Chapter 7

Nutsによるアプリケーション開発

これまでの章で、サブクラス拡張が可能なホワイトボックス的なコンポーネント Nuts と、ベクターコンポーネントを用いたコンポーネント差分プログラミングの再利用性、柔軟性について論じた。本章では、Nuts コンポーネントを用いた開発事例について報告する。

7.1 Nuts/Builder

7.1.1 Nuts/Builder とは

Nuts コンポーネントを用いたプログラミングでは、コンポーネント結合の初期構造を構造記述ファイル中に列挙する (第 3.1節参照)。これは、プログラマがテキストエディタで作成しても良いが、ビジュアルなツールによりコンポーネントを積み上げることで、構造記述ファイルを生成することもできる。そのようなツールとして Nuts/Builder(図 7.2) を開発した。Nuts/Builder では、部品バンク (部品の種類がリストされたウインドウ) からワークシート上に必要な部品をドラグして積み上げていくことにより、簡単に構造記述ファイルを生成できる。

Nuts/Builder 自体は、初期状態で 381 個の Nuts 部品の積み上げで構築されている。Nuts/Builder の部品積み上げの様子を図 7.1に示す。全コード量はおよそ 6800 行であり、Nuts/Builder 固有の部品として表 7.1に挙げる 56 種の部品を開発した。開発には、1.0 人・月を要した。

7.1.2 Nuts/Builder で導入した透明コンポーネント

Nuts/Builder で用いた透明コンポーネント／ベクターコンポーネントを以下に挙げる。

7.1.2.1 workSheetManager/rootManager コンポーネント

workSheetManager クラスは NutsGroupManager クラスのサブクラスである。NutsGroupManager コンポーネントは、図 7.2の Nuts/Builder のワークシートを管理し、一つの部品

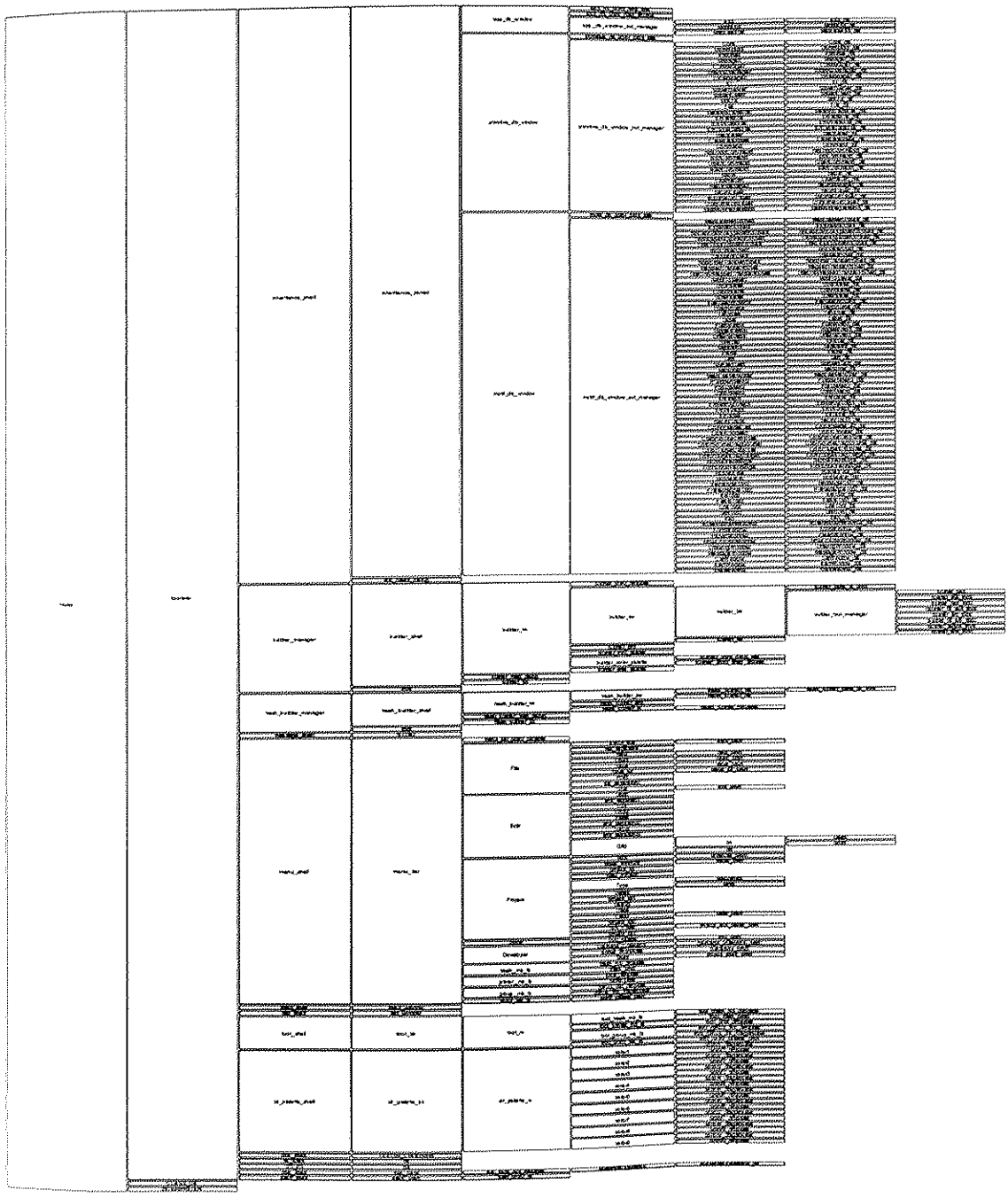

図 7.1: Nuts/Builder の部品積み上げの様子

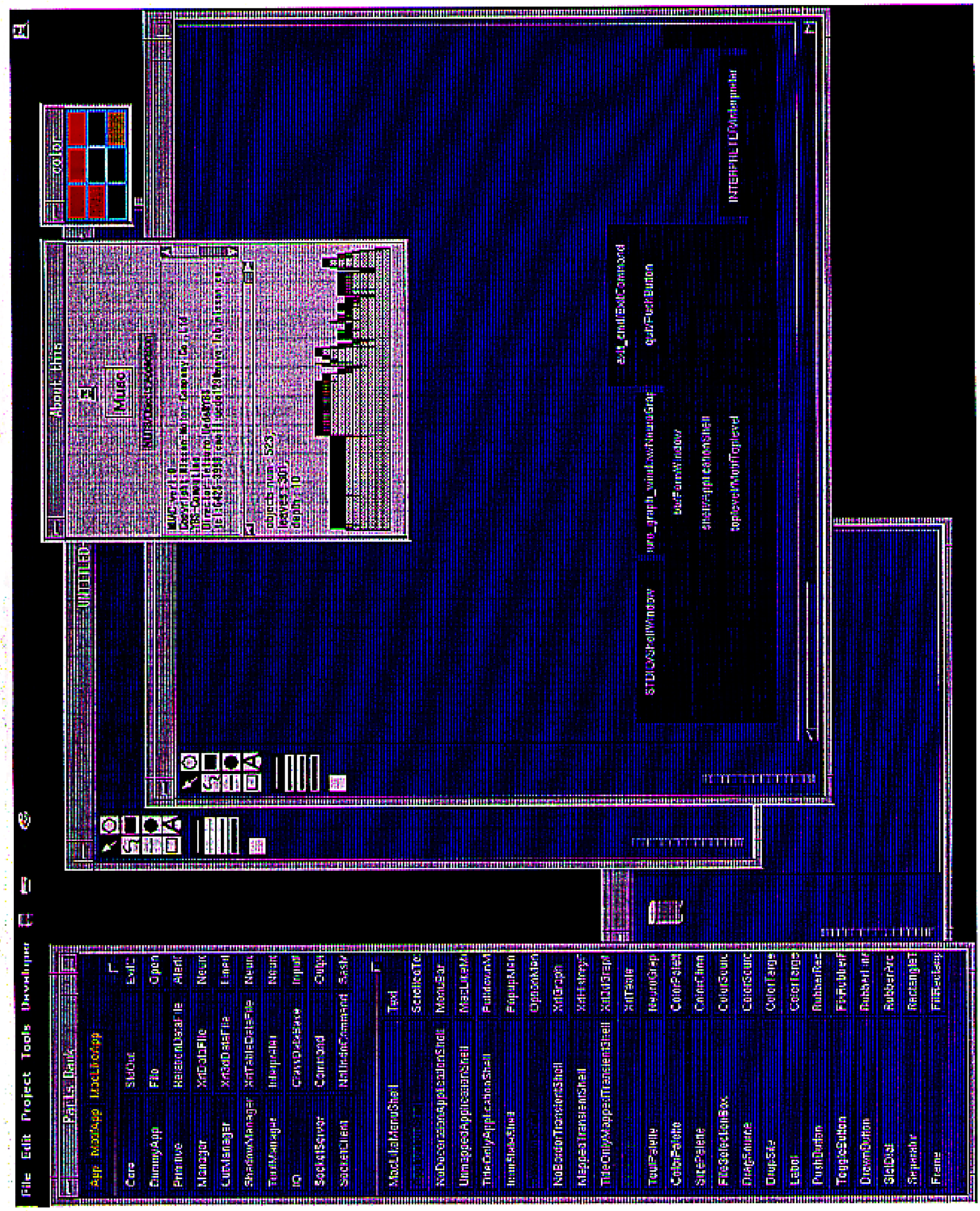

図 7.2: Nuts/Builder の画面イメージ

表 7.1: Nuts/Builder で用いた部品のクラス (数字は行数)

