Mochi Sheet：大規模なビジュアルプログラムの効率的編集を支援するズーミングインタフェース

豊田正史† 高橋伸† 柴山悦哉†

本稿では大規模なビジュアルプログラム（VP）の効率的な編集を支援する複数フォーカスのズーミングインタフェースMochi Sheetを提案する．VPの効率的な編集を行うには，ユーザが，編集に必要なレイアウトを少ない手数で得られること，および次の編集に必要なレイアウトへすばやく移行できることが必要となる．Mochi Sheetは入れ子状の階層構造図式をフィッシュアイビューを用いて表示し，ズーミングのアニメーションを行う手法を基にしている．この手法を用いたエディタにおいて効率的なレイアウト編集を支援するためにMochi Sheetでは，複数のフォーカスの同時拡大，ズーム操作のアンドゥおよび選択的なアンドゥ，および図形の重なりの自動回避の各機能を提供している．我々はこのインタフェースを汎用のズーミングライブラリとしてAmulet GUIツールキット上に実装した．さらにMochi Sheetを用いてビジュアルプログラムのエディタを実装しその有効性を確認した．

Mochi Sheet: A Zooming Interface which Supports Efficient Editing of Large Visual Programs

MASASHI TOYODA,† SHIN TAKAHASHI† and ETSUYA SHIBAYAMA†

Mochi Sheet is a multi-focus zooming interface that supports efficient editing of large visual programs. It allows the user to obtain easily a desired layout for editing and to make a smooth transition to another layout for editing. Mochi Sheet is based on a fisheye-like zooming interface that displays multiple focuses and context of a hierarchical network in a single view and supports animation of zooming. To support efficient editing in the interface, we provide three features as follows: simultaneous zooming of multiple focuses, undo and selective undo of zooming, and automatic overlap avoidance of graphical objects during editing. We implemented Mochi Sheet as a library for AMULET GUI Toolkit. Using Mochi Sheet, we implemented a visual programming editor, and observed that users can edit visual programs efficiently.

1. はじめに

ビジュアルプログラミング環境（VPE）には，エンドユーザからプログラマまで様々なユーザを対象としたものが存在する．中でもプログラマ向けのVPEはプログラムの可読性および保守性を向上させる手法として有効である．プログラマ向けのVPEにはデータフローグラフ，コントロールフローグラフなどグラフ図式を用いたビジュアルプログラム（VP）を扱うものが多く，本稿でもそのようなVPを取り扱う．

VPを表現するグラフ図式の編集には，テキストと比べて比較的大きなスクリーンスペースが必要となるため，図式が大規模になるにつれてその編集作業は難しくなる．大規模なVPの扱いは，実用的なプログラミングを行う場合には非常に重要である．ところが大規模なVPを容易に編集できるVPEは少なく，スケーラビリティに関してはまだ問題が残されている．

大規模なVPの編集作業の難しさは，表示および編集操作に関する難しさに分類することができる．表示に関しては既存のfisheye viewを基にした複数フォーカスのズーミングの手法[1),2)]が応用できる．しかしこれらはトポロジおよびレイアウトが静的なグラフを対象としているため，そのままでは編集操作の支援に関して問題点が残されることになる．

我々は，この問題を解決したうえで，文献2)と同様のなめらかなズーミングを用いたエディタにおいて効率的な編集作業が行えるユーザインタフェースライブラリMochi Sheetを構築した．さらにMochi Sheetを用いてKLIEG VPE[3)]のエディタを実装し有効性

† 東京工業大学情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

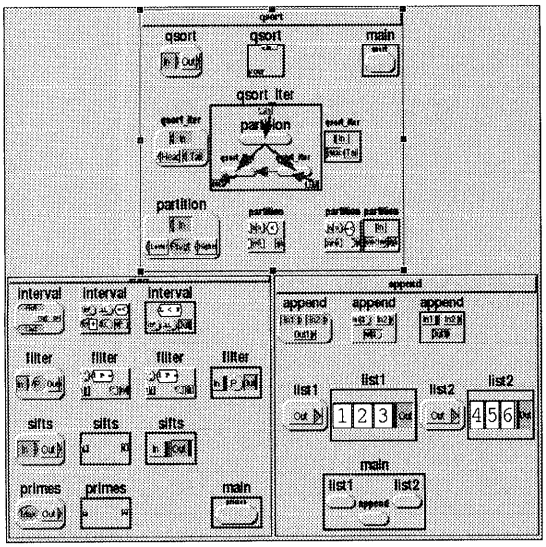


図1 KLIEG エディタの全体図
Fig. 1 Snapshot of KLIEG editor.

