードでフォーカスを割り当てられるキ ーを押し間違えてしまう。
  - パソコンに使い慣れている人は問題ないが, 初心者が扱う場合, 割り当てられたキーを覚えるのに時間がかかる。

6.3 他に考えられる位置の対象

先程結果を述べたようにキーボードのショートカットで, デスクトップ上のフォーカスされる位置の対象は良い点と悪い点がそれぞれ挙げられている。パソコン初心者が腕や手の障害で全くマウスを動かせない人がパソコン操作をできるか難しい。特に設定されたキーボード配列を四つ覚えることと, 対象にされる位置も把握しなければならない。このようにパソコンに使い慣れている人は問題ないが初心者が扱う場合それらの問題点が生じる。したがって, 他にウィンドウの位置を対象にする範囲を減らすことが考えられる。図 6.2 のように上下対象としたウィンドウフォーカス操作を実行する。

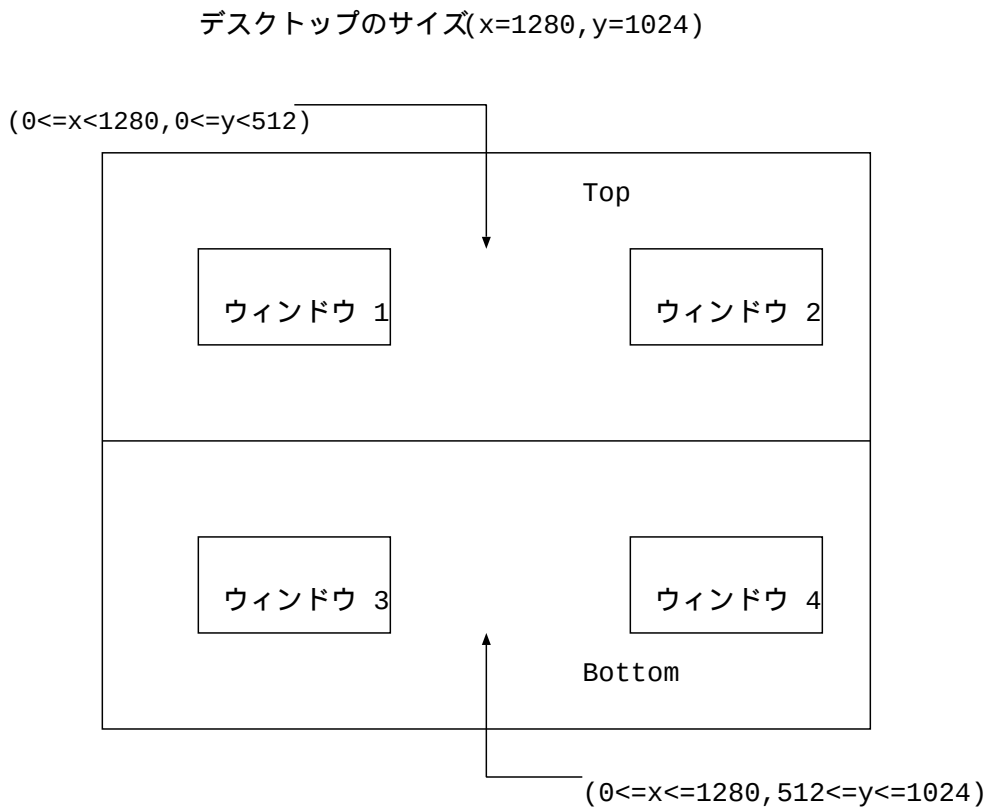

Fig. 6.2 対象にされるウィンドウ (上下)

条件内に含まれるプログラムを以下の通りに追加した。

```
#file:fvwm.h

typedef struct WindowConditionMask
{
```

```
  struct
  {
   unsigned needs_pos_top: 1;
   unsigned needs_pos_bottom:1;
  } my_flags;

#file:conditional.c

void CreateConditionMask(char *flags, WindowConditionMask *mask)
{
  while (condition)
  {
    else if (StrEquals(condition,"PositionTop"))
    {
    mask->my_flags.needs_pos_top = 1;
    }
    else if (StrEquals(condition,"PositionBottom"))
    {
       mask->my_flags.needs_pos_bottom = 1;
    }

Bool MatchesConditionMask(FvwmWindow *fw, WindowConditionMask *mask)
{
    if(mask->my_flags.needs_pos_top) {
    /* debug */
      fprintf(stderr, "DEBUG: (name,W,H,x,y) = (%s,%d,%d,%d,%d)\n",
      fw->name, Scr.MyDisplayWidth, Scr.MyDisplayHeight,
      fw->frame_g.x, fw->frame_g.y);
    if (fw->frame_g.x < 0
  || fw->frame_g.x > Scr.MyDisplayWidth
  || fw->frame_g.y >= Scr.MyDisplayHeight / 2
  || fw->frame_g.y < 0) {
    fprintf(stderr, "DEBUG: return 0\n");
      return 0;
    }else{ fprintf(stderr, "DEBUG: return 1\n");}
  }

    if(mask->my_flags.needs_pos_bottom) {
    /* debug */
```

```
    fprintf(stderr, "DEBUG: (name,W,H,x,y) = (%s,%d,%d,%d,%d)\n",
    fw->name, Scr.MyDisplayWidth, Scr.MyDisplayHeight,
    fw->frame_g.x, fw->frame_g.y);
  if (fw->frame_g.x < 0
|| fw->frame_g.x > Scr.MyDisplayWidth
|| fw->frame_g.y < Scr.MyDisplayHeight / 2
|| fw->frame_g.y > Scr.MyDisplayHeight) {
  fprintf(stderr, "DEBUG: return 0\n");
    return 0;
  }else{ fprintf(stderr, "DEBUG: return 1\n");}
}
```

