Interaction Techniques for Navigating and Editing Nested Networks through Multi-focus Distortion Views

by

Masashi Toyoda

A thesis submitted to

Department of Mathematical and Computing Sciences
Graduate School of Information Science and Engineering
Tokyo Institute of Technology

for the degree of

Doctor of Science

March 1999

For my family

and for my friends of fifteen years' standing
K. Jahana, H. Funatsu, T. Inoue, Y. Shirai and A. Tomioka

Acknowledgements

First, I would like to thank Professor Etsuya Shibayama for his continuous support and encouragement for this work. I also thank Professor Satoshi Matsuoka and Shin Takahashi for their helpful advice and comments.

Next, I would like to thank Buntarou Shizuki for his work on the visual tracer of KLIEG visual programming environment, and Toshiyuki Masui for his ideas and advice for the directory editor HishiMochi.

Finally, I would like to thank the TRIP meeting members, and my colleagues for their friendship and help. I also thank our test users for their participation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Two of the most basic and important tasks with computers are the navigation and editing of structured or semi-structured information such as:

- File systems,
- Hypertexts (e.g., HTML documents),
- Visual programs (e.g., JavaBeans),
- Object-oriented Designs (e.g., OMT and UML),
- Presentations.

To build these structures, hierarchical and graph representations are usually used. For example, in UNIX file systems, directories and files are hierarchically structured, and hypertexts have a graph structure, in which documents are linked by hyperlinks. Navigation is by browsing a display of information or by looking for objective data by following the hierarchy or links, and editing involves modification of individual data and information structures.

This task is required not only by experts and programmers, but also by ordinary users. The recent spread of World Wide Web (WWW) and people's desire to edit their home pages have increased the likelihood that ordinary users will need to perform such a task.

The requirements of editing differ from those of navigation. For example, the user interfaces of Web page builders differ from those of Web browsers. Here are some important differences in the view required for editing:

- It is important for the user to grasp the structure of the information, because editing involves modification of the structure. In addition, it is often useful to understand where the user is editing and the structure around editing points during modifying individual data. An outline view of the structure is required, since there is usually too much information to display all of the details in a single screen. In practice, most web page builders provide an outline view of hyperlink structures, unlike most web browsers.

- During editing, it is frequently necessary to handle multiple parts simultaneously. For example, the user may drag-and-drop information from one node to another, edit one node comparing with other nodes, and create a hyperlink between two nodes. Therefore, a web page builder is useless unless it supports multiple windows, whereas a web browser can be used in a single window without major problems.

The main objective of this research is to construct a framework of graphical user interfaces (GUIs) that satisfy above requirements and are simple and easy to use for editing.

1.1 Motivation

Practical editor applications still rely on traditional multi-window or multi-buffer (split window) based user interfaces, though they do not support efficient editing interfaces satisfying above requirements. Navigation and editing with these interfaces are complicated and time-consuming because:

- No relationship between opened windows (buffers) is shown, and the layout of the windows (buffers) is independent of the information space, making it difficult to grasp the structure of the information and to understand where the user is editing. To show an outline of the structure, one or more additional windows (buffers) are required. However, an outline window has a difficulty in showing correspondences of multiple windows to data in the information space.

- Management of multiple windows and buffers is tedious and time-consuming. In multi-window systems, the many opened windows overlap during editing, making it difficult to find the window needed for the next editing operation. Moreover, the user may have to rearrange windows in order to see them simultaneously. In multi-buffer systems, the user must explicitly split the window in the most suitable way (e.g., vertical or horizontal), and enter appropriate data in each buffer.

Distortion views [33, 48, 47, 29, 49, 41, 5, 11] are information visualization techniques that solve problems of multi-window and multi-buffer systems by showing details of multiple focal points and the overall context of the information displayed in a single view. To realize such a representation, distortion views magnify areas around focal points while shrinking other areas, as shown in Figures 1.1 and 1.2, which show distortion views of a regular 2D grid. Multi-focus distortion views (Figure 1.2) allow the user to specify multiple focal points, and provide more flexible layouts of focused areas than single-focus distortion views (Figure 1.1). Since distortion views always display the relative positions of each editing part in the overall information space, the user can easily grasp the structure of the information, and the relationships between focal points in the overall context. In addition, since magnified focal points are not overlapped in distortion views, there is no need to rearrange multiple focal points.

This research was motivated by the fact that distortion views have not been used in practical editor applications, although multi-focus distortion views seem to have potential for editing large information spaces. However, in reality, there are still problems with managing multiple foci, although management is easier than in multi-window and multi-buffer systems. The greater flexibility and freedom of multi-focus layouts often make greater demands on the user than do single-focus views, in which a change of layout involves only focus movement and a change in the magnification factor. In multi-focus views, the user may have to navigate the information by performing boring focusing and defocusing operations on multiple parts of the screen, in order to obtain the layout most suitable for a particular editing situation, which changes frequently during editing. The aim of this research is to develop a user interface that allows users to easily obtain the layouts required for various editing situations.