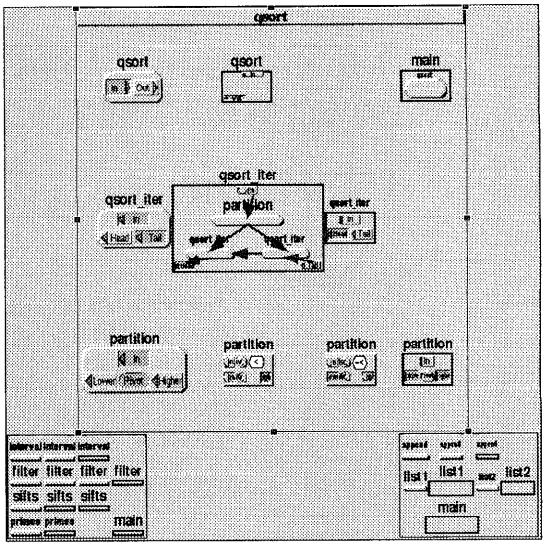


図2 モジュールを拡大
Fig. 2 Magnifying a module.

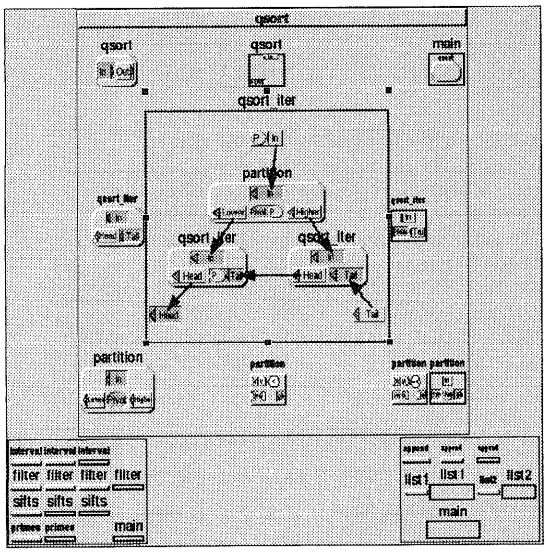


図3 ネットワークを拡大
Fig. 3 Magnifying a network diagram.

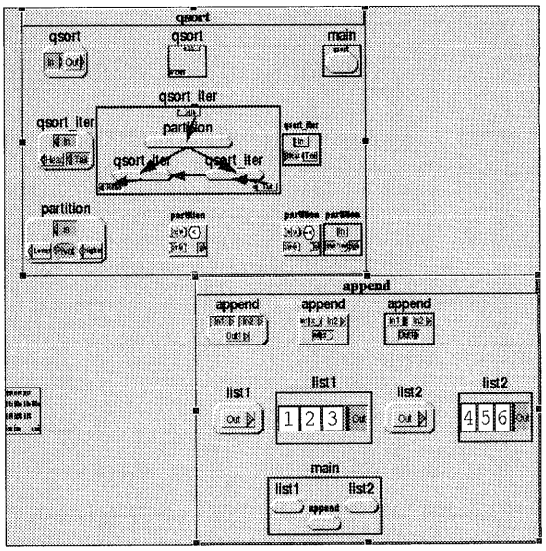


図4 複数箇所を拡大
Fig. 4 Magnifying multiple parts.

を確認した．また Mochi Sheet は汎用のライブラリとして実装されているため，VP のみならずファイルブラウザなど，他の様々なアプリケーションに対して応用することも可能である．

ここでズーミングを用いたエディタの具体的な挙動を示すため，**図1**～**図4** に KLIEG エディタの画面スナップショットを示す．KLIEG はプロセスとその間のデータフローに基づく VPE である．エディタでは，プロセスのネットワーク構造，および関連するプロセス定義をまとめたモジュールの階層構造を表示し，編集を可能にする．エディタでは，1つのウインドウの中で編集したい部分を適宜拡大して編集を行うことができる．図1では3つのモジュールが表示されており，図2，図3では1つのモジュールを拡大し，さらにその中のネットワーク図を拡大している様子を示している．拡大したネットワーク図は詳細が見えるようになり，周りの図はそれに応じて縮小され，詳細が見えなくなっている様子が分かる．また拡大後も図形の相対位置は保たれている．また，図4のように2つのモジュールを同時に拡大して，プロセスをドラッグ&ド

ロップするという編集操作も容易に行うことができる．

以下，2章ではまず大規模なVP編集作業の効率化における問題点とその解決方法について述べる．3章ではMochi Sheetのユーザインタフェース，およびズームのアルゴリズムについて解説する．4章で実装について述べ，5章では関連研究との比較を行う．6章でまとめと今後の課題を述べる．

