Hyper Mochi Sheet: 複数フォーカスズーミングエディタにおけるナビゲーションのためのフォーカス予測手法

Hyper Mochi Sheet: Predictive Focus Techniques for Navigation in Multi-focus Zooming Editors

豊田 正史　　高橋 伸　　柴山 悦哉*

Summary. We propose a user interface that supports navigation of hierarchical networks in multi-focus zooming editors. This interface makes it easier to navigate large scale networks by predicting appropriate sizes of focused nodes. In addition, it provides suitable layouts for editing by selecting proper focuses during navigation. We implemented this interface as a library named Hyper Mochi Sheet for Amulet GUI toolkit. Using Hyper Mochi Sheet, we implemented a visual programming editor that enables efficient editing of large visual programs.

1　はじめに

大規模なグラフ構造を扱うビジュアルプログラム (VP) のエディタにおいては、大規模な階層ネットワーク図式をどのように狭い画面に表示し、その内部をナビゲーション・編集するインタフェースを提供するかが問題となる。大規模なグラフ構造の理解を容易にするため、多くの VP はいくつかのノードをまとめて入れ子状に階層化する方法をとっている。この階層化は部品のライブラリを分類する場合にも使われる。またノード間のデータフローを表わすリンク、部品の使用箇所から定義を参照するためのリンクなど、入れ子関係で表現できない階層間のハイパーリンクが存在する。結果としてエディタでは、図 1 に示すような、図式表現を扱う必要がある[1] 。

また VP の表示には、複数フォーカスおよびコンテキスト情報が重要となる。VP の編集の際には複数箇所を同時に閲覧することが良く行われる。例としては、部品のドラッグ&ドロップ、部品の定義を参照しながら自分のプログラムを編集する、他のプログラムを参考にして編集を行う、などが挙げられる。したがって複数の部分にフォーカスを当てられることが重要となる。また、大きなグラフを編集する際には、グラフのどの部分を編集しているのか、というコンテキストに関する情報があると便利である。

以上の特徴から VP の表示には、fisheye view[3] を応用した複数フォーカスの階層ネットワーク表示手法 [6] [?] [4] [1] が有効である。しかし、これらはレイアウトが固定されたネットワークの fisheye view 表示手法であり、ナビゲーションおよび編集のユーザインタフェースには重点がおかれていない。

* 東京工業大学 情報理工学研究科 数理・計算科学専攻、{toyoda, shin, etsuya}@is.titech.ac.jp

[1] ノードの使用部分から定義部分へのリンクは画面に現れていない

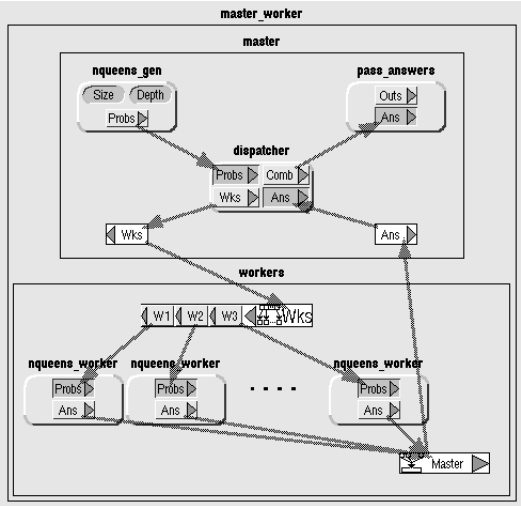

図 1. 階層ネットワーク構造

本研究では、ナビゲーション支援に重点を置いている。複数フォーカス fisheye view を用いたエディタにおいてどのようなナビゲーションインタフェースが有効であるかは、まだ明らかになっていない。単一フォーカスの場合にはナビゲーションは単なるフォーカスの移動である。一方、狭い画面の上では詳細に見られるフォーカスの数が限られるため、複数フォーカスでのナビゲーションでは、新たな場所にフォーカスを加えるだけでなく、不必要なフォーカスを削除する操作も行わなくてはならない。このため編集に適切なレイアウトを得るには比較的手間がかかる。したがってフォーカス操作を容易にするための支援を行うことが重要となる。