Muac	450	MuacPartsDialog	224
MuacApp	282	MuacPartsDragIcon	41
MuacAppInheritanceWindow	51	MuacPartsLabel	161
MuacAtomMrg	37	MuacPartsLabelDragSource	105
MuacBuilderCmd	75	MuacPartsLabelDropSite	118
MuacBuilderManager	889	MuacPartsLabelManager	69
MuacBuilderWorkArea	74	MuacPopupShellCmd	50
MuacCatMTmpCmd	71	MuacPopupSubClassCmd	54
MuacCheckWindowHierarchyCmd	49	MuacPrimitiveInheritanceWindow	59
MuacClearCmd	50	MuacPrintDropSite	131
MuacColorPaletteCascade	75	MuacPrinterCascade	75
MuacColorPaletteCmd	52	MuacPrinterIcon	36
MuacColorPaletteDropSite	69	MuacProjectMenu	216
MuacColorPaletteIcon	40	MuacSaveAsCmd	48
MuacExecuteCmd	52	MuacSaveCmd	48
MuacFileSelectionBox	164	MuacStartupWindow	258
MuacGroupCascade	75	MuacSubClassDialog	234
MuacGroupDragSource	108	MuacSubClassDropSite	67
MuacGroupDropSite	70	MuacToolPaletteCmd	52
MuacGroupIcon	31	MuacToolPaletteShell	58
MuacGroupIcon2	10	MuacTrashBoxCascade	75
MuacInheritanceWindow	116	MuacTrashBoxIcon	187
MuacMenuWindow	38	MuacTrashBoxIcon2	38
MuacMipBankWindow	131	MuacTrashBoxLabel	97
MuacMipPartsLabel	128	MuacTrashCmd	53
MuacMotifInheritanceWindow	52	MuacTrashDropSite	74
MuacNewCmd	52	MuacTrashManager	157
MuacParts	534	MuacWorkAreaDropSite	89

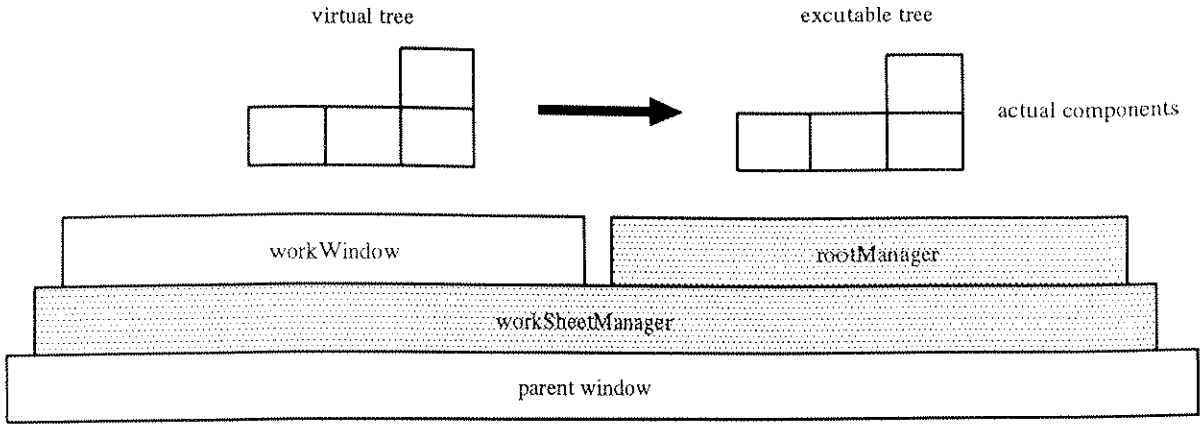

図 7.3: workSheetManager と rootManager

として扱えるようにするグループマネジャである。ワークシート内でのコンポーネントアイコンの追加／移動／削除などの処理は、すべて workSheetManager コンポーネントを介し、部分木内で閉じた制御となっている。例えば、ワークシートに新たなコンポーネントがドラグされた場合は、この情報がウインドウから workSheetManager コンポーネントに通知され、新たなコンポーネントが workSheetManager コンポーネント上の仮想的なアプリケーション木の上に配置される。

複数のワークシートで作業するためには、workSheetManager コンポーネントに`clone`メッセージを送り、ワークシートを複製して用いる。このように workSheetManager コンポーネントを用いることにより、ワークシートが一つの高機能コンポーネントとして利用できる。

仮想的なアプリケーション木とは、積み上げ中のコンポーネントのクラス名、部品名だけを収めた疑似的なコンポーネントの木である (図 7.3の virtual tree)。これ自体は、ワークシート上での積み上げの状態を表示／編集するためのウインドウリソースを管理するために存在している。

積み上げ中のアプリケーション木は、直ちに実行／デバッグが可能である。その場合は、仮想アプリケーション木に基づいて実際の部品木が仮想的なルートコンポーネントである rootManager コンポーネント (グループマネジャから派生) の上に構築される (図 7.3の excutable tree)。単独で動作する場合は、部品木はルートコンポーネントの上に構築されるが、この場合は rootManager コンポーネントがルートコンポーネントの代理を果たす。例えば、Nuts/Builder で構築中のアプリケーションがウインドウアプリケーションの場合は、rootManager コンポーネントがダイアログシェルとして機能し、rootManager 以下の部品木を単独のウインドウアプリケーションであるかのように動作させる。

設計が完了した時は、rootManager コンポーネントをルートとする部分木のコンポーネント間の親子関係を出力することで、構造記述ファイルが生成される。

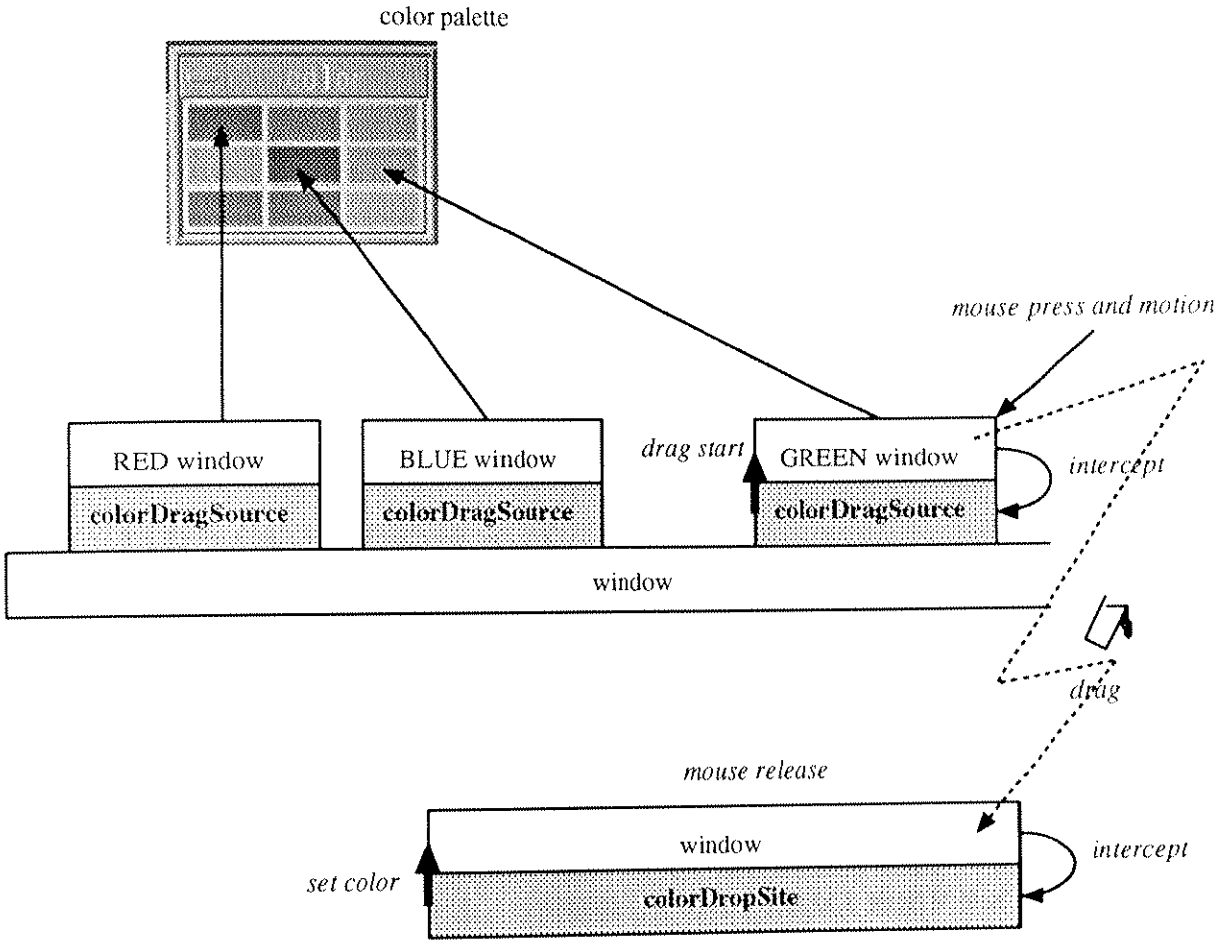

図 7.4: dragSoruce と dropSite

7.1.2.2 dragSource/dropSite コンポーネント

前節の基本的な Nuts/Builder に、ベクターを用いて画面上のウインドウリソース (アイコンやラベルなど) をマウスでドラグ &ドロップする機能を追加した。

dragSource コンポーネントは、アイコンなどをドラグ可能にし、ドロップ (ドラグ中にマウスを放した) 時に dropSite コンポーネントと交信する。dropSite コンポーネントは、dragSource コンポーネントからのドロップアクションを検知して、dragSource と交信し、実際の処理を行う。これらを共にベクターコンポーネントとして実装することで、既存のウインドウリソースにドラグ／ドロップの操作を適宜追加可能とした。

dragSource/dropSite の適用例として、colorDragSource/colorDropSite 各ベクターコンポーネントについて説明する (これらは Nuts 部品としてライブラリ化した)。

colorDragSource コンポーネントを図 7.4のようにウインドウに挿入すると、ウインドウのバックグラウンド色を引数としたドラグソースになる。すなわち、colorDragSource が結合したウインドウは、その上でマウスをドラグすると、バックグラウンド色に着色された