2. 編集作業の効率化

ここではまずグラフ表現を扱うVP（以下，単にVPと記す）の表示に関する問題点を示し，それをズーミングによって解決できることを示す．さらにズーミングを導入したエディタにおける編集操作に関する問題点およびその解決方法について述べる．

VPの表示は，その構造に適合した図式で行うことが重要である．大規模なVPはプログラム構成要素を階層的に組み合わせた構造をなす．プログラムは複数のモジュールから構成され，モジュールは複数のグラフから構成され，グラフは複数のサブグラフから構成される．この結果，プログラム全体は大規模な階層構造をなすことになる．

大規模な階層構造の表示にはスクリーンの狭さが問題となる．編集を行うには編集する部分を詳細に見る必要がある．しかし，一度に詳細に見ることできる部分はスクリーンの狭さから数カ所に限られる．さらに階層的なグラフ表現を扱うため，編集している部分がグラフ全体のどの部分なのか，といった情報が重要となる．こういった情報も同時にスクリーンに表示しなくてはならない．

また2カ所以上を同時に参照しながら編集を行う場合が多い．あるモジュールで定義されている部品を他のモジュールで利用する際には，モジュール・グラフ間で部品をドラッグ&ドロップする操作がよく用いられる．このため複数のモジュールの詳細を同時に参照することがしばしば必要となる．

したがって，大規模な図形の階層構造における全体の概要，複数の編集部分の詳細，およびその間の関連をどのように表示するかという点が問題となる．

この表示の問題に関してはズーミングインタフェースが有効である．ズーミングインタフェースには大きく分けて2種類の代表的な手法が存在する．Pad++[4]のように単純なパンとフルズーミングを行うインタフェースを用いるもの，およびfisheye view[5]を代表とする一部分のみを拡大した歪んだビューを提供するものである．

これらの2種類のズーミング手法のうちVPの表示に有効なのはfisheye viewの手法である．Fisheye viewではデータ全体の概観表示および一部の詳細表示を同じビューで行うため，プログラム全体の構造を見失うことなく一部分の詳細を参照・編集することが可能である．フルズーミングの手法では一部分を拡大するとその周辺が見えなくなってしまうため上記の要望を満たすことができない．さらにSchafferらは階層的なネットワーク構造のナビゲート作業にはfisheye viewのインタフェースがフルズームより効率的であるというユーザテストの結果を発表している[6]．

Fisheye view手法から派生したRubber Sheet[1]，Continuous Zoom[2]は，2次元平面上の入れ子状に構成された図形の階層構造に対して，(a) 図形の相対位置を保ち，(b) 図形の重なりを避けながら，(c) 複数のフォーカスを指定したズームを行うという点で我々の目的に適合している．特に文献2)はズームによる変化をなめらかにアニメーション表示するため，視点の変化を把握しやすいという特徴を持っている．

本研究の目的は複数フォーカスのfisheye viewを導入したエディタにおいて，ユーザが少ない手間で効率的に編集を行えるインタフェースを提案することである．単に文献1)，2)のズーミング手法を用いただけでは効率的な編集を支援するには不十分である．文献1)，2)ではナビゲーションのインタフェースに重点が置かれており，編集用のインタフェースが不足しているためである．

効率的な編集を支援するためには，階層構造図式の編集におけるサイクルについて考慮しなければならない．大規模な階層構造図式の編集作業は，通常以下に示すサイクルを繰り返すことによって行われる．

(1) 編集したい図形を（1つまたは複数）拡大する．
(2) 拡大した図形の内部において図形の追加，削除，移動等の編集を行う．また拡大した図形間での図形の移動・コピーなどを行う．
(3) 拡大した図形を元の大きさに戻す．

この編集サイクルのうち本質的に時間を必要とする部分は(2)である．したがって，効率的な編集を支援するためには，(1)，(3)においてユーザが，編集に必要なレイアウトを少ない手数ですばやく得られるインタフェースを提供することが重要となる．これは(2)に使える時間を増やすことが編集サイクルの効率化につながるからである．また(2)においても必要なレイアウトを少ない手間で得られるようにすることで編集操作を効率化することができる．以下にサイクルにおける各項目に対応する問題点を列挙する．

複数のフォーカスへの対応 前述したように，複数箇

所を同時に参照しながら編集作業を行う機会は多い．文献1)，2)には複数箇所を同時に拡大するインタフェースがないため，対象となる部分を1つ1つ拡大しなくてはならない．さらに一度拡大した部分が，他の部分の拡大により縮小されてしまうこともあるため，拡大のやり直しが必要な場合もある．これは手間のかかる煩わしい作業である．