我々は、編集操作からの予測を用いてナビゲーションの支援を行う方法を提案する。階層ネットワークの fisheye 表示におけるナビゲーションの手段としては: (1) 階層構造の親子リンクを辿って子ノードを拡大・縮小する、(2) ハイパーリンクを辿ってリンク先を拡大する、という 2 種類の方法がある。ユーザは、ノードを詳細に編集するときはノードをどちらかの方法で拡大し、編集が終われば元の大きさに縮小する。このようなナビゲーションに対して、以下のような支援を行う。

平常時、フォーカス時のサイズの決定 ユーザはズーム操作によって、ノードのサイズを自由に変更することができる。しかし、ノードの編集を行う度に毎回ズーム操作を行うのは煩わしいため、適切なサイズへの変更をすばやく行えるようにすることが必要となる。またユーザがハイパーリンクを辿った時には、リンク先のノードを自動的に拡大しなくてはならない。そこでユーザの意図にあった適切なサイズをある程度予測し、そのサイズへの切り替えを行えるようにする。

フォーカスの選択 編集に適切なレイアウトを得るには、必要なノードの拡大、不必要なノードの縮小という手間のかかる操作が必要となる。この手間を軽減するために必要なフォーカスを予測・選択し、ナビゲーションの際には自動的に不

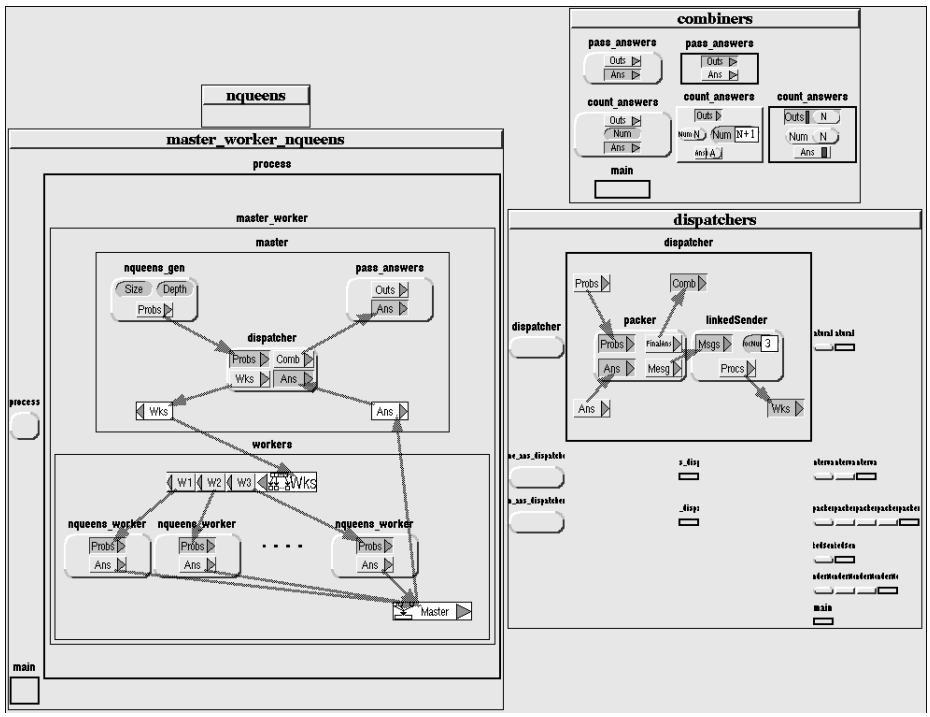

図 2. KLIEG **のエディタ**

必要なノードを縮小する。例えば、ユーザがハイパーリンクを辿った時、リンク元を見る必要がない場合にはリンク元を縮小する。

我々はこれらの手法を Mochi Sheet[7] に導入し、汎用のライブラリ Hyper Mochi Sheet として実装した。Mochi Sheet は階層ネットワークの編集をなめらかなズーミングを用いて行う編集用ズーミングインタフェースライブラリである。汎用のライブラリとした理由は、上記の VP の特徴には一般的なものが多く、ファイルブラウザ、プレゼンテーションエディタなど様々なアプリケーションに応用が可能であるためである。我々は Hyper Mochi Sheet を用いて KLIEG[5] のエディタ (2) を実装し、その有効性を確認するための実験を行っている。