バケツアイコンが現れる。これを用いて、図 7.4左上のような様々な色のバックグランドを持ったウインドウを作り、それぞれを colorDragSource と結合させるだけで、ドラッグ可能な色パレットができる。

次に colorDropSite コンポーネントを別のウインドウ (dragSource と同じでも良い) に挿入すると、そのウインドウ上で colorDragSource のドラグアイコンがドロップされたとき、ウインドウが colorDragSource が持って来た色に着色される。

colorDragSource/colorDropSite は共にベクターコンポーネントなので、どこにでも挿入可能であり、これらが結合した任意のウインドウをカラーパレットにしたり、カラーターゲットにできる。

Nuts/Builder は、ドラグ &ドロップ操作でワークシート上に部品木を構築できるインターフェースを備えており、dragSource/dropSite 各クラスのコンポーネントを多用している。

7.2 DRMA 地図ドライバ

7.2.1 DRMA 地図ドライバとは

(財) デジタル地図協会 (DRMA) は、国土地理院発行の地図を基に、デジタル化した道路地図を発行している。これを DRMA 地図データベースと呼ぶ。DRMA 地図データベースは、10km 四方を 1 つのメッシュとした道路のリンク情報を持っており、DRMA 地図データベースの道路リンク情報を用いることで地図を用いたアプリケーション (第 8章の Navimos はその例) の開発が可能となる。DRMA 地図ドライバは、DRMA 地図データベースを読み取り、道路のリンク情報を作成する。今回開発したのは、DRMA 地図ドライバと、DRMA 地図ドライバを用いて作成した道路リンク情報を画面上に表示する DRMA 地図ビュアー (図 7.5) である。具体的には以下のような部品群を開発した。

- メッシュ単位でのデータの読み込みが可能な DRMA 地図ドライバ
- 画面上の任意の場所をマウスでドラグすることによる地図画面のスクロール
- 画面のキャッシュ／スワップによる不便を感じないレベルのスクロール
- 画面縮尺の拡大／縮小

開発は、Linux 上で gcc、X-Window、lesstif を用いて行った。コード量 (Nuts ライブラリは除く) は 1940 行で、0.3 人・月を要した。

7.2.2 地図データの読み込み

7.2.2.1 DRMA 地図データベースの構成

DRMA 地図データベースは、図 7.7のように、1 次／2 次メッシュで管理されている。2 次メッシュは 10km × 10km の正方形であり、内部のリンクやロケーション情報は左下角からのメートルオーダー (0〜10000) 座標で記憶されている。

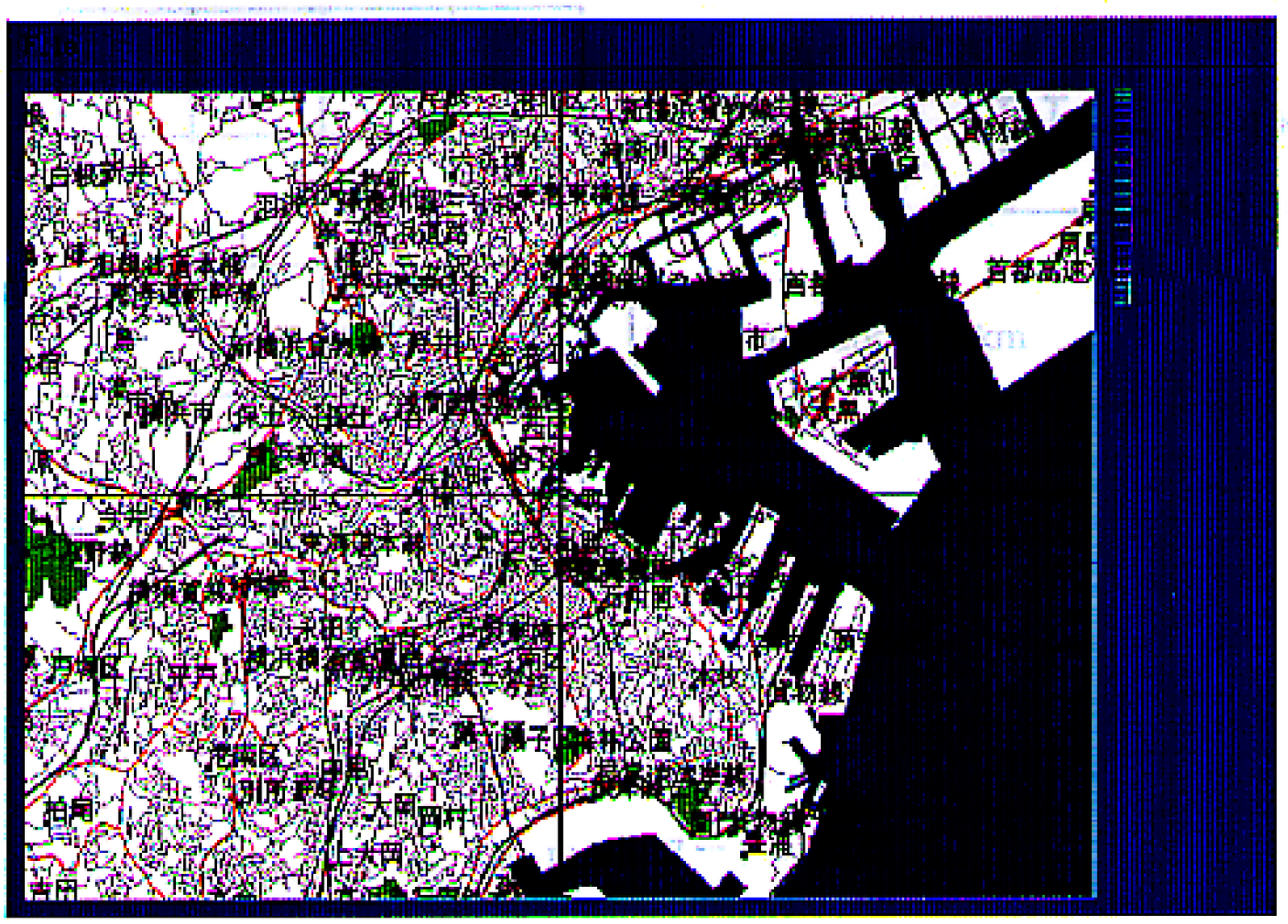

図 7.5: DRMA 地図ビューア

1次メッシュにはユニークな4桁の番号がふられていて、上位下位2桁づつで経度緯度の値を表している (図 7.6)。1次メッシュは、2次メッシュを8×8した正方形になっている。

2次メッシュは2桁の番号で管理され、1次メッシュ内の左下角を 00 とした相対的な位置を表している。2次メッシュの一区画は、2万5千分の1の地形図に相当する。データは、2次メッシュ毎にファイルに格納されている。例えば、[5239]1 次メッシュの [74]2 次メッシュは、“523974.dat” というファイルに格納される。

7.2.2.2 メッシュデータのクラス

メッシュのデータを統一的に扱うために、図 7.8に示す5つのクラスを作成した。以下に各クラスについて説明する。

- mesh クラス：mesh の基底クラス。メッシュの番号 (_id)、メッシュの位置 (_x,_y)、内部に含まれるデータの数 (_num)、などの各メンバー変数を持つ。生成時に_x、_y、_id の値を受け取る。公開メソッドとしてdraw()、load() を持つ。

- mesh1 クラス：1次メッシュのクラス。内部に 8 × 8mesh 型参照を保持するマトリックスを持つ。

- mesh2 クラス：2次メッシュのクラス。実際にdraw()、load() を定義している。

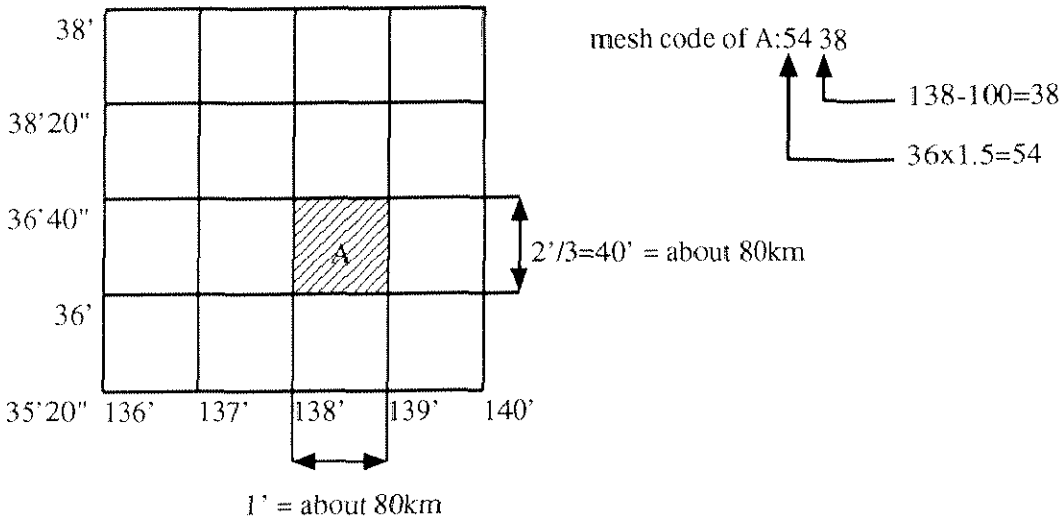

図 7.6: mesh コードの規則

- meshUser クラス：任意のサイズの 2 次メッシュを保持できるクラス。meshUser クラスの生成時引数は、保持する 2 次メッシュの横縦の数を意味する。