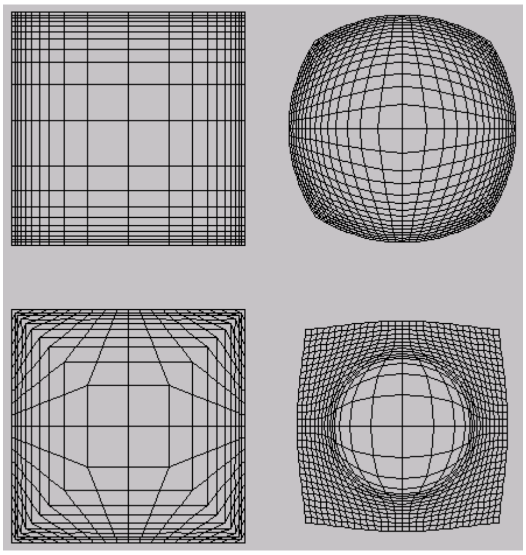


Figure 1.1: Single focus distortion views

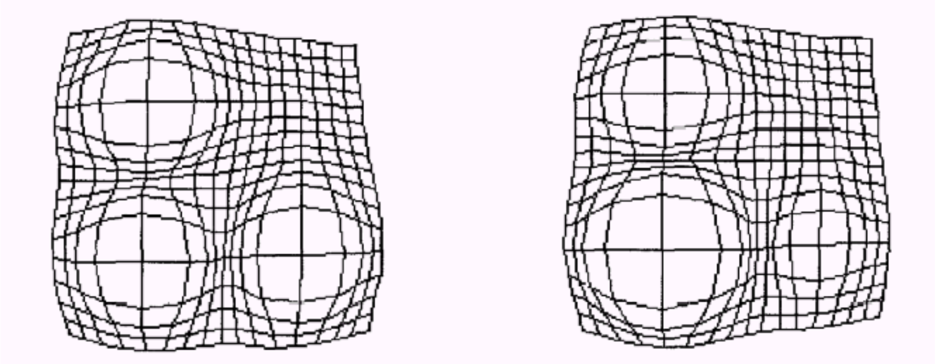


Figure 1.2: Multi-focus distortion views

1.2 Contributions

The major contributions of this dissertation are as follows:

- Development of a new GUI framework for navigating and editing structured information through multi-focus distortion views.
- Automatic focus management techniques for simplify users' tasks, by using a command history and application semantics.
- Implementation of the techniques as a GUI library, Hyper Mochi Sheet.
- Enhancement of usability with application semantics.

1.3 Content

Chapter 2 shows the application domain of our techniques using example applications of Hyper Mochi Sheet.

Chapter 3 provides a survey of related literature on distortion view techniques.

Chapter 4 analyzes general problems of using existing distortion views for navigation and editing. We show that the most important issue is trade-off between freedom of distortion layouts and the number of command invocations, and introduce our principles for solving these problems.

Chapter 5 proposes three interaction techniques for multi-focus management: focus size prediction, predictive focus selection, and dynamic query. Our techniques are shown to reduce the number of command invocations, thereby keeping distortion layouts free and flexible.

Chapter 6 shows that the use of application semantics provides enhanced support for navigation and editing, and decreases the number of command invocations. This is demonstrated by a case study, using a visual programming environment (VPE) as a sample application.

Chapter 7 discusses this work and describes contributions.

1.4 Publications

Some of the work in this thesis has been published or will be published in the following papers.

Chapter 5

Masashi Toyoda, Shin Takahashi, and Etsuya Shibayama. Mochi Sheet: A Zooming Interface witch Supports Efficient Editing of Large Visual Programs. *Transactions of Information Processing Society of Japan*, Vol. 39, No. 5, pp. 1395–1402, 5 1998. in Japanese.

Masashi Toyoda and Etsuya Shibayama. Hyper Mochi Sheet: A Predictive Focusing Interface for Navigating and Editing Nested Networks through a Multi-focus Distortion-Oriented View. In *Proceedings of ACM CHI'99*, May 1999. Accepted and to be published.

Masashi Toyoda, Toshiyuki Masui, and Etsuya Shibayama. HishiMochi: A Dynamic Search System with Nonlinear Zooming. In *Workshop on Interactive Systems and Software '98*, pp. 143–152, December 1998. in Japanese.

Chapter 6

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

Chapter 2

A Tour of Applications

Basically, our objective applications of Hyper Mochi Sheet are editors of nested networks. Nested networks are hypertexts, which consists of hierarchically nested nodes and hyperlinks. A node is a basic component of hypertexts, and includes texts, pictures, or other primitive representations. Nodes can be grouped hierarchically, so that hierarchical information can be directly mapped to nested nodes. In addition, any two nodes may be linked by a hyperlink, which allows the user to navigate from the source node to the destination node.

Nested networks can represent various information structures such as:

- File systems,
- Hypertexts (e.g., HTML documents),
- Visual programs (e.g., JavaBeans),
- Object-oriented Designs (e.g., OMT and UML),
- Presentations.

In the following, we show three example applications of Hyper Mochi Sheet; a visual programming environment, a presentation tool, and a directory editor.