本論文の構成は次の通り。2 節では関連研究について述べる。3 節ではフォーカスサイズの予測手法を、4 節ではフォーカス選択手法を解説する。5 節でまとめと今後の課題を述べる。

2 関連研究

複数フォーカスの fisheye view を用いて、ナビゲーションおよび編集を支援するユーザインタフェースはいままでほとんど提案されていない。ここでは一部の機能を実現しているもの、および、ズームの手法が異なるものとの関連について述べる。

Continuous Zoom [1] は、階層ネットワークのなめらかなズームを実現したインタフェースである。しかしレイアウトが固定で編集ができない。またハイパーリン

ク機能も提供していない。ナビゲーションに関しては一つのノードに対するズーム操作しか用意されておらず、望みのレイアウトを得るには手間がかかる。

Pad++ [2] は、基本的に単一フォーカス、パン・フォーカス方式のインタフェースを提供する。またハイパーリンク機能も提供している。パン・フォーカス方式であるためコンテキスト情報を参照することはできない。複数フォーカスはマルチウインドウを用いて対応しているがウインドウのレイアウトはユーザが適切に行う必要があり、不必要なウインドウもユーザが自分で削除しなければならない。またフォーカスのサイズを自動的に決定するなどの支援も行われていない。

3　フォーカスサイズの予測

フォーカスサイズの予測を用いると、ノードがフォーカスされた時のサイズ、および平常時のサイズをユーザが明示的に指定しなくとも、システムがある程度適切に決定することができる。この手法では、ノードのサイズをユーザの意図に応じて以下のように 2 種類に分類する。Mochi Sheet の使用経験から、これら 2 種類以上のサイズはほとんどの場合必要ないと考えられる。

小サイズ 平常時のサイズ。ノードを周りの子ノードとともにレイアウトするとき用いる。詳細でない編集ならば行える。大サイズより面積が小さい。

大サイズ フォーカス時のサイズ。ノードの内部を詳細に編集するときに用いる。小サイズ以上の面積を持つ。

サイズ予測を用いることで親子間のナビゲーションを効率よく行うことが可能になる。階層を下に移動するにはノードを大サイズに切り替えていけばよく、編集が終わればノードを小サイズに切り替えれば良い。小・大サイズをユーザが明示的に指定する手間はほとんどの場合なくなるため、ユーザは編集に専念できる。

3.1　サイズ予測の方法

サイズの予測は、対象とするノードの親ノードおよび子ノードへの編集操作の履歴を用いて行う。この予測は、Mochi Sheet のユーザテスト、使用経験から得られた以下のようなヒューリスティックスを用いて行っている。

1. 同じノードに対する連続したリサイズ操作は、サイズの微調整を意図していることが多い。このため微調整中のサイズへの切り替えは必要ない。
2. ノード内を編集するのは、多くの場合ノードの大きさを決めてからである。
3. 内部の編集を終えるとノードを小サイズに切り替える。

図 3 は、これらのヒューリスティックスに基づいたサイズ予測を模式的に表わした図である。各数直線はノードの面積を表わしており、小・大サイズ、および実際のサイズ (実サイズ) がプロットされている。この図における予測は以下のような規則で行われている。これらの規則は、微調整中の必要ないサイズを無視して、サイズが決定したと見なせるときだけ小・大サイズを変更するようになっている。

ノードを新しく生成した時 ノードの小・大サイズを「小サイズ $<$ 大サイズ」となるよう適当に定める。

ノードにリサイズ操作を行った時 1 のヒューリスティックスから、まだサイズの微調整中である可能性があると判断し、ノードの小・大サイズは変更しない。ノードは一時的に小サイズでも大サイズでもなくなる (図 3 中央への遷移)。