- meshDummy クラス：ファイルが存在しないなどの理由で、空白となる 2 次メッシュを表す。

7.2.2.3 内部的なメッシュ番号とメッシュの相対位置計算

前述したように、データファイルは 2 次メッシュの番号毎に記憶されているため、2 次メッシュの ID を、上位 4 桁に 1 次メッシュの番号を、下位 2 桁に 2 次メッシュの番号を用いた 6 桁で表す。

この場合、本来の 2 次メッシュの番号は 8 進数で 00～77 であるため、2 次メッシュ内の相対位置計算は、8 進数で計算する方が簡単である。このため、2 次メッシュの番号を mesh2 に格納する際、下 2 桁は本来の 8 進数の値に変換して記憶する。例えば 523974 は、下 2 桁を 8 進数と見た場合の 10 進数 523960 として記憶する。これにより、下 2 桁を取り出せば直ちに 8 進数演算が可能になる。

8 進数演算を用いることで、ある 2 次メッシュから特定の相対位置にあるメッシュの番号を簡単に計算できる。たとえば、523974 から北に 4、東に 7 メッシュ分離れたメッシュの番号を簡単に計算できる。これらのメッシュの相対位置計算は、画面スクロールに連動した動的なスコープ変更に必要となる。

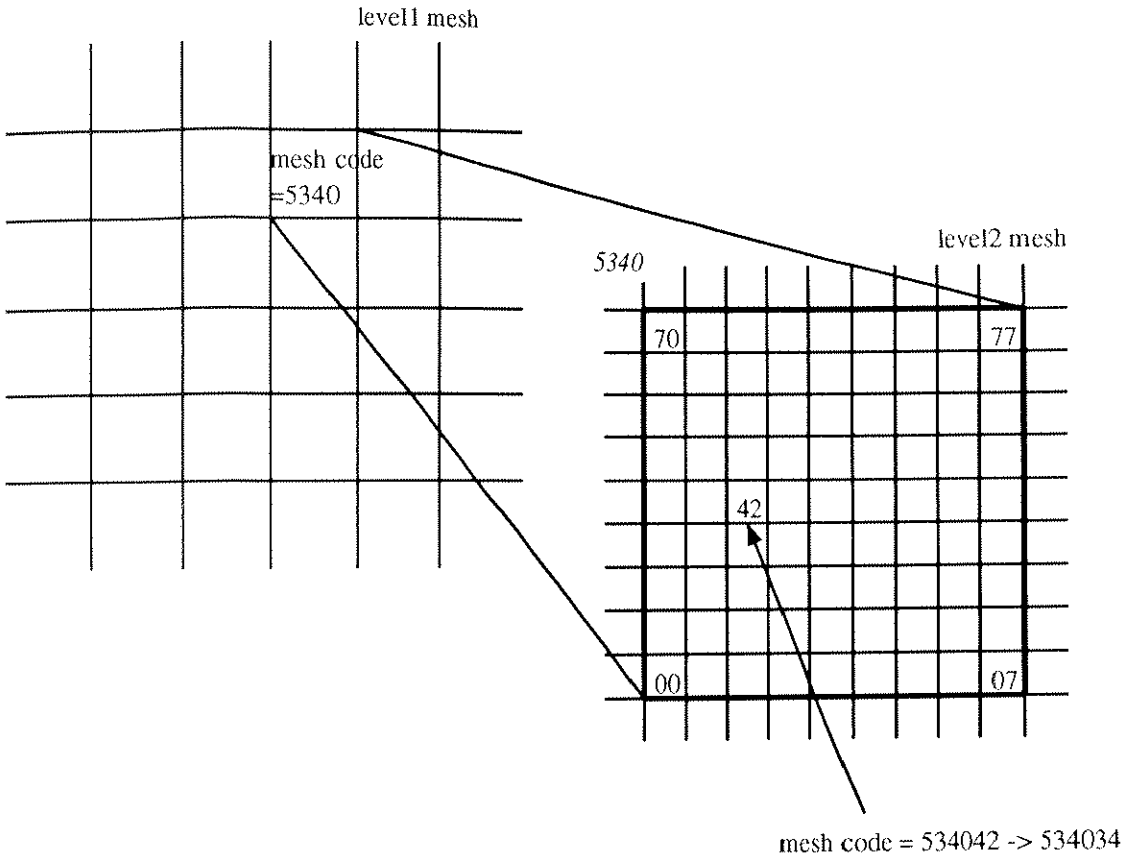

図 7.7: DRMA 地図データベースのメッシュ構造

7.2.2.4 地図要素のクラス

地図要素としては、ARoadLink、BRoadLink、Railway、Sea、Shape、Locate がある。それぞれ、一般道、国道、鉄道、海岸線や河川、公園などの敷地、地名を表している。これらの地図要素を統一的に扱うクラスとして、mapGlyph クラスを定義した。そして、具体的な地図要素を mapGlyph のサブクラスとして定義した (図 7.9)。mapGlyph には、`draw()`、`load()` などのメソッドがあり、実際に ARoadLink などのオブジェクトをバインドした後は、

```
mapGlyph *mg -> load()
```

という統一的な方法で地図要素にアクセスできる。

全体的なデータの参照関係は図 7.10のようになる。

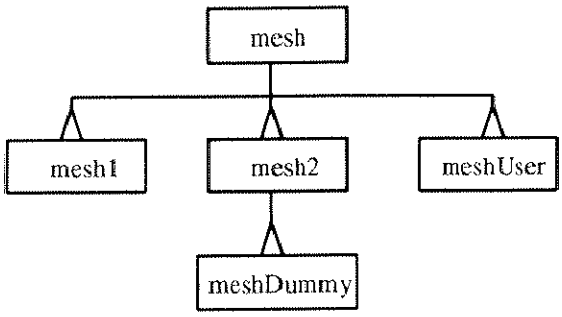

図 7.8: mesh クラスの階層構造

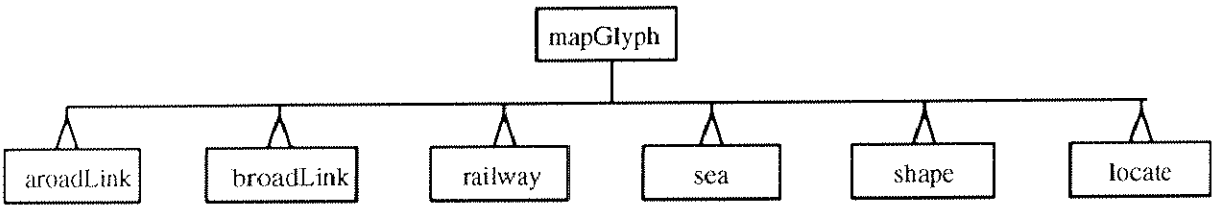

図 7.9: 地図要素クラスの階層構造

7.2.3 画面表示とスクローリング

7.2.3.1 初期表示

ディスプレイのサイズにほぼ等しい画像バッファを用意する。次にバッファサイズに見合った[1]meshUser オブジェクトを用意し (左肩のメッシュ番号を指定する)、

```
meshUser *mesh = new meshUser(523964);
mesh -> load();
mesh -> draw();
```

とするだけで表示される。

7.2.3.2 バッファ内スクロール

図 7.11に示すように、ユーザは画像バッファ中の一部分をスコープから覗いている形になる。スコープが画像バッファからはみ出さない間は、スコープのビューポイントを変更するだけなので、スクロールは高速に行われる。

[1]例えば画面が 1000 × 1000 ドットで、2 次メッシュを 500 × 500 ドットで表示する場合は、meshUser としては、2 × 2 の mesh2 オブジェクトを保持できるようにする