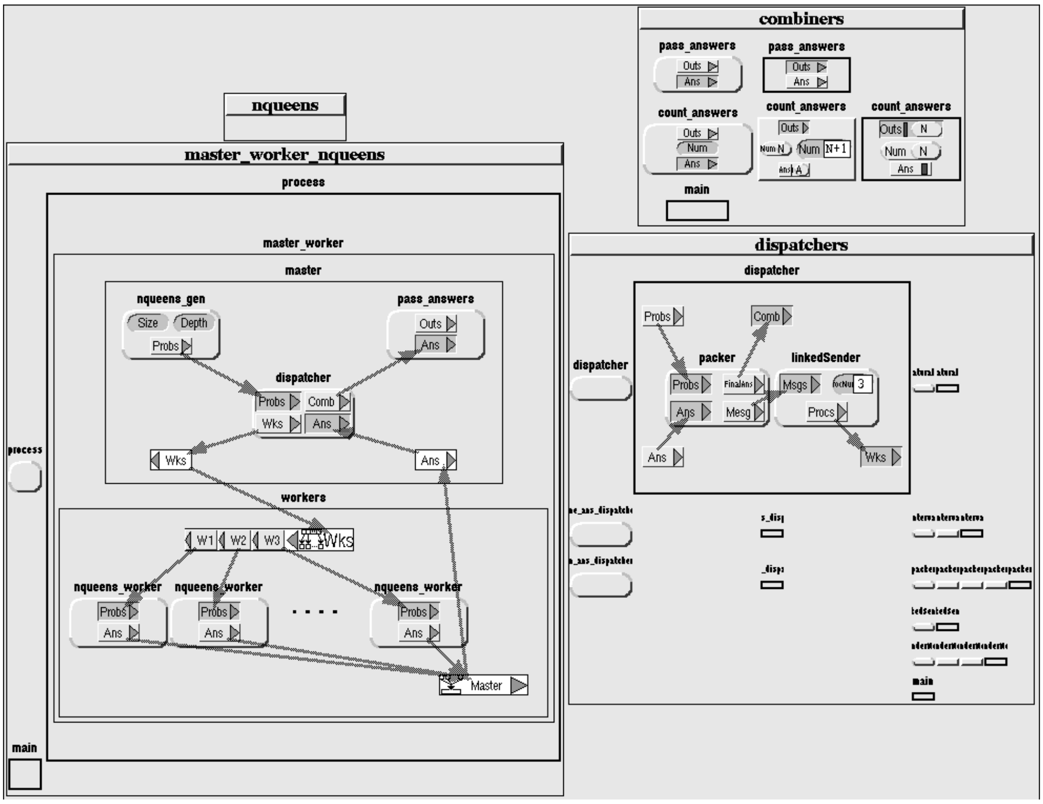


Figure 2.1: A visual programming environment KLIEG

2.1 KLIEG: A Visual Programming Environment

Figure 2.1 shows a visual programming environment KLIEG [66, 53, 65, 52], which addresses the scalability problem. KLIEG allows the programmer to edit multiple modules in one view and to construct nested data-flow networks for programming in the large. In Figure 2.1, there are four modules at the top level (**nqueens**, **combiners**, **master_worker_nqueens**, and **dispatchers**). The **master_worker_nqueens** module is the most magnified in size, and **nqueens** is rather shrunken. In addition to data-flow links, a program includes invisible hyperlinks from components to their definitions and documents, and a document is also a hypertext. When the user follows a hyperlink, the system focuses its definition. Moreover, the system automatically defocuses unnecessary parts. Using KLIEG, the user can easily drag-and-drop components between modules, and can navigate through a program using hyperlinks during editing.

2.2 KagamiMochi: A Presentation Tool

Figure 2.2 shows a novel presentation tool, KagamiMochi, which can handle hierarchically structured slides with hyperlinks. In effect, it can be used as a 2D visual outline processor for hypertexts. It allows the creator to edit a presentation through multi-focus views. It can also simultaneously show multiple slides and their overview during presentation. Each picture in Figure 2.2 is a different view of the same presentation. The top view is an overview, and the bottom view is a focus+context view in which the slide titled “Structure of Diagrams” is the focus. With a single mouse operation, the presenter can follow a hyperlink from a slide to the next, and the system automatically moves the focus to the next slide and adjusts the size of the slides. The creator of the presentation does not have to explicitly designate these sizes during editing, as the system predicts the sizes from a history of editing operations.

2.3 HishiMochi: A Directory Editor

Figure 5.15 shows a directory editor, HishiMochi[1][63]. It visualizes the directory structure of a file system as nested rectangles. The user browses the directory structure by magnifying and shrinking directories. Each leaf node is a text browser or an image viewer, so that the user can see the contents of the file. In addition, HishiMochi provides editing functions such as adding, deleting, copying, and moving directories and files.