ノード内で編集操作を行った時 ノード内で図形の追加、削除、移動、リサイズを行った場合に編集を行ったとみなす。ノード内の編集が行われると、2 のヒューリスティックスから、ノードの大きさが決められたと判断する。そこで、ノードの小サイズか大サイズのどちらかを実サイズに変更する (図 3 下への遷移)。この変更は以下のように行う (但し、S_{Real}: 実サイズの面積、S_{Large}: 大サイズの面積、S_{Small}: 小サイズの面積)。

$$\begin{cases} |S_{Real} - S_{Small}| < |S_{Real} - S_{Large}| & \text{ならば小サイズを実サイズに変更} \\ |S_{Real} - S_{Small}| > |S_{Real} - S_{Large}| & \text{ならば大サイズを実サイズに変更} \end{cases}$$

結局、実サイズの面積に近い方のサイズが変更されることになる。

ノードを小サイズに切り替えた時 3 のヒューリスティックスから、内部の編集は終わり、各子ノードのサイズが決定されたと判断する。各子ノードについて小サイズか大サイズのどちらかを実サイズに変更する[2] (図 3 下への遷移)。この変更は、ノード内の編集時と同じ方法で行う。

3.2 サイズ予測の補助インタフェース

ノードを小または大サイズへ切り替えるには、ノードのポップアップメニュー、またはノードに付けられたボタンを使用する[3] 。また、予測が常にユーザの意図を反映するとは限らないため、誤りを修正するためのインタフェースも必要となる。さらに、深い場所の編集が終わった時、何階層か上まで小サイズに切り替えたい場合がある。このために祖先ノード[4] のうち深いものから順に小サイズ (大サイズ) に切り替えるインタフェースも提供する。

図 4〜図 6 は KLIEG のエディタでこれらの補助インタフェースを使用する様子を表わしている。図 4 は中央にあるネットワーク図をポップアップメニューを用いて小サイズに切り替えようとしている様子である。切り替えが終わると図 5 のとおり半透明の 4 つのボタンが図形の近くに現れる。予測されたサイズが正しければ気にせずに他の操作を行えばこれらのボタンは消える。

左右の矩形状のボタンは、予測されたサイズを修正するために使用する。各ノードはそれぞれリサイズの履歴を持っており、左ボタンは履歴中で今のサイズより小さいもの、右は大きいものを順番に提示する。上下の矢印状のボタンは、親ノードに対して同じ切り替え操作を行う時に使用する。上ボタンを繰り返し押すことで先祖のサイズを順番に同種類のものに変更することができる。下ボタンは上にいきすぎたときに下の階層に戻るために使用する。図 6 は図 5 の状態から上ボタンを押した結果である。さらに上の階層のノードが小サイズに切り替わっていることがわかる。

[2] リサイズ操作と、小・大サイズへの切り替えは区別する
[3] アプリケーションプログラマが選択して実装できる
[4] 親ノード、親ノードの親ノード、、、をすべて集めた集合