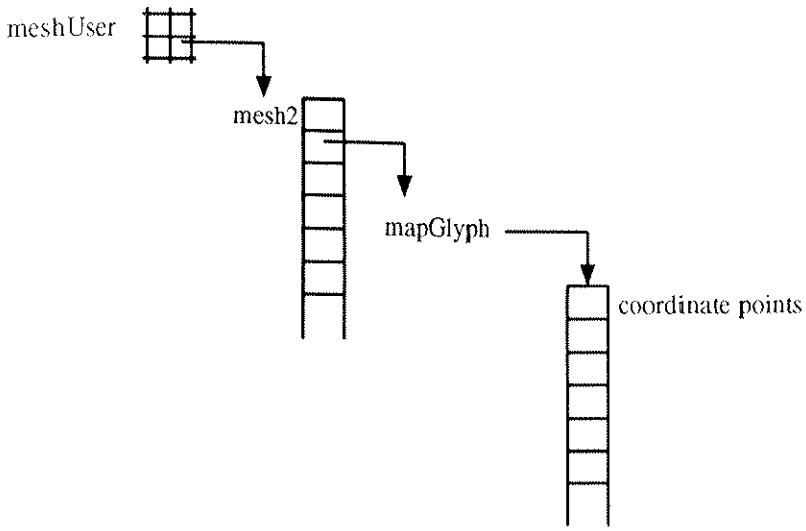

図 7.10: オブジェクト間の参照関係

このスクロール機能には、Nuts コンポーネントのスクロール部品 (NutsScopeWindow) を利用した。NutsScopeWindow クラスは、画像バッファに最初に一度だけ表示した画像を、スコープのビューポイントを移動することにより、高速にスクロールする。mapvScopeWindow(地図を表示するためのウインドウ) は、ベクターコンポーネントではなく、NutsScopeWindow をサブクラス拡張して用いる。その理由は、今回の地図表示のように、リンク数が膨大 (1 メッシュあたり 1 万以上) で描画に時間がかかる場合は、NutsScopeWindow によるスクロールの方が性能面から見て適しているためである。

7.2.3.3 バッファ外スクロール

スコープがバッファからはみ出したときは、対応するエリアを新たにデータベースから読み込まなければならない。例えば図 7.12に示すように、スコープが右に外れた場合は、バッファの右半分をバッファの左半分に移し、右側を新たに読み込んだ後にビューポートをバッファの横幅の半分だけ左にずらす。

このようなスワップ作業をすべての方向について記述すると、あたかも広い範囲の画像バッファがあるようにスクロールが可能になる。

スワップ作業は、前述したスクロール部品 (NutsScopeWindow) によって、スコープが画像バッファからはみ出したとき`clipover()` というメソッドが呼ばれるようになっているので、これをオーバーライドする。実際のデータスワップ処理は meshUser に記述した。

7.2.3.4 データキャッシュ

スワップ機能を用いると、広大な画像エリアを表示できるが、スワップするときにファイルへのデータアクセスが発生するため、若干時間がかかる。この問題に対処するために、

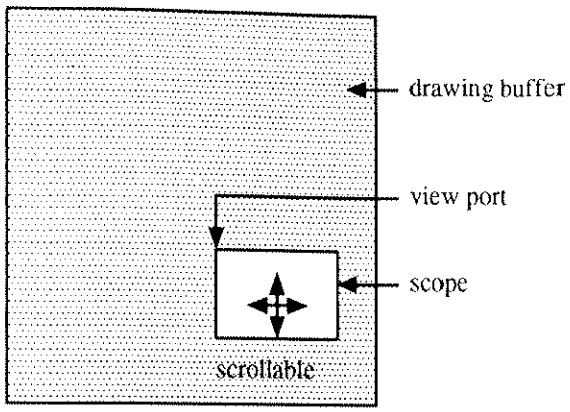

図 7.11: スコープと画像バッファの関係

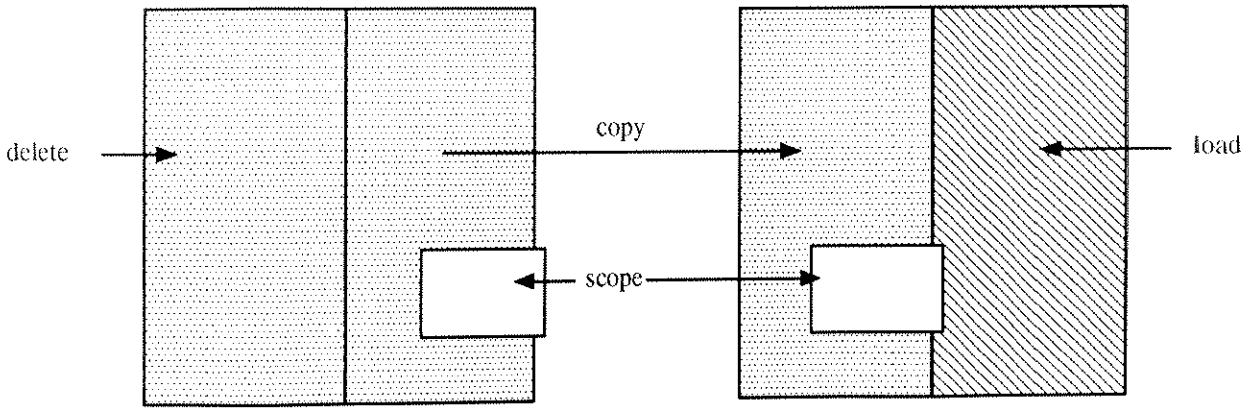

図 7.12: データのスワップ

最近使ったメッシュデータをキャッシュに保存しておき、新たにメッシュのデータが必要になった時には、まずキャッシュを見に行くようにした。キャッシュに該当するメッシュデータがあれば直ちにロードされ、キャッシュにないときに始めてディスクから読み出すようした。ユーザによる地図のスクロールでは、何度か同じ範囲を通過する傾向にあり、このキャッシュ機能で、ファイルアクセスの頻度を落すことができた。

具体的には、メッシュのロードが必要になると、キャッシュに対して`get(id)` を呼ぶ。id に指定したメッシュが存在すれば、そのメッシュのデータが返る。なければディスクからロードする。スクロールして画面からはみ出た部分は直ちに破棄しないで、キャッシュに `put(id)` する。`put()` されたメッシュはキャッシュの先頭に入る。このときキャッシュがいっぱいだと、もっとも古くキャッシュに`put()` されたメッシュが削除される。

キャッシュ機能は、Nuts コンポーネントのキャッシュ部品 (NutsCache) を利用した。

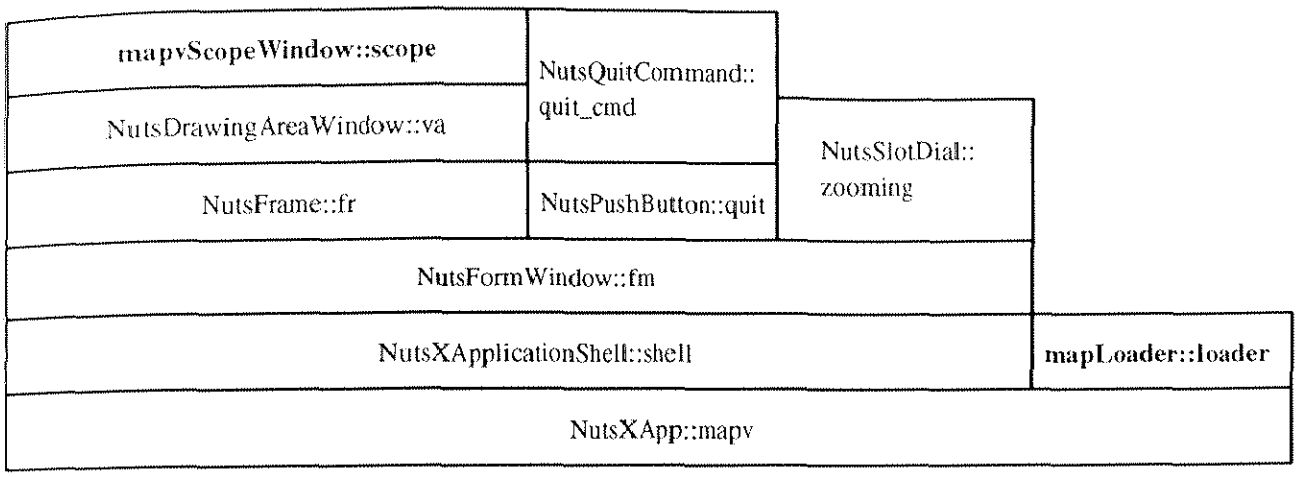

図 7.13: プログラムの全体構成

7.2.4 全体構成

プログラム全体は図 7.13のような構成になっている。構造記述ファイルは以下のようになる。

```
newNutsXtApp(mapv);
newNutsXApplicationShell(mapv,shell);
newNutsFormWindow(shell,fm);
newNutsFrame(fm,fr);
newNutsDrawingAreaWindow(fr,va);
newmapvScopeWindow(va,scope);
newmapLoader(mapv,loader);
newNutsPushButton(fm,quit);
newNutsQuitCommand(quit,quit_cmd);
newNutsSlotDial(fm,zooming);
```

mapvScopeWindow は、NutsScopeWindow のサブクラスで、meshUser オブジェクトを保持し、前述したclipover() などをオーバーライドしている。mapLoader は、実際にファイルから地図データを読み込み、mapGlphy オブジェクトを生成する。今回個別に開発した部品は mapvScopeWindow と mapLoader、mesh などのデータ部品だけである。それ以外の部品は、すべて Nuts コンポーネントを再利用している。