編集による図形の重なりの回避 VPにおけるグラフのノードには意味情報が含まれるため，ノードが重なるとプログラムの意味が理解できなくなることが多い☆．しかし，ズームによって歪んだレイアウトにおいて，手動でノードの重なりがないように編集を行うのは手間のかかる難しい作業である．

歪んだレイアウトを元に戻す方法 編集サイクルにおける(3)の元に戻す手順においては，ズーム操作で戻すのが一番簡単に思いつく方法である．しかしズーム操作で元の大きさに正確にすばやく戻すことは難しい．

以上の問題点を踏まえて，我々はMochi Sheetライブラリを設計・実装した．Mochi Sheetは文献2)と同様のなめらかなズーミングを提供するライブラリである．Mochi SheetはVPのエディタの他にもファイルシステムブラウザなど様々なアプリケーションに応用できる．Mochi Sheetは，上記の問題点を以下のように解決している．

- 複数の図形を選択して同時に拡大・縮小できるインタフェースを導入する．ユーザは少ない手間で，ほぼ望みのレイアウトを得ることができる．
- ズーム中および編集中に，図形が重ならないように☆☆自動的に再配置を行うことで，歪んだレイアウトにおいても容易に編集を行うことができる．再配置のために縦横の制約線上に図形を並べる制約を導入した．この制約により，空いている空間を利用した再配置が容易になる．
- ある図形におけるズーム操作のみを選択的にアンドゥするインタフェースを提供する．この機能により，拡大した編集部分を簡単にすばやく元の大きさに戻すことができる．また図形を編集履歴中における好みの大きさに1回の操作で戻すことが可能になる．

3. Mochi Sheet

この章では，Mochi Sheetのユーザインタフェース，およびズーム・レイアウトのアルゴリズムに関して説明する．Mochi Sheetは，矩形の領域を持つ図形☆☆☆を扱う．矩形の中に子供にあたる矩形を入れ子状に配置することにより，階層的な木構造を構成できる．以後，木構造の節および葉にあたる矩形をノードと呼ぶ．また，内部に子ノードを持つノードをクラスタノードと呼んで区別する場合がある．

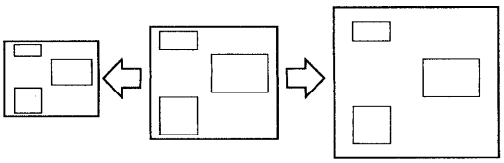


図5 子ノードの拡大を行わないズーム
Fig. 5 No magnification on children nodes.

3.1 インタフェースと挙動

Mochi Sheetが扱う編集操作は，ズーミング，ノードの追加・削除・移動，ズームのアンドゥの3種類である．ただし，図形のレイアウトに影響をあたえない編集操作についてはここでは扱わない．本節ではそれぞれの操作に対するMochi Sheetのインタフェースについて述べる．

3.1.1 ズーミング

ズームは，選択したノードの8方向につけられるハンドルで行う．ハンドルをつかんで動かしている間，ノードとその周辺のノードが重ならないようになめらかに変化する．Mochi Sheetはレイアウトの編集が主な目的であるため，個々の図形の大きさを細かく指定できるハンドルインタフェースを採用している．

複数のノードを同時にズームするには，複数のノードを選択したうえで，選択したノードのうちの1つに対してズーム操作を行えばよい．選択された各ノードはズーム操作を行っているノードと同じ大きさになる．これを利用して複数のノードの大きさを揃えることも可能である．

クラスタノードを小さくした結果，子ノードが元の大きさで表示できなくなったときに，その内部に対してズームアウト（縮小）が行われる．逆に大きくした場合，プログラマはズームイン（拡大）を行うか，行わないかをクラスタノードに対して指定することができる．ズームインを行わない場合，子ノードは元の大きさのままで，各ノードに均等に空間が分配される．これらは用途によって使い分けられる（図5）．

☆ ノードの重ね合わせに意味のあるVPも存在する．
☆☆ グラフにおける辺の重なりの回避は本稿では扱わない．
☆☆☆ 輪郭が矩形である必要はない．

3.1.2 ノードの追加，削除，移動

ノードの追加，削除，移動は，通常の図形エディタと同様の操作で行うことができる．これらの操作が行われるとすべての図形が重ならないように再配置が行われる．移動に関してはクラスタノード間にまたがる移動も行うことができる．ズーム中における図形の再配置のアルゴリズムに関する詳細は3.2節で述べる．