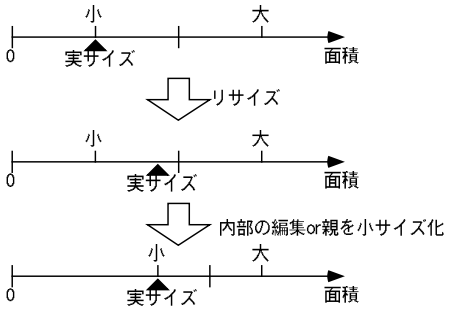

図 3. サイズ予測の様子

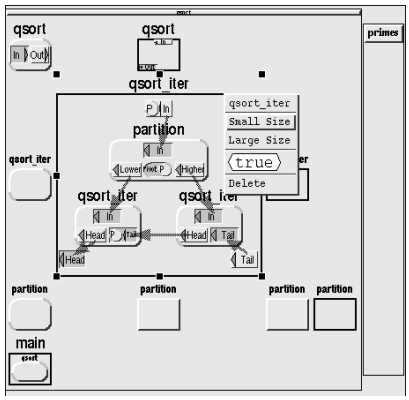

図 4. Small Size を選択

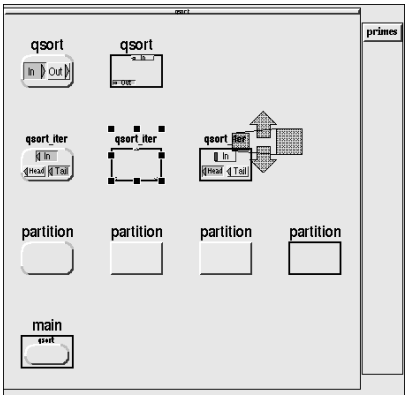

図 5. 上ボタンを押す

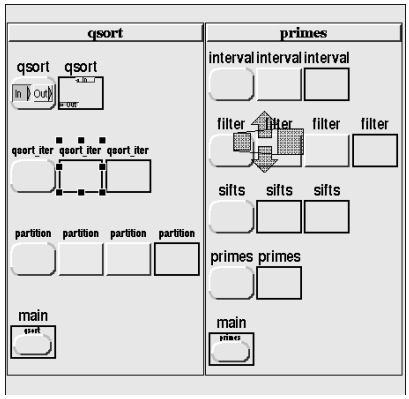

図 6. 親も小サイズに変更

4 フォーカス選択

フォーカス選択では、ユーザが親子リンク・ハイパーリンクを辿った時にリンク周辺のノードのサイズをどのように変化させるかを予測する。この予測を行うことで、「あるノードを編集しながらリンク先のノードを同時に見る」「今見ているノードを編集するのをやめてリンク先ノードを編集しにいく」といったユーザの意図をある程度読み取り、必要なフォーカスを自動的に選択することが可能になる。

Hyper Mochi Sheet におけるリンクは、親子リンク、ハイパーリンクの 2 種類である。親子リンクは子ノードを大サイズに切り替えた時に辿ったと見なす。ハイパーリンクはノードに付けられたボタンやポップアップメニューを用いて辿ることができる。リンクを辿ったときにサイズが変更されうるノードを以下に挙げる。これらのノードのどれを拡大し、どれを縮小するかが予測の対象となる。

1. リンク先ノード、およびその祖先ノード

2. リンク元ノード、およびその祖先ノード

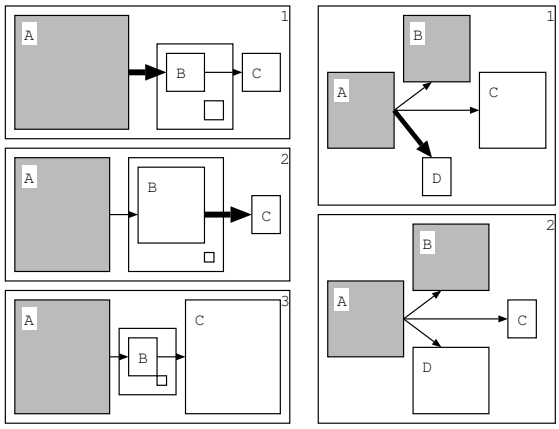

図 7. フォーカス選択

3. リンク先の兄弟ノード[5] 、およびその祖先ノード

まずリンク先のノードは、無条件に内容の詳細が見えるように大サイズに拡大しなければならない。このとき、その祖先ノードもリンク先のノードが見えるように大サイズに拡大される。図 7 の左側 2 番目の図はハイパーリンクを辿った時の様子を表している。左側 1 番目の状態から太い矢印のリンクを辿るとノード B が拡大されその親も拡大される。

リンクを辿った時には、その様子をアニメーションすることで、どこがリンク先なのかを分かりやすく表示する。リンク先を拡大する際には、リンク先の祖先ノード内の一番親のノードから順番に大サイズに切り替え、その様子をアニメーション表示する。

リンク先ノードは常に拡大するため予測の必要がない。従って、サイズの予測はリンク元、リンク先の兄弟ノードに対して行う。これらのノードサイズの予測は、ユーザがノードを編集する意志があるかどうか (編集中かどうか) を判断することで行う。編集する意思があると思われるなら大サイズを選択し、意思がないと思われるならば小サイズに縮小する。