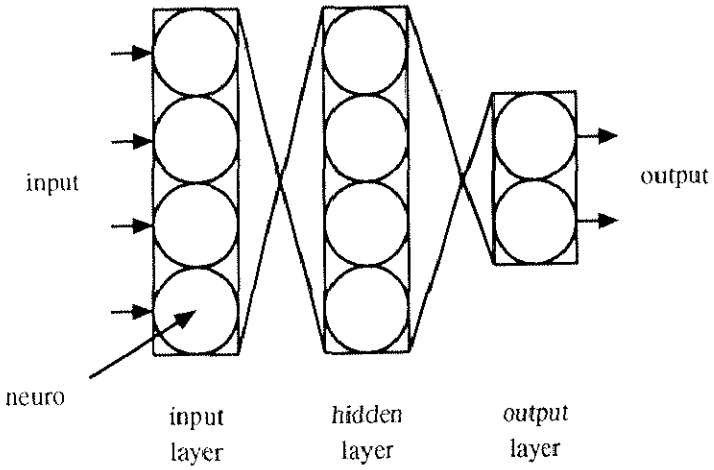

図 7.14: 階層型ニューラルネットワークの構成

7.3 ニューラルネットワークシミュレータ

7.3.1 階層型ニューラルネットワーク

本節では、階層型ニューラルネットワークを Nuts の構造に対応させ、Nuts 部品を用いて、さまざまな構成のニューラルネットワークシミュレータを構築する方法を説明する。

階層型のニューラルネットワークは、図 7.14のように入力層、隠れ層、出力層からなる。階層型ニューラルネットワークにおける代表的学習アルゴリズムであるバックプロパゲーション [32] の流れは以下の通りである。

1. 入力層にあるニューロンが入力信号を受取り、各ニューロン間のリンクに付加された重み付け総和の値によって、隠れ層のニューロンを活性／抑圧する。
2. 隠れ層のニューロンの活性状態は出力層にまで伝搬する。
3. 出力層の出力と出力層に与えられた教師信号との差によって決まる学習値を求める。
4. 出力層から入力層に学習値を逆伝搬し、ニューロン間リンクの重みを更新する。
5. 出力層での教師信号との誤差が、ある値以下になるまでこれを繰り返す。

学習の速度や答えの精度、収束の可否は、以下のパラメータで決まる。

- 隠れ層の段数
- 隠れ層に属するニューロンの数
- ニューロンの種類

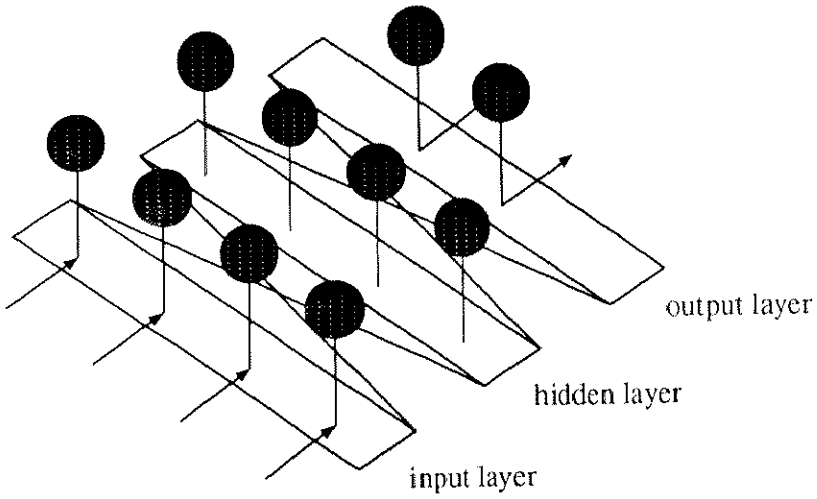

図 7.15: Nuts における階層型ニューラルネットワークのモデル

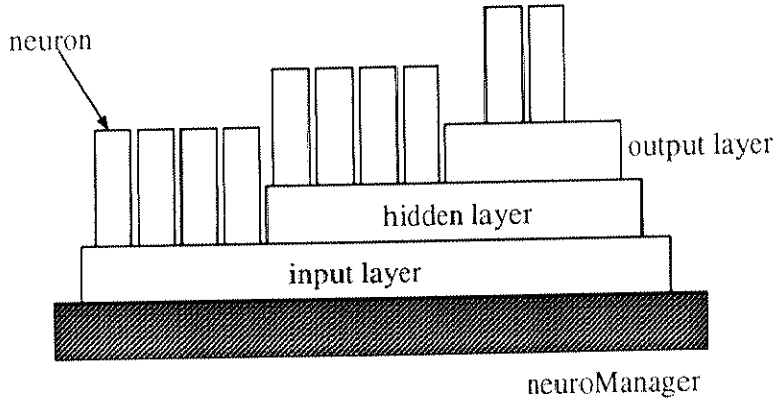

図 7.16: Nuts における階層型ニューラルネットワークの部品構成

ニューラルネットワークによる学習では、これらのパラメータを発見的に変更しながら試行錯誤で最適なネットワークの構成を求める。従って、ニューラルネットワークを計算機の上でシミュレーションするニューラルネットワークシミュレータでは、これらのパラメータが容易に変更できなければならない。

7.3.2 Nuts による階層型ニューラルネットワークシミュレータの開発

Nuts では、コンポーネントの結合関係が容易に変更できるという特徴がある。この特徴を活かして、階層型ニューラルネットワークの構成を Nuts の構造に適合させることができれば、さまざまな階層型ニューラルネットワークの構成を Nuts の構造記述ファイルで容易に変更でき、柔軟なニューラルネットワークシミュレータを開発できる。そこで階層型

表 7.2: ニュラルマネジャの受理するメッセージ

Test neuro all	すべてのテストパターンをテスト
Test neuro (パターン番号)	指定したテストパターンをテスト
Show neuro state	ニューロンの活性化状態の表示
Set neuro autostop (値)	誤差が値より小さくなったとき自動停止
Set neuro configure	ニューロンの活性化関数のパラメータ値をセット
Set neuro times (回数)	Run neuro で行うシミュレーション回数のセット
Run neuro (回数)	シミュレーションの開始、回数指定がないと Set neuro times でセットした値。
Step neuro	Run neuro (1) に同じ

ニューラルネットワークの構成を、図 7.15 のように、入力層、隠れ層、出力層の各層が結合し、それぞれの層の上にニューロンが結合した形とする。ニューロン間のデータ授受は各層を通して行う。この構成を Nuts では、図 7.16のように構築できる。すわなち、入力層の上に複数の隠れ層、隠れ層の上に出力層が結合しており、それぞれの層の上にニューロンが結合している構成になる。階層型ニューラルネットワークの構成をこのようにモデル化することで、シミュレータの構成を Nuts 部品の結合によって容易に変更できる。具体的には、構造記述ファイルの編集により、隠れ層の段数、隠れ層に結合するニューロンの数、使用するニューロンの種類を容易に変更できる。

さらに、ニューラルネットワーク全体の管理を行うニューラルマネジャを導入する。ニューラルマネジャは、以下のような処理を行うグループマネジャとして (NutsNeuroManager クラス) 実装する。

- テストパターン／教師パターンのファイルからの読み込み
- 学習状態の監視／シミュレーション自動停止
- ネットワークへのコマンドの送出 (表 7.2参照)

ニューラルマネジャは、グループマネジャのサブクラスである。ニューラルマネジャは、上記の処理をネットワーク内部で閉じた処理としてニューラルマネジャのグループ中に隠蔽し、ネットワーク外部からネットワーク全体を一つのシミュレーション部品として見せるように振舞う。

ニューラルマネジャが受理するメッセージを表 7.2に示す。また、Nuts のニューロン部品を表 7.3に示す。ニューロンには活性化関数 ($y = f(x)$ x : 入力 y : 出力) の異なる 3 種類の部品がある。

表 7.3: Nuts のニューロン部品

ニューロン名	活性化関数
線形ニューロン	$y = x$
しきい値ニューロン	$y = 0\ (x <= c), y = 1\ (x > c)$ c : しきい値
シグモイドニューロン	$y = -c/(1 + e^{-a(x+x_0)})$ a, c, x_0 : シグモイドパラメータ

表 7.4: 4bits → 2 ビットエンコーダの学習パターン

テストパターン		教師パターン
0 0 0 1	→	0 0
0 0 1 0	→	0 1
0 1 0 0	→	1 0
1 0 0 0	→	1 1

表 7.5: 4bits → 2 ビットエンコーダのテスト結果

テストパターン		出力パターン
0 0 0 1	→	0.044 0.045
0 0 1 0	→	0.048 0.955
0 1 0 0	→	0.963 0.037
1 0 0 0	→	0.960 0.960

7.3.3 ニューラルネットワークシミュレータの構成例

Nuts によるニューラルネットワークシミュレータの構成例として、4bits → 2bits エンコーダについて説明する。このエンコーダのテストパターン及び、教師パターンを表 7.4 に示す。4bits → 2bits エンコーダを学習するニューラルネットワークの部品構成を図 7.17に、構造記述ファイルを図 7.19に、操作パネルのイメージを図 7.18に示す。本シミュレータでは、ニューラルネットワークを表 7.2に挙げたメッセージ群で作動させるためのメッセージ入力部 (図中 sw)、およびメッセージの解釈部 (図中 INTERPRETER)、結果の出力部 (図中 STDOUT) を設けた。例えば、入力部に “manager:Run neuro”(manager 部品に Run neuro メッセージを送るという意味) とタイプすることで、本シミュレータはシミュレーションを開始する。

4bits → 2bits エンコーダで 500 回学習を行った結果を表 7.5に示す。出力ニューロンにしきい値 0.5 のしきい値ニューロンを使用すれば、出力を表 7.4の正しいパターンに分離でき