3.1.3 ズームのアンドゥ

ズームのアンドゥは，対象とするノードに対して行ったズーム操作のみのアンドゥを行う．たとえば，あるノードをズームした後移動して，アンドゥを行った場合，図形は元の大きさには戻るが，元の場所には戻らない．このアンドゥ機能を用いることで編集のために拡大したノードを簡単にすばやく元の大きさに戻すことができる．またリドゥを行うことで次回同じ部分を編集する際にはすばやく拡大することもできる．このためアンドゥ・リドゥの操作はノードのアイコン化・非アイコン化と同等の操作と考えることもできる．また元の大きさに戻す際にアニメーション表示を行うことで，レイアウトが大幅に変化する場合にもユーザは編集している部分を見失わずにすむ．

アンドゥはノードのポップアップメニューから行う（図6）．メニューは一般的なアンドゥと同様に，アンドゥ（Undo Zooming），リドゥ（Redo Zooming）の項目を持つ．さらに次に述べる選択的アンドゥのためのダイアログを呼び出す項目（Selective Undo Zooming...）も持っている．また，ポップアップメニューにはアプリケーションに特有のメニュー項目を容易に付け足すことができる．図6ではアンドゥ用のメニュー項目の下にKLIEGエディタにおける編集操作のためのメニュー項目が付け加えられている．

何度かズーム操作を行ったノードを一番最初の大きさに戻したい場合，アンドゥ・リドゥのみでは不十分である．ユーザはアンドゥ操作を何回も繰り返さなければならない．選択的なアンドゥは，直前のズームだけではなく，それ以前のズーム操作を選択的にアンドゥできる機能である．これを用いるとノードをある時点での大きさに一度の操作で戻すことが可能になる．選択的アンドゥはズーム操作の履歴を示すダイアログボックスから，大きさを選択することで行う（図7）．ダイアログボックスにはアンドゥの履歴として，幅，高さを表す数値のペアが列挙される．現在の大きさを表すペアには「*」が付けられている．ユーザがペアを選択するとエディタ上での大きさが実際に変化するので，大きさを決定する前にレイアウトがどのように変化するかをあらかじめ確認することができる．

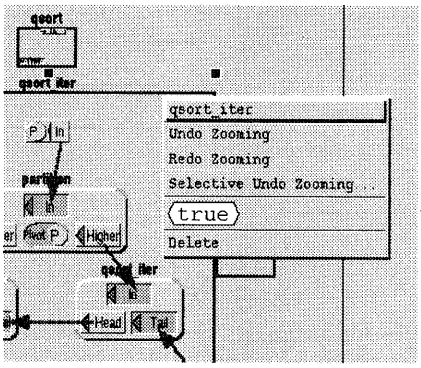


図6 ズームのアンドゥメニュー
Fig. 6 An undo menu of zooming.

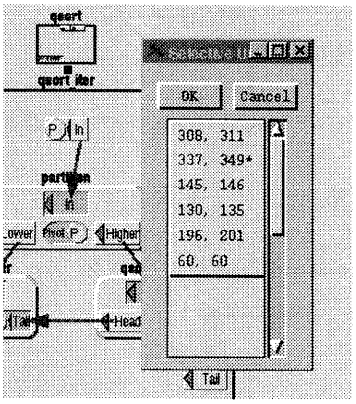


図7 選択的アンドゥのダイアログボックス
Fig. 7 A selective undo dialog box.

アンドゥの履歴の順序はある程度ユーザが変更することが可能である．これは選択的なアンドゥを実行すると，選択した大きさがアンドゥの履歴に加えられるためである．これを利用すると，たとえば，ノードの現在の大きさと一番最初の時点での大きさとをアンドゥ・リドゥ操作のみを用いて切り替えられるようにすることができる．ユーザは一番最初の大きさへの選択的アンドゥを行うだけでよい．

またアンドゥの代替案としては，図形のスケールを何段階かに変更できるようにするインタフェースも考えられる．しかし決まった大きさにしか変更が行えないため，編集には不向きである．

3.2 ズーム・レイアウトアルゴリズム

Mochi Sheetでは，ノードの追加・移動の後もノードが重ならないようにするため，子ノードを，動的に生成した縦横の制約線の上に並べるという制約を導入している．この制約は比較的きついものであるが，細かいレイアウトを必要としないVPEのようなアプリケーションには十分である．この制約線を用いて各

ノードが重ならないようにズームを行う．また制約線を利用して，空いているスペースを探しノードに割り振ることも行う．以下では，まず基本的なズーム・レイアウトのアルゴリズムを述べ，続いて空きスペースを探し有効利用する方法について述べる．