[1] **Hi**erarchy sear**ch** **i**nterface for **Mochi**

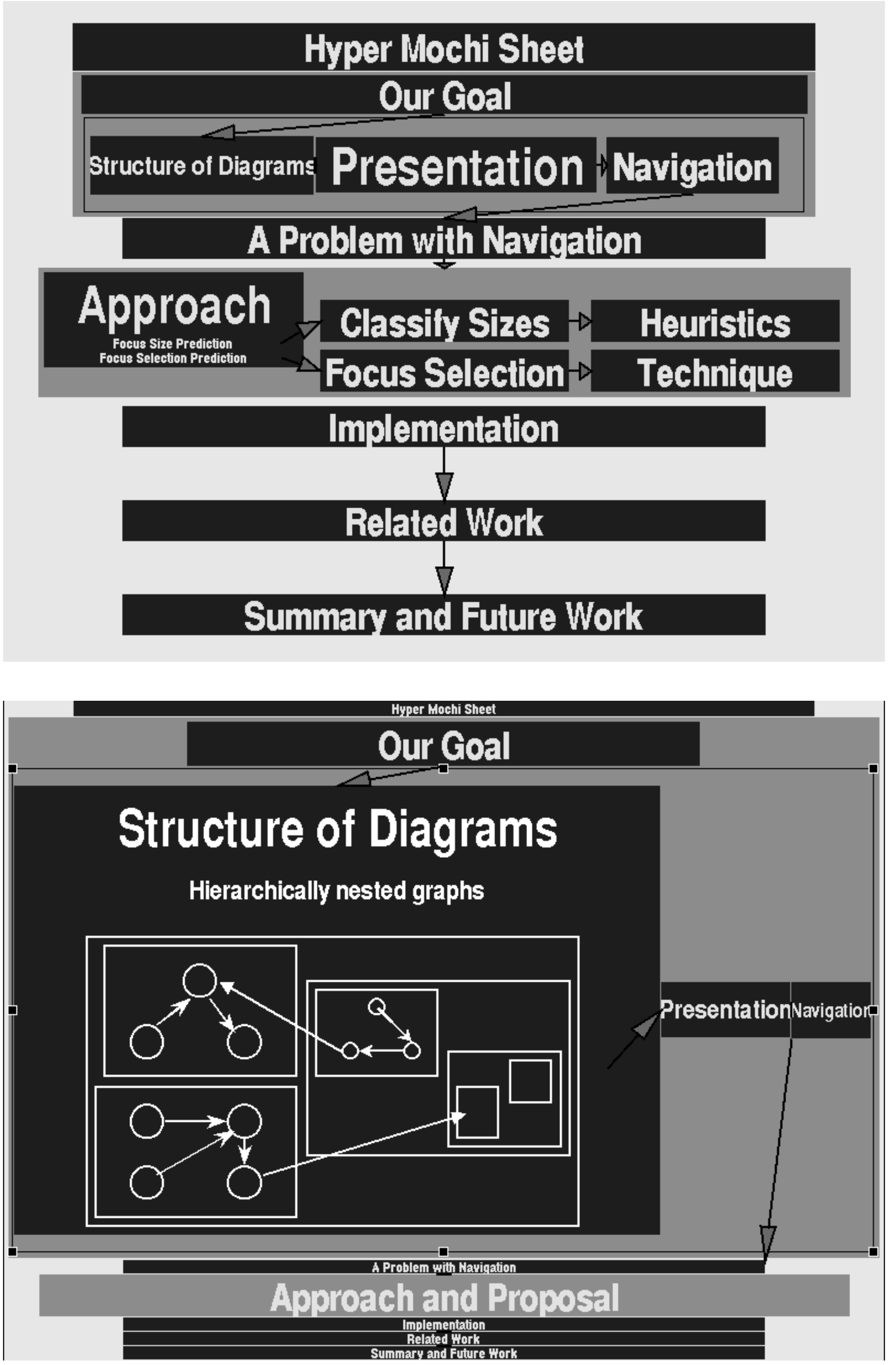
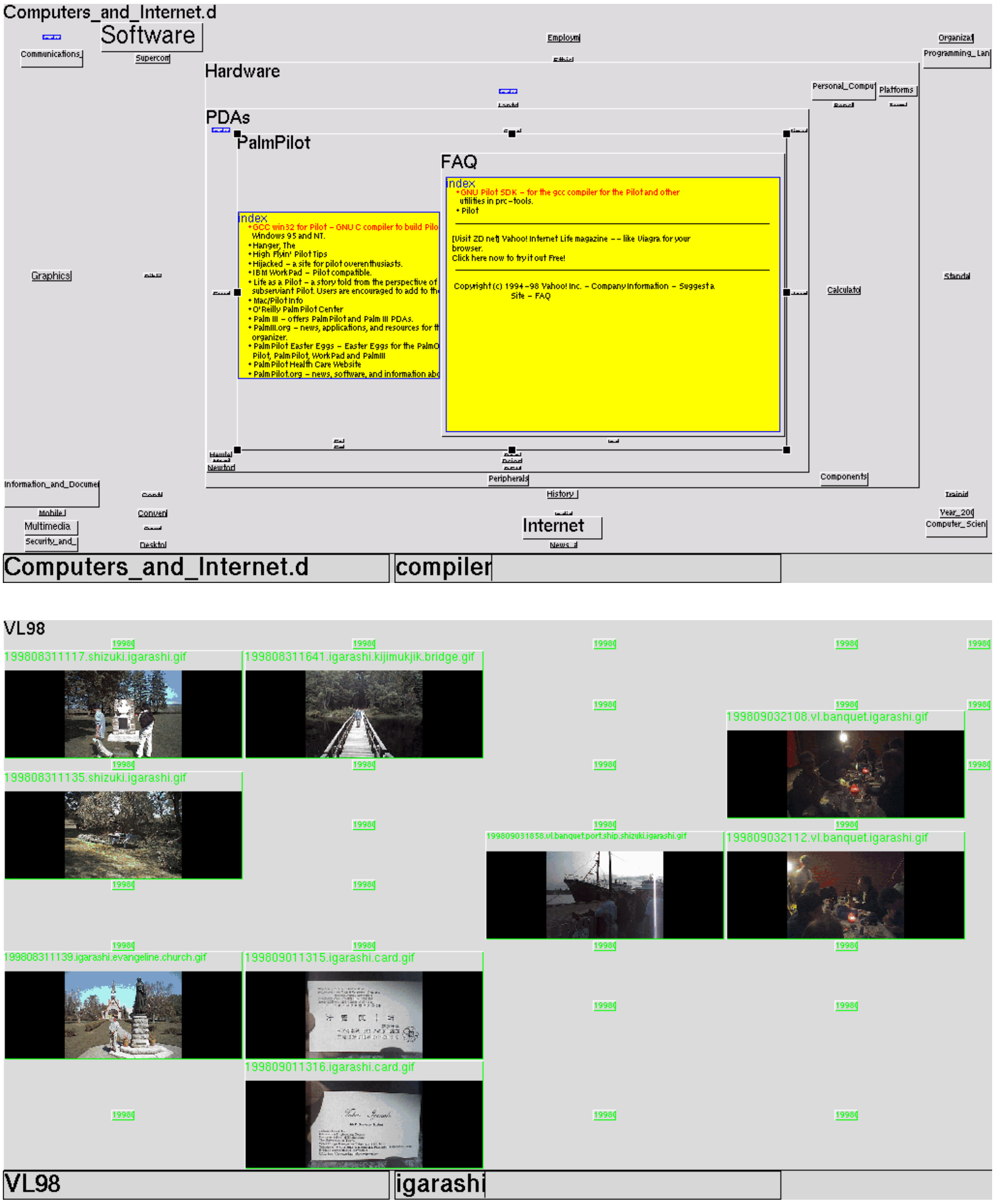