あるノードの編集意思の判断には、「ノードの子ノードの中に、大サイズまたは一時的サイズのノードが存在すれば、編集を行う意思がある」というヒューリスティックスを用いる。これを用いて、以下のような手順で周辺のノードサイズの変更を行う。

1. リンク先ノード、およびその祖先ノードを大サイズに変更する。
2. リンク元ノード、およびリンク先の兄弟ノードの編集意思を上記のヒューリスティックスを用いて判断する。
3. 編集意思のあるノードを大サイズに切り替える[6] 。このときも祖先ノードの一番親から切り替えのアニメーションを行う。

[5] 親子リンクの時は、階層構造における兄弟ノード、ハイパーリンクの時はハイパーリンクにおける兄弟ノード

[6] ほとんどの場合、編集意思のあるノードは大サイズであるので、その大きさを保つだけで良い

4. 編集意思のないノードを小サイズに切り替える。この際、祖先ノードの一部またはすべてを小サイズに切り替える。これは祖先ノードに編集意思のあるノードが含まれる可能性があるからである。この場合、祖先ノードの内、子の方から順に切り替えを行っていき、編集意思のある祖先ノードにたどり着いた時点で切り替えを終了する。

図 7 はフォーカス選択のようすを表わしている。この図では灰色のノードが編集中と判断されている。このため左側 2 番目ではノード A は拡大されたままである。ノード B は編集中でないためノード C へのリンクを辿ると (左側 3 番目)、親ノードと共に小サイズに切り替えられる。さらに上の祖先ノードはノード A が編集中であるのでそのままとなる。

リンク先の兄弟ノードに対して予測を行うことは、リンクの一覧 (例:キーワードサーチの結果) などを扱うときに有効である。リンク先のうち編集中のものだけを拡大しておけるため、一覧を順番に参照していく際に必要のないものは自動的に小サイズに切り替えられる。この様子を図 7 の右側に示す。ノード A からリンクが張られているノード B、C、D のうち編集中の B のみがサイズを保ち、その他は最後に A からリンクを辿られたものだけが拡大されている。

5 まとめと今後の課題

複数フォーカスの fisheye view を用いたエディタにおけるナビゲーションに有効な予測手法を提案した。今回の手法では、ヒューリスティックスが固定されているが、今後は編集およびレイアウトの履歴など、予測の手がかりを増やし、パラメタ化を行って、学習により好みのレイアウトが得られるように拡張していきたい。

手がかりを増やすためには、アプリケーションのセマンティクスを利用したり、ハイパーリンクに属性をつけるなどの工夫が必要になると考えられる。また実装したエディタを用いてユーザテストを行い有効性を確認して行く予定である。

参考文献

[1] Lyn Bartram, Albert Ho, John Dill, and Frank Henigman. The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Space. In *Proceedings of UIST '95*, pp. 207–215, November 1995.

[2] Benjamin B. Bederson and James D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In *Proceedings of UIST '94*, pp. 17–26, November 1994.

[3] George W. Furnas. Generalized Fisheye Views. In *Proceedings of ACM CHI'86*, pp. 16–23. Association for Computing Machinery, 1986.

[4] Manojit Sarkar, Scott S. Snibbe, Oren J. Tversky, and Steven P. Reiss. Streching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens. In *Proceedings of UIST '93*, pp. 81–91, November 1993.

[5] Masashi Toyoda, Buntarou Shizuki, Shin Takahashi, Satoshi Matsuoka, and Etsuya Shibayama. Supporting Design Patterns in a Visual Parallel Data-flow Programming Environment. In *Proc. 1997 IEEE Symposium on Visual Languages*, pp. 76–83, September 1997.

[6] 三末和男, 杉山公造. 図的思考支援を目的とした図の多視点遠近画法について. 情報処理学会論文誌, Vol. 32, No. 8, pp. 997–1005, 1991.

[7] 豊田正史, 志築文太郎, 高橋伸, 柴山悦哉. Mochi Sheet: ズーミングとレイアウト編集機能の統合. インタラクション'97 論文集, pp. 79–86. 情報処理学会, February 1997.