6.1 で紹介したリソース設定は MatchesConditionMask の対象範囲の設定以外, 同様に記述しているため省略する。

対象範囲の設定されている top(上) の例は

```
x<0 || x>W || y<0 || y>=H/2 → 0<=x && x<=W && 0<=y && y<H/2
```

同様に二つ記述し,fvwm.h と conditional.c をコンパイルさせてインストールする。

```
# some simple default key bindings:
Key Tab   A   S    Next [CurrentPage PositionTop]focus

key Tab   A   C    Prev [CurrentPage PositionBottom] focus
```

上記の設定ファイル内に新たな条件として一列目の Top はデスクトップ (x=1280,y=1024) の x(1280) 以下かつ y の半分 (512) 未満の中にウィンドウがあれば上の範囲だけフォーカスが実行される。

二列目の Bottom は x(1280) 以下かつ y の半分 (512) から y(1024) 以下の中にウィンドウがあれば下の範囲だけフォーカスが実行される。

6.4 改善した結果 2

上下対象としたウィンドウを実行した結果, 下記の通りにキーボードの操作性に関して良い点と悪い点をまとめた。

- 良い点
  - 対象にされる範囲は少なく, フォーカス対象で割り当てたキーは覚えやすい。
  - キーを割り当てる組合せは (上:Tab+Shift, 下:Tab+Ctrl) わかりやすいので初心者でも扱いやすい。

- 悪い点
  - 六つ以上ウィンドウを上下それぞれ三つ以上並んでしまうと, それらのウィンドウをフォーカスするのにキーを押し続けなければならない。
  - ウィンドウが複数配置されると比較的にボタンの押す回数が多い。

6.5 考察

今回はデスクトップ上のウィンドウに位置対象を条件にしたキーボードでフォーカス操作を実行した。結果として, ウィンドウをキーボードでの作業を効率良くフォーカスすることができ, キーを押す回数を少なくしたり, フォーカス操作の巡回が少なくなることがわかった。しかし, 悪い点としてウィンドウサイズを大きくする, 配置を真ん中にしてフォーカスを実行した場合, フォーカス操作がしにくくなることと対象範囲を増減することで, 手や腕の障害でパソコンを使う熟練者と初心者の片方だけの人が扱いやすいことが考えられる。こういった人が両方扱える場合フォーカス操作を改善しなければならない。よって改善点としてウィンドウサイズの範囲指定を考えなければならない。

7 まとめ

今回の研究はアクセシビリティとウィンドウマネージャの種類を紹介し, キーボードナビゲーションの新たな改善としてデスクトップ (画面) 上のウィンドウを位置対象とした条件でフォーカスするキーボードで操作するカスタマイズを実行した。結果, 指定された範囲だけフォーカスでき, 複数に配置されたウィンドウを巡回する問題を改善できた。しかし, ウィンドウのサイズを大きくしてしまうと割り当てたキーを押し間違えたり, 対象とされる位置を減らしてしまうとフォーカスの巡回が多くなる問題が残った。このような問題の改善を見つけなければならない。

また, ウィンドウマネージャに関して,Fvwm2 をベースとしたカスタマイズの他にも, Fvwm95 のウィンドウマネージャ上で, 指定された位置でのフォーカスもカスタマイズできれば, より MS-Wondows を利用する障害者がパソコン操作しやすくなることも考えられる。また, 人がパソコン操作する中で, 障害者にとってパソコンを使いやすく, 利便性を

兼ね備えるにはどのようなウィンドウマネージャのカスタマイズが有効か見つけることが, 今後の課題になる。

参考文献

[1] Oracle.Parkway: “Oracle Solaris 11 GNOME デスクトップのアクセシビリティーガイド”, http://docs.oracle.com/cd/E2692401/pdf/E26302.pdf

[2] 日経 BP 社: “見過ごしてはならない Linux のデスクトップ 10 選”, http://itpro.nikkeibp.co.jp/article/MAG

[3] 東京反訳 (株): “音声認識に関するリンク集”, http://www.8089.co.jp/onsei-ninshiki/56

[4] 山岸 寛: “ウィンドウマネージャのユーザインターフェースに関する考察”, 新潟工科大学工学部情報電子工学科卒業論文 (2003 年)

[5] Matt Chapman: “Window Manages for X”, http://www.PLIG.org/xwinman

[6] 矢吹道朗, 江面敦: ウィンドウマネージャ徹底解説 (テクノプレス, 2000 年)

[7] 水澤 智裕: 入門 X Window (アスキー出版局, 1993 年)