Figure 2.2: A presentation tool KagamiMochi



index
- GCC win32 for Pilot – GNU C compiler to build Pilo
  Windows 95 and NT.
- Hanger, The
- High Flyin' Pilot Tips
- Hijacked – a site for pilot overenthusiasts.
- IBM WorkPad – Pilot compatible.
- Life as a Pilot – a story told from the perspective of
  subserviant Pilot. Users are encouraged to add to th
- Mac/Pilot Info
- O'Reilly PalmPilot Center
- Palm III – offers PalmPilot and Palm III PDAs.
- PalmIII.org – news, applications, and resources for th
  organizer.
- PalmPilot Easter Eggs – Easter Eggs for the PalmO
  Pilot, PalmPilot, WorkPad and PalmIII
- PalmPilot Health Care Website
- PalmPilot.org – news, software, and information abo

index
- GNU Pilot SDK – for the gcc compiler for the Pilot and other
  utilities in prc–tools.
- Pilot

[Visit ZD net] Yahoo! Internet Life magazine – – like Viagra for your
browser.
Click here now to try it out Free!

Copyright (c) 1994–98 Yahoo! Inc. – Company Information – Suggest a
Site – FAQ

Figure 2.3: A directory editor HishiMochi

Chapter 3

Literature Survey

Numerous distortion view techniques have been proposed in last twenty years. Research on distortion views can be classified into information suppression techniques, image transformation techniques, and combinations of these techniques. The information suppression techniques determine the important and unimportant parts of the information space based on the current focal points, and suppress unimportant parts from the display; the image transformation techniques distort the original image by magnifying focal points and shrinking other parts.

We first introduce simple information suppression techniques. Then we describe image transformation techniques in which some combination of these techniques is included. Finally, we review multi-focus distortion views for nested networks, which are suitable for our target information structures.

3.1 Information Suppression Techniques

3.1.1 Generalized Fisheye Views

Generalized fisheye views [15] proposed by Furnas balance local detail and global context for various information structures, such as lists, trees, acyclic directed graphs. This technique was applied to a syntax-directed editor of C programs.

Generalized fisheye views are based on the DOI (Degree of Interest) function, which calculates the user's degree of interest in point x, and any points that have less DOI than a certain threshold DOI are suppressed. The DOI function is represented as follows:

$$DOI(x|. = f) = API(x) - D(x, f)$$

where $DOI(x|. = f)$ is the degree of interest in point x, given that the current focal point is f, $API(x)$ is the predefined *A Priori Importance* of x, and $D(x, f)$ is the distance between x and f. This means that the DOI increases with API and decreases with distance from the focal point.

3.1.2 Fisheye Presentation Strategy

Mitta extended Furnas' fisheye view and applied the algorithm to displaying aircraft maintenance data [38]. The example showed a solenoid assembly that includes a number of mechanical parts, such as screws and nuts.

The fisheye view was generated from the graph that represented parts and their connectivity, and parts that did not have enough DOI were merely suppressed. The improvement on Furnas' fisheye view was the ability to handle multiple focal points.

3.1.3 Fractal Views

Koike proposed fractal views [26], an improvement of Furnas' fisheye view, with the ability to handle tree structures with an unbalanced number of branches. Koike used fractal algorithms to make the amount of visible data for each focal point as equal as possible. This technique was applied to a syntax-directed editor for C programs. Later, it was applied to the VRML scene graphs in [27].

3.2 Image Transformation Techniques

Leung classified image transformation techniques in 1994[32], using transformation functions and magnification functions. A transformation function $T(x)$ transforms the point x in an undistorted view to one in a distortion view. A magnification $M(x)$ function, which is derived from the transformation function, represents magnification factor at the point x. Figure 3.1 shows these functions of a polyfocal projection [22], one of the earliest work on distortion views, used to display statistical data on cartographic maps. In Figure 3.1, x represents a distance from the current focal point (0) in an undistorted image, and $T(x)$ represents the distance in the distorted image. The distance is normalized by the distance from the focal point to the border of the screen. Using these functions, a regular two-dimensional grid is distorted as in Figure 3.2.

Leung's classification used such one-dimensional functions. Later, Keahey extended this to two dimension, using a concept of nonlinear magnification fields [24]. Though this concept provides more general description of distortion views, we use one-dimensional functions in the following to simplify descriptions.

3.2.1 Bifocal Display

Spence and Apperley proposed a one-dimensional bifocal display for database browsing [55]. Later, Leung extended the display to a two-dimensional version [31], and used it to display a map of the London Underground.

The bifocal display is characterized by transformation and magnification functions in Figure 3.3, and distorts a 2D regular grid as shown in Figure 3.4. In contrast to a polyfocal projection, a bifocal display transforms a point to x and y directions independently. Therefore, the focal region in the distorted image is also a rectangle. The focal region is magnified uniformly, and other regions are shrunken, either uniformly or uniformly to x or y direction according to their relative position to the focal region. The factors of demagnification are determined so as not to change the size of the display.

The user navigates the map by moving the focal region by a touch-sensitive screen. The user cannot change the size or the magnification factor of the focal region.

3.2.2 Fisheye Views

Hollands compared the user performance with a fisheye view and a simple scrolling view of a fictional subway network [19]. Hollands used the fisheye view similar to the bifocal display. The user performed three tasks: a route task, a locate/route task, and an itinerary task. The fisheye view improved users' performance in both the locate/route task and the itinerary task.

In this fisheye view interface, the user moved the focal point by clicking on a station on the map. There were no other operations to change layouts of the fisheye view.