表 7.6: Nuts を用いた開発例

アプリケーション名	行数	クラス数	開発工数 (人・月)
app1	4898	37	3.0
app2	7645	68	3.0
app3	1958	20	1.0
app4	846	9	0.1 以下
app5	465	5	0.1 以下
Nuts/Builder	6749	57	1.0
DRMA 地図ビュアー	1940	3	0.3
NN シミュレータ	1603	19	0.5

ることがわかる。

Nuts を用いれば、図 7.17、図 7.19のように、ニューラルネットワークの構成が Nuts 部品の構成として容易に把握できる。また、構造記述ファイルの編集で容易にニューラルネットワークの構成を変更できる。例えば、2bits → 4bits デコーダに変更したい場合は、構造記述ファイルを図 7.20のように書き直せば良い。

7.4 ビジネスアプリケーション開発事例

Nuts の適用事例として、ビジネスアプリケーションの開発がある。このアプリケーションは、企業のマーケティング部門で用いる意志決定支援のためのツール群である。具体的には、商品の価値を入力するエディタ、商品の価値を比較するツール、商品価値を管理するエディタなどからなる商品価値比較検討システムである。そのうちの一つの画面イメージを図 7.22に示す。

このシステムは、5 つのサブシステムからなり、それらを Nuts フレームワークを用いて開発した。その際、まずシステム全体に共通の機能をこの場合のアプリケーションフレームワークとして構築し、その後にそれぞれのサブシステムで必要な部品群を個別に開発して組み込んだ。それぞれのアプリケーションのコード量、及び固有に開発した部品数を表 7.6に示す。

部品種別の平均コード比率を図 7.21に示す。図中、フレームワーク部品とは、本アプリケーションで共通に用いたフレームワーク構築のために開発した部品群である。また部品群とは、それぞれのサブシステム固有の部品群を指す。このグラフからわかるように、全コード量の 1 割が本アプリケーションのフレームワークとして再利用可能であった。また、全コード量のおよそ 7 割が Nuts コンポーネントの再利用によりまかなわれた。おおまかに言って開発効率は Nuts コンポーネントの再利用により 3 倍になったと言える。