3.2.1 基本アルゴリズム

基本アルゴリズムは，1つのクラスタノードの子ノードのレイアウトを決定する．このアルゴリズムは対象とするクラスタノードの寸法，および，各子ノードの座標および寸法を入力とし，各子ノードを実際に表示する際の座標および寸法を出力する．階層構造に対しては，このアルゴリズムをルートのクラスタノードから再帰的に適用することで全体のレイアウトを定められる．以下に実行手順を示す．

(1) 縦横軸の制約線の生成 x，y 各軸についてノードの中心の座標を比較し，しきい値以下の距離のものは同じ制約線上に置くように縦横の制約線を生成する☆（**図8**）．

(2) 要求幅の計算 ノードが重ならないために必要な各区間（x1〜x3，y1〜y4）の要求幅を計算する．ある x 軸上の区間 xk の要求幅 R_{xk} は以下の式で表される．

$$R_{xk} = \max\left(\frac{W_{(k-1,j)}}{2}\right) + \max\left(\frac{W_{(k,j)}}{2}\right)$$

ただし $W_{(i,j)}$ は，図8の右図における (i,j) の示す交点の要求する幅を表す．交点の要求する幅は，交点に矩形がのっているときはその矩形の幅，のっていないときは 0 となる．たとえば，x2 の要求幅は r2 と r3 の幅を 1/2 したものの和のとなる．y 軸上の区間の要求幅は矩形の高さについて同様の計算を行うことで求める．

(3) ノードの配置 x，y 各軸について要求幅の和 R_x，R_y を計算する．R_x，R_y がクラスタノードの幅 C_x，C_y より大きい場合，x，y 各軸について C_x/R_x，C_y/R_y というスケールで各子ノードの大きさを縮小する（**図9**）．要求幅の和がクラスタノードの幅より小さく，拡大を行う指定の場合，同じスケールで拡大を行う．拡大を行わない指定の場合，区間の幅をできるかぎり平均化するように余りのスペースを割り振る（図8右）．

ノードの追加・削除・移動が行われた場合，(1) の制約線の生成からやり直す．ズームが行われた場合には (2) からやり直すだけですむため，なめらかにズームを行うことができる．

☆ ノードが同じ交点にのらないように前処理が行われていることを仮定する．

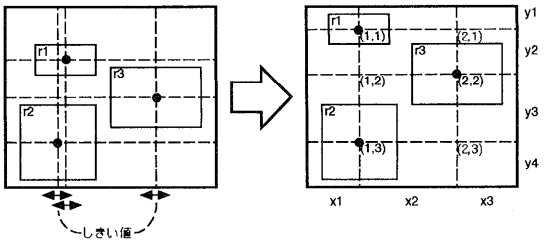


図8 制約線生成とノードの配置
Fig. 8 Generating alignment lines and layouting nodes.

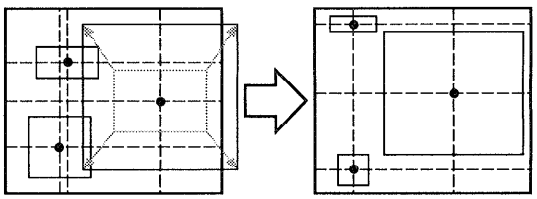


図9 要求幅が大きくなった場合の配置
Fig. 9 A layout when the required width is increased.

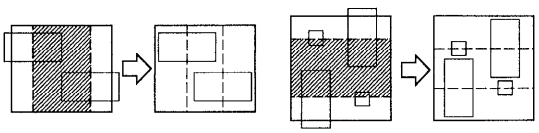


図10 空きスペースの割振り
Fig. 10 Using free spaces.

3.2.2 空きスペースの割振り

制約線にノードをのせるという制約を導入したことで，空きスペースを有効にノードに割り当てることが可能である．スペースの割振りは要求幅がクラスタノードの幅より大きいときに行う．たとえば，**図10**に示した2つの例では斜線で示した区間の幅を狭めてスペースを節約し，ノードの元の大きさを保つことができる．ただしノードの中心点の相対位置は保存するようにする．Continuous Zoom では，ノードの中心点ではなく頂点の相対位置を保つためこのような空きスペースを利用することができない．