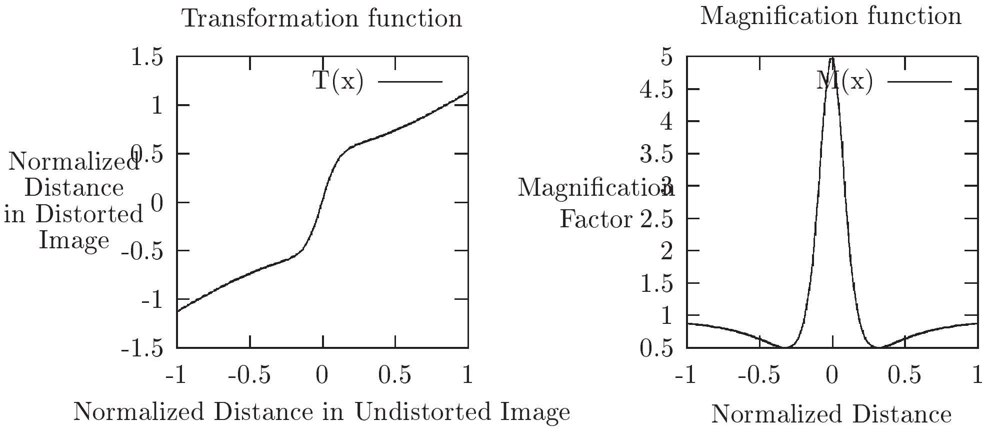


Figure 3.1: A transformation function of a polyfocal projection and the corresponding magnification function

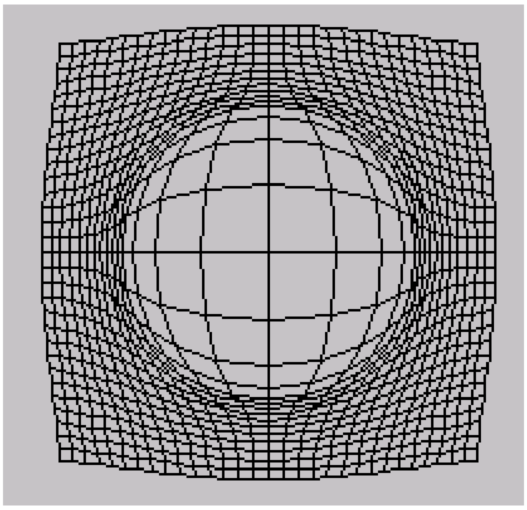


Figure 3.2: A polyfocal projection on a 2D regular grid

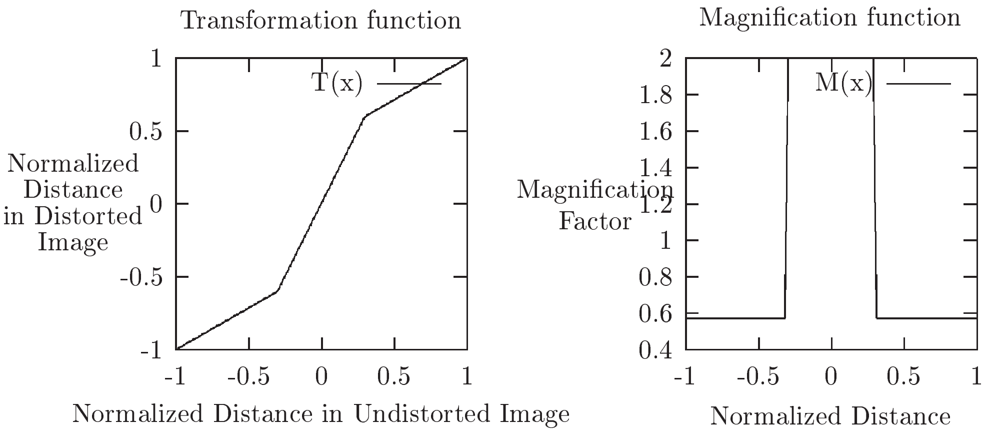


Figure 3.3: A transformation function of a bifocal display and the corresponding magnification function

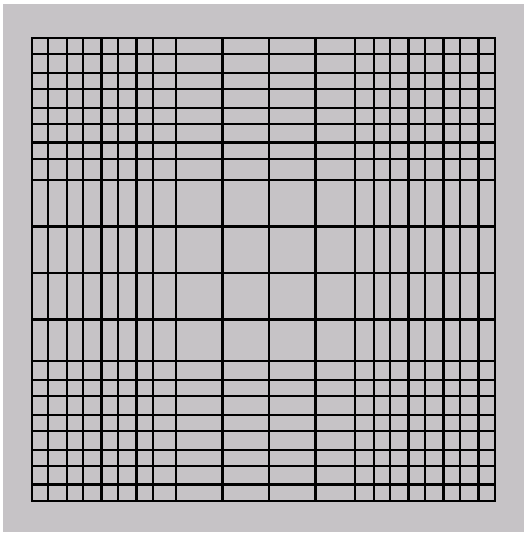


Figure 3.4: A bifocal display on a 2D regular grid

3.2.3 Perspective Wall

The perspective wall [33], proposed by Mackinlay, gives a three-dimensional look and feel to distortion views, as shown in Figure 3.6. Wide linear information is folded into a three-dimensional visualization, in which the center panel represents detail and two perspective side panels represent context. Transformation and magnification functions for the x direction are shown in Figure 3.5. This technique is used to visualize files, sorted by their modified time and classified by their type.

The perspective wall allows the user to move arbitrary points to the center panel and to change the magnification factor of the center panel. The user cannot change the size of center panel.

3.2.4 Graphical Fisheye Views

Sarker proposed two types of graphical fisheye views [48] for topological graphs. One was a cartesian fisheye view and the other was a polar fisheye view. Both views were based on one transformation function, shown in Figure 3.7. Transformation was performed only on nodes of graphs, then these nodes were linked by straight lines.