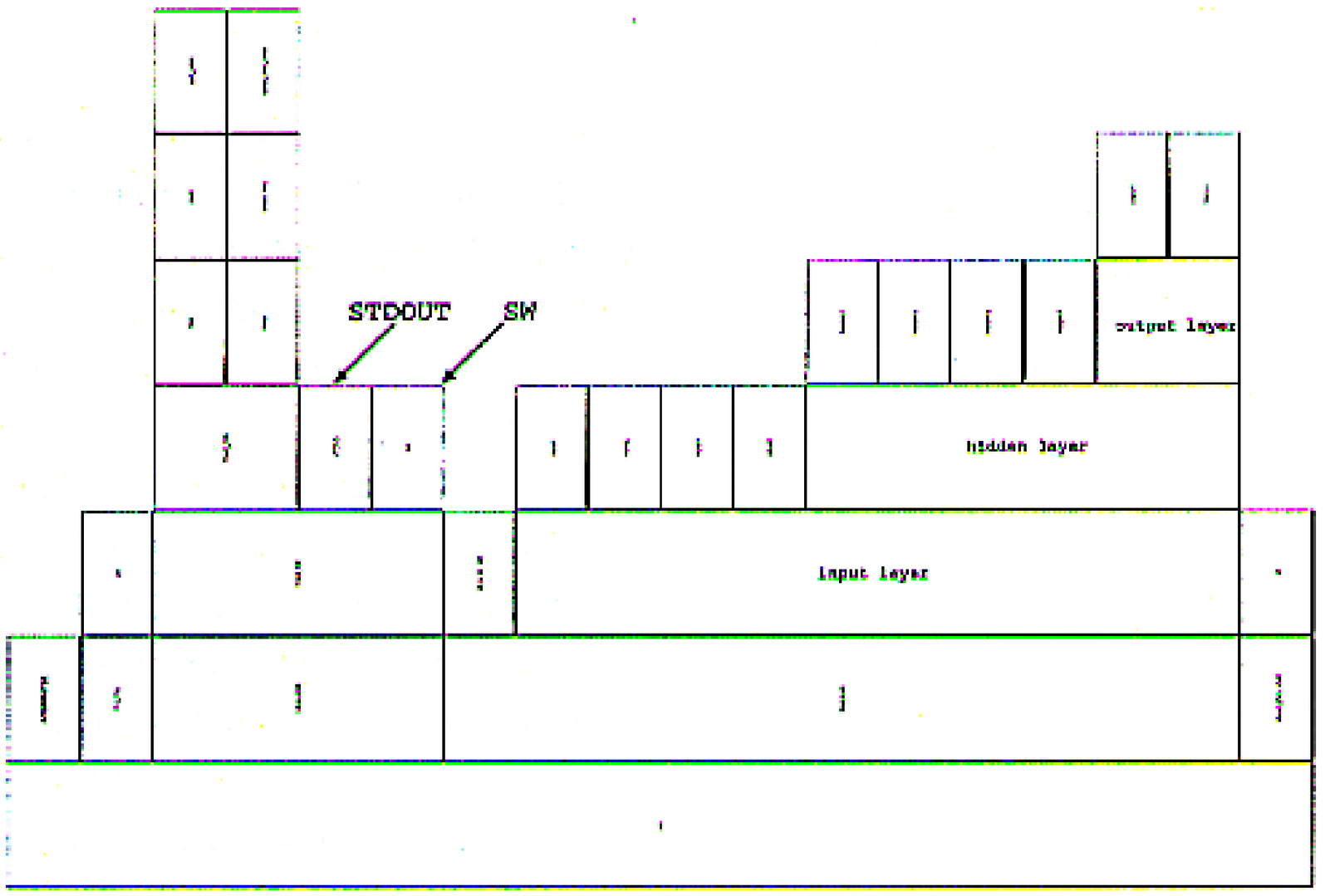

図 7.17: 4bits → 2bits エンコーダの部品構成

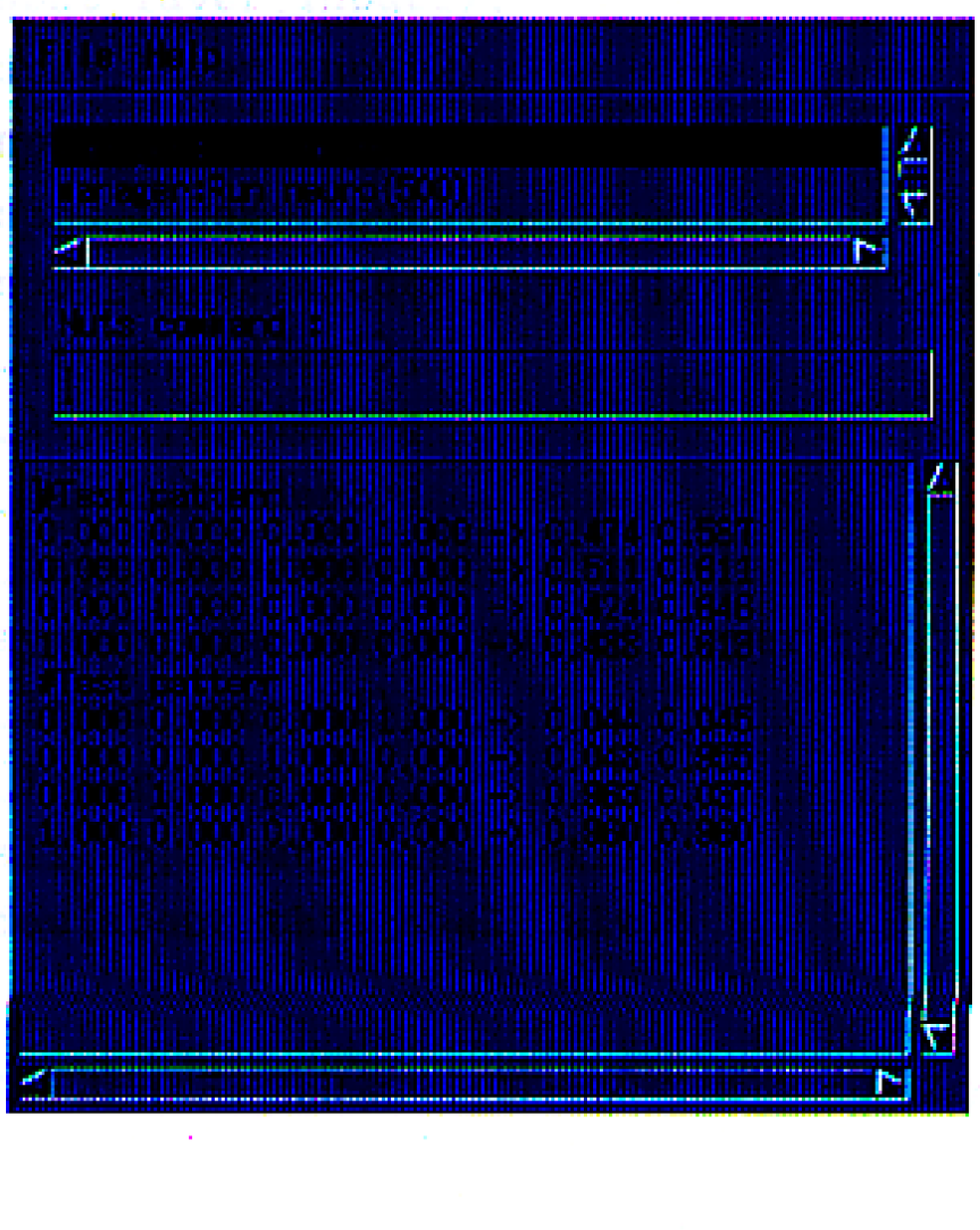

図 7.18: 4bits → 2bits エンコーダの操作パネル

```
newNutsXtApp(bp);
newNutsInterpreter(bp,INTERPRETER);
newNutsXApplicationShell(bp,NeuroSim);
newNutsFormWindow(NeuroSim,main_fm);
newNutsMenuBar(main_fm,main_mb);
newNutsPulldownMenu(main_mb,File);
newNutsPushButton(File,Quit);
newNutsQuitCommand(Quit,quit_cmd);
newNutsPulldownMenu(main_mb,Help);
newNutsPushButton(Help,about_this);
newNutsAboutThisCommand(about_this,about_this_cmd);
newNutsScrolledText(main_fm,STDIO);
newNutsShellWindow(main_fm,sw);
newNutsNeuroManager(bp,manager);
newNutsDataFile(manager,encode_dat);
newNutsInputNeuroLayer(manager,input_layer);//入力層の定義
newNutsLinerNeuro(input_layer,input1);
newNutsLinerNeuro(input_layer,input2);
newNutsLinerNeuro(input_layer,input3);
newNutsLinerNeuro(input_layer,input4);
newNutsNeuroLayer(input_layer,hidden_layer);//隠れ層の定義
newNutsNeuro(hidden_layer,hidden1);
newNutsNeuro(hidden_layer,hidden2);
newNutsNeuro(hidden_layer,hidden3);
newNutsNeuro(hidden_layer,hidden4);
newNutsOutputNeuroLayer(hidden_layer,output_layer);//出力層の定義
newNutsNeuro(output_layer,output1);
newNutsNeuro(output_layer,output2);
newNutsXTransientShell(bp,about_this_shell);
newNutsAboutThisWindow(about_this_shell,alt);
```

図 7.19: 4bits → 2bits エンコーダの構造記述ファイル

```
newNutsXtApp(bp);
newNutsInterpreter(bp,INTERPRETER);
newNutsXApplicationShell(bp,NeuroSim);
newNutsFormWindow(NeuroSim,main_fm);
newNutsMenuBar(main_fm,main_mb);
newNutsPulldownMenu(main_mb,File);
newNutsPushButton(File,Quit);
newNutsQuitCommand(Quit,quit_cmd);
newNutsPulldownMenu(main_mb,Help);
newNutsPushButton(Help,about_this);
newNutsAboutThisCommand(about_this,about_this_cmd);
newNutsScrolledText(main_fm,STDIO);
newNutsShellWindow(main_fm,sw);
newNutsNeuroManager(bp,manager);
newNutsDataFile(manager,decode_dat);
newNutsInputNeuroLayer(manager,input_layer);//入力層の定義
newNutsLinerNeuro(input_layer,input1);//入力層のニューロン数を2つに変更
newNutsLinerNeuro(input_layer,input2);
newNutsNeuroLayer(input_layer,hidden_layer);//隠れ層の定義
newNutsNeuro(hidden_layer,hidden1);
newNutsNeuro(hidden_layer,hidden2);
newNutsNeuro(hidden_layer,hidden3);
newNutsNeuro(hidden_layer,hidden4);
newNutsOutputNeuroLayer(hidden_layer,output_layer);//出力層の定義
newNutsNeuro(output_layer,output1);//出力層のニューロン数を4つに変更
newNutsNeuro(output_layer,output2);
newNutsNeuro(output_layer,output3);
newNutsNeuro(output_layer,output4);
newNutsXTransientShell(bp,about_this_shell);
newNutsAboutThisWindow(about_this_shell,alt);
```

図 7.20: 2bits → 4bits デコーダの構造記述ファイル

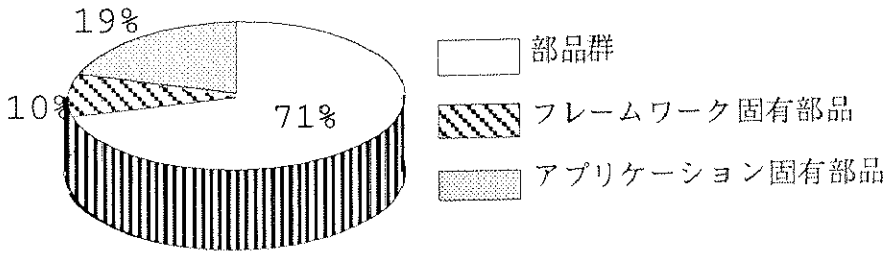

図 7.21: Nuts ライブラリを用いたコード比率

7.5 考察

本章では、Nuts コンポーネントを用いたプログラム開発の事例について検討した。これらの開発事例では、Nuts コンポーネントがホワイトボックス的な柔軟性を持っていることから、サブクラス拡張によってさまざまな分野の個別要求に対応するコンポーネントを迅速に開発できた。

また、構造や制御に統一したモデルを採用し、特に構造については構造記述ファイルで一元的に管理しているため、メンテナンスが容易である。

さらに、コンポーネント差分プログラミングのために開発した、様々なベクターコンポーネントは、アプリケーションの枠を越えて再利用可能なものであり、ベクターコンポーネントを用いたプログラミング効率の高さを示している。また、コンポーネント差分プログラミングにおける拡張機能追加の簡便さ (ベクターコンポーネントをプログラム中に存在するコンポーネントに結合させるだけでよい) が、ベクターコンポーネントおよび被修飾コンポーネントの再利用の促進に結びついていると考えられる。

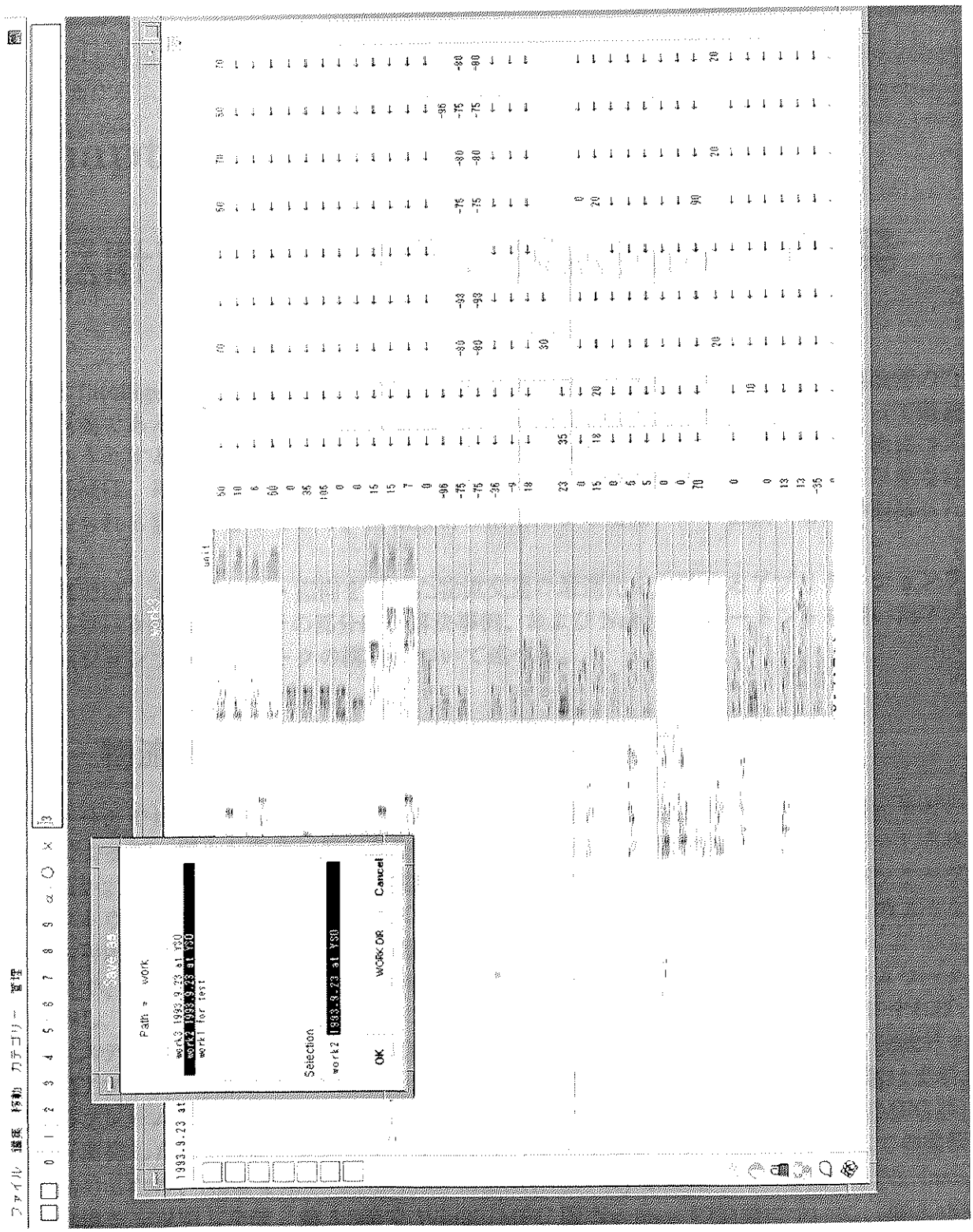

図 7.22: ビジネスアプリケーションの一つの画面イメージ (秘匿義務のため画面の一部をぼかしている)