スペースを割り振るときには，x，y 軸それぞれに対して各区間を可能な限り狭めた幅（狭め幅）を計算する．ある x 軸上の区間 xk の狭め幅 S_{xk} は以下の式で表される．

$$S_{xk} = \max\left(\frac{W_{(k-1,j)}}{2} + \frac{W_{(k,j)}}{2}\right)$$

要求幅と狭め幅との差（xk の場合 $R_{xk} - S_{xk}$）が大きいものから区間を狭めていく．このとき x 軸，y 軸の区別はせずに順番を決める．ただし x，y 軸にま

たがる区間（x2とy3など）を比較する際は，スケール（3.2.1項の手順(3)で計算した値）のより小さい軸の方を優先する．すなわち，より縮小されている軸の幅を優先的に狭める．たとえば，図8の右の図では，x2，y3，y2の順に狭めていく．ある区間を狭めると他の区間が狭められなくなる場合があるが，このときは狭められなくなった区間を無視して残りの区間を狭める．

4. 実　　装

Mochi SheetはAmulet Version 2.0[7]上に実装した．Amuletはプロトタイプオブジェクトシステムと制約に基づくGUI開発環境である．Amuletでは，図形オブジェクトの集約を行うgroupオブジェクトが提供されている．Mochi Sheetのクラスタノードは，このgroupオブジェクトを継承して実装したため，Amuletのgroupと同様に扱うことができる．

ズームアルゴリズムではgroupに直接含まれる（1階層下の）図形のみを変更するため，階層構造中に異なるズームアルゴリズムを持つgroupを加えることが可能である．これにより文献4)において述べられているsemantic zoomと同等の機能を実現しやすくなっている．たとえば，1章で示したエディタではモジュール，プロセスなどは，どのような大きさになっても名前だけは表示するというgroupで実装されており，その内部にまたMochi Sheetのgroupを含んでいる．

Mochi Sheetは汎用のライブラリとして実装されているので，他の様々なアプリケーションに応用することができる．たとえば，ディレクトリの階層構造を表示するファイルシステムのブラウザは容易に実現可能である．また，クラスタノードをOHPシートに見立てたプレゼンテーション作成エディタにも応用が可能である．このエディタではシートの一覧と，1枚のシートの詳細とを同じビューで見ながら編集が行える．

実装は，Sunの Ultra Sparc上で行った．100個程度の図形が画面上に表示されている場合には秒間4フレーム程度でアニメーション表示を行うことができる．これは図形の個数に関してもスピードに関しても実用的には問題ないレベルである．数が200個程度になると秒間2フレーム程度になる．なお，Amuletでは100個の図形を表示するのに約0.2秒かかるため，秒間5フレーム程度が限界である．このため，画面上に表示できる図形の数を100個程度に制限し，フォーカスの部分は他より多くの図形を表示することでフレームレートが極端に落ちないように工夫している．

5. 関連研究

我々の提案したズーミングインタフェースは文献1), 2)を参考にしている．文献1)，2)においては表示およびナビゲーションに重点が置かれており，編集のためのインタフェースをどのように提供するかという点についてはあまり考慮されていない．文献1)では拡大部分の編集が可能であるとされているが，編集後の図形の重なりの回避は行っておらず，文献2)では初期配置はあらかじめ重なりのないものを用意することになっており，レイアウトの編集については考慮されていない．

Mochi Sheetではズーミング用にハンドルインタフェースを採用している．目的がブラウズのみならばこのインタフェースは煩わしいものとなるため，文献2)ではマウスのボタンを押している間，ズームイン・アウトするインタフェースを採用している．しかし，このインタフェースでは縦横比を変更するズームができない．また文献1)では，図形とは独立した領域を指定しそれをラバーバンドとしてズームを行うため，個々の図形の大きさを細かく指定するのが難しい．

Fisheye viewを基にしたズーミングのインタフェースをエディタに取り入れたビジュアルプログラミング言語は現在ほとんど存在しない．Fisheye viewを取り入れたVPEとしてはVIPR[8]が存在するが，フォーカスが1つしか指定できないため，上述した複数の場所を参照しながらの編集作業を行うことができない．またズーミングおよびフォーカス指定のインタフェースがMochi Sheetと比較して煩雑である．

ノードの自動的な再配置に関しては，力学モデルに基づいたアルゴリズムが多数提案されている．中でも頂点の大きさを考慮するグラフの配置アルゴリズム[9]が利用しやすい．しかし再配置の計算を，なめらかなズームを行うのに十分な速度で行うのは難しい．

6. まとめと今後の課題

大規模なビジュアルプログラムを効率的に編集できる複数フォーカスのズーミングインタフェースを提案し，これを汎用ライブラリMochi Sheetとして実装した．ユーザは大規模なビジュアルプログラムの編集を，拡大・編集・アンドゥ操作の繰返しにより効率良く行うことができる．汎用ライブラリとして実装されているためビジュアルプログラムのみならずファイルブラウザなどの様々なアプリケーションに応用することも可能である．Mochi Sheetを用いて実際にVPのエディタを実装しインタフェースの有効性を確かめた．