The cartesian fisheye view (the left hand side of Figure 3.8) transforms a point to x and y directions independently. The center of Figure 3.8 shows the transformation based on the polar coordinate system. The polar fisheye view remaps this transformation on a rectangular space, as shown in the right hand side of Figure 3.8.

Serker and Brown also provided an information suppression/enhancement mechanism, based on the concept of a priori importance (API) proposed by Furnas. This scaled nodes by their importance, and suppressed nodes with less importance than a threshold value.

The user can move the single focal point by dragging with the mouse. The user also changes the parameters of the transformation function and the API function, using a dialog box.

3.2.5 Document Lens

The perspective wall has a problem: it cannot use spaces in the four corners. The document lens [47] overcomes this problem by adding transformation in the y direction to the perspective wall. The document lens is similar in

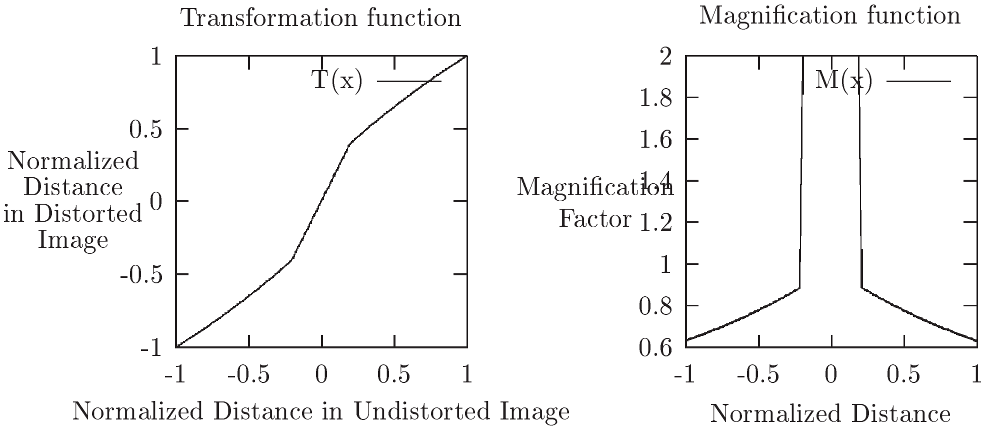


Figure 3.5: A transformation function of the perspective wall and the corresponding magnification function

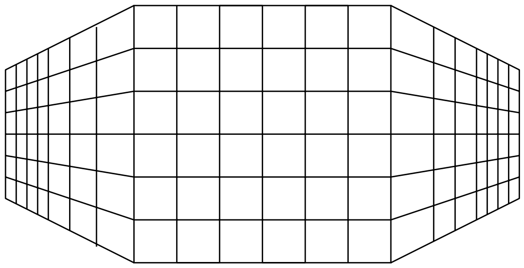


Figure 3.6: The perspective wall of a 2D regular grid

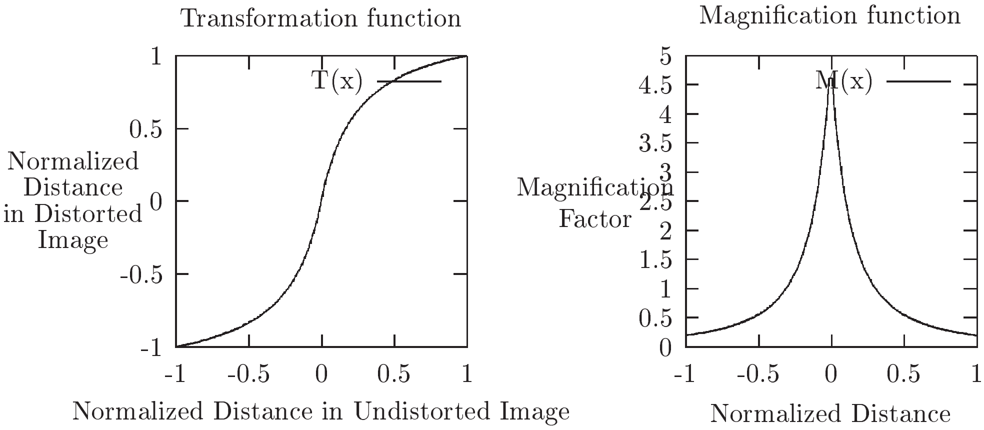


Figure 3.7: A transformation function of graphical fisheye views and the corresponding magnification function

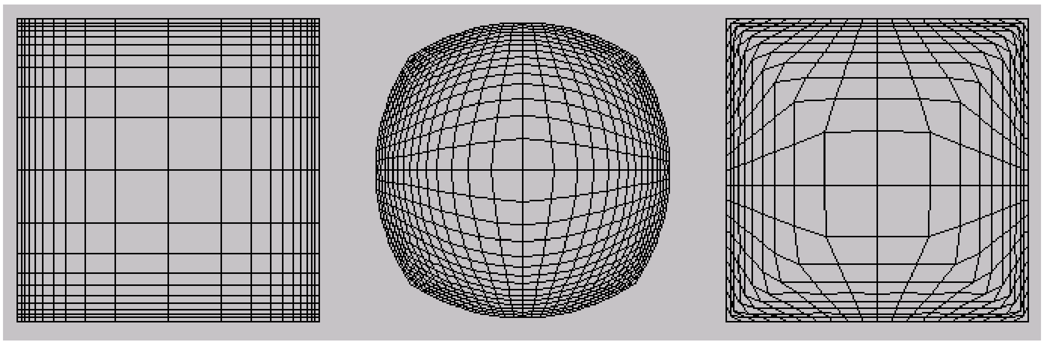


Figure 3.8: Cartesian, polar, and normalized polar fisheye views of a 2D regular grid

appearance to the polar fisheye view of graphical fisheye views, except that the focal region is magnified uniformly as a rectangular lens.