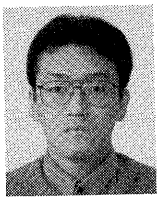
今後の課題としてはノードの自動再配置に力学モデルを応用するなど，より一般的なレイアウト編集を可能にすることがあげられる．さらにユーザインタフェースをより使いやすく改良していくことも必要である．

また，実際に10人の学生にエディタを使用して簡単な作業を行ってもらい，作業の様子を観察した．その結果，ズームのアンドゥを導入したエディタの方が短時間で作業を行えることが分かった．その反面アンドゥのインタフェースには，「実際にアンドゥを行おうと思ったときには，何回ズーム操作を行ったか忘れており，普通のアンドゥと選択的アンドゥのどちらを使うのがよいのか分からなくなってしまう」といった問題点も観察された．このような問題点を解決していくのも今後の課題である．

参考文献

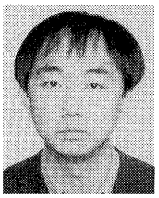
1) Sarkar, M., Snibbe, S.S., Tversky, O.J. and Reiss, S.P.: Streching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens, *Proc. UIST '93*, pp.81–91 (1993).
2) Bartram, L., Ho, A., Dill, J. and Henigman, F.: The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Space, *Proc. UIST '95*, pp.207–215 (1995).
3) Toyoda, M., Shizuki, B., Takahashi, S., Matsuoka, S. and Shibayama, E.: Supporting Design Patterns in a Visual Parallel Data-flow Programming Environment, *Proc. 1997 IEEE Symposium on Visual Languages*, pp.76–83 (1997).
4) Bederson, B.B. and Hollan, J.D.: Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics, *Proc. UIST '94*, pp.17–26 (1994).
5) Furnas, G.W.: Generalized Fisheye Views, *Proc. ACM CHI '86 Conference on Human Factors in Computing Systems*, pp.16–23, Association for Computing Machinery (1986).
6) Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S. and Roseman, M.: Navigating Hierarchically Clustered Networks through Fisheye and Full-Zoom Methods, *ACM Trans. Computer-Human Interaction*, Vol.3, No.2, pp.162–188 (1996).
7) Myers, B.A.: Amulet Project Home Page, http://www. cs.cmu.edu/~amulet/.
8) Citrin, W. and Santiago, C.: Incorporating Fisheying into a Visual Programming Environment, *Proc. 1996 IEEE Symposium on Visual Languages*, pp.20–27 (1996).
9) Wang, X. and Miyamoto, I.: Generating Customized Layouts, *Proc. Graph Drawing '95*, pp.504–515 (1995).

(平成9年6月30日受付)
(平成10年1月16日採録)



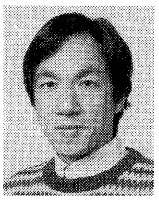
豊田 正史（学生会員）

1971年生．1994年東京工業大学理学部情報科学科卒．1996年同大学大学院情報理工学研究科数理・計算科学専攻修士課程修了．現在同大学院博士課程在学中．ビジュアルプログラミング，ユーザインタフェースソフトウェアに興味を持つ．ACM，IEEE CS，日本ソフトウェア科学会各会員．



高橋 伸（正会員）

1968年生．1991年東京大学理学部情報科学科卒．1993年同大学大学院理学系研究科情報科学専攻修士課程修了．1995年同大学大学院博士課程退学後，東京工業大学情報理工学研究科数理・計算科学専攻助手，現在に至る．ユーザインタフェースソフトウェア，視覚化，制約に興味を持つ．日本ソフトウェア科学会およびACM各会員．



柴山 悦哉（正会員）

1959年生．1983年京都大学理学研究科数理解析専攻修士課程修了．同年同専攻博士課程を中途退学し，東京工業大学理学部情報科学科助手に着任．以降，龍谷大学専任講師，東京工業大学理学部助教授を経て，1994年より，東京工業大学大学院情報理工学研究科助教授．1991年理学博士（東京大学）．主としてプログラミング言語に関する研究を行っており，言語設計，言語処理系の実装，プログラミング環境，意味論などに興味がある．また，関連分野として，システムソフトウェア，ユーザインタフェースソフトウェア，ソフトウェア工学などにも興味を持っている．ACM，IEEE CS，日本ソフトウェア科学会各会員．