This technique can be used to browse a document laid out on a two-dimensional surface. The user can move a lens of fixed size over the surface of the document in 3D.

3.2.6 Table Lens

The table lens [46] applies distortion views to browse large tables, such as spreadsheets. The appearance of the table lens is similar to the bifocal display, except that it allows multiple foci and focal regions are based on discrete rows or columns. In addition, the table lens integrates multiple visualizations of data. For example, a numerical value is represented as a bar graph when it is in a context region, and is also represented as a text when it is in a focal region.

The user can select multiple rows or columns as focal regions and can flexibly control the focal regions. The user can therefore zoom a focal region, adjust a number of cells in a focal region, and slide a focal region.

3.2.7 FOCUS

The FOCUS [56] is another application of distortion views, to browse tables that include data that have multiple attributes. The user can perform various queries by direct manipulation, using cell sorting and selecting operations. For example, the user can select data based on a range of an attribute value, by sorting cells by the attribute value, and can select a desirable range by dragging with the mouse.

3.2.8 Hyperbolic Browser

Hyperbolic Browser [29] is a technique to display large tree structures. A tree is laid out on the hyperbolic plane and is mapped onto a circular display region, in which the focal node is at the center. It is extended to handle two foci in [30].

The user can select a focal node by clicking on the desired node using the mouse. The node is moved to the center of the display and other nodes are remapped on the hyperbolic plane. This transition is smoothly animated.

The user also moves the any node to any position by dragging with immediate feedback.

3.2.9 CATGraph

The CATGraph algorithm[23] uses the arctangent function for a transformation function. Similar to graphical fisheye views, it provides the cartesian and the polar transformation. In addition, it allows multiple foci.

The user can select multiple nodes as focal points, and can scale these points. The user can also create, delete, and move nodes by simple mouse operations.

3.2.10 Nonlinear Magnification Fields

The distortion views that we have shown distort images by transformation functions. In contrast, nonlinear magnification fields[24] allow the user to directory modify the shape of a magnification function by deriving the transformation function from the shape of the magnification function. The user can flexibly combine various distortion effects.

Keahey proposed a magnification brush interface that allowed the user to magnify an arbitrary area by painting the area. In addition, Keahey proposed a type of automatic focus management technique, called data-driven magnification, in which focal points are dynamically moved according to data that change in real-time. For example, Keahey applied this technique to a simulated air traffic control system, where regions of higher traffic density are automatically magnified.

3.3 Multi-focus Distortion Views for Nested Networks

3.3.1 Multi-viewpoint Perspective Display

Multi-viewpoint Perspective Display[60, 35] is one of the earliest works on multi-focus distortion views of nested networks. In [35], three distortion techniques were proposed and compared:

Fisheye display This is similar to Sarker's polar fisheye view [48], except that it uses arctangent as the transformation function.

Orthogonal fisheye display This is similar to Sarker's cartesian fisheye view [48], except it uses arctangent as the transformation function.

Biform display This improves the bifocal display [31] by supporting multiple foci.

It is concluded that the biform display is appropriate for displaying nested networks, since it preserves more of the topological relationships of networks than other techniques.

The biform display is used in D-ABDUCTOR[36] system, which is a diagram based ideas organizer based on the KJ-method. Using D-ABDUCTOR, the user edits layouts of text cards with segments of ideas by moving, grouping, and linking related cards with lines. During editing, the system automatically arranges layouts of cards [61]. This automatic support improves users' performance of editing processes [37]. With biform display, the user navigates hierarchy of cards by opening and closing groups, and by zooming.

Though D-ABDUCTOR provides good support for editing and navigating nested networks, it lacks support for navigation through links, and automatic management of multiple foci.

3.3.2 Treemaps

Treemap [20] visualizes hierarchical information structures efficiently using the available display space. It maps hierarchies onto a rectangular region in a space-filling manner. For example, it slices the display space horizontally for the first hierarchy, then cuts each slice vertically for the second hierarchy, and so forth. The position, size, and color of tiles convey the properties of nodes. For example, the size of a file in a directory can be mapped to the size of the corresponding tile.

Treemap allows the user to enlarge or shrink each tile. The user can also zoom into a sub-hierarchy, so that only the sub-hierarchy is displayed on the screen.

In addition, it also supports *dynamic queries* [54], which continuously update a search result as the user adjusts a slider or selects a button to ask a simple question about a property. If the user selects a property range, nodes within the selected range are highlighted. The user can also filter out all of the nodes outside the selected range. However, since filtering changes mosaic layouts drastically, the user may become confused.

3.3.3 Layout-independent Fisheye View

In [42], Noik also shows an algorithm that generates multi-focus distortion views of nested networks. This algorithm is not based on transformation of given layouts of nodes. Rather it determines sizes of nodes based on focal points and a layout-independent DOI function before a graph layout phase. Therefore it can generate a view that is not affected by the original layout of nodes. Noik applies this algorithm for navigating large hyperdocuments with nesting and link inheritance [14] techniques in [41].

In these papers, however, there is little discussion of how to construct the user interface for navigating hyperdocuments and managing multiple foci. They do not support editing, because implementations of graph layout algorithms are also left to the application programmer.

3.3.4 The Rubber Sheet Approach

Sarker proposed the rubber sheet approach[49] based on a morphing technique. The user can specify multiple focal regions as arbitrary closed convex polygons called *handles*, and can stretch these regions like a rubber sheet. Nodes in a focal region are scaled uniformly, and the other nodes are arranged by a morphing function.

This approach treats hierarchical information. When the area of a handle becomes large enough, detailed information (e.g., child nodes) is displayed in the handle. However, it is difficult to access nodes deep within a hierarchy, because the user cannot specify a focal region within an existing focal region. To access deep node, the user should precisely select a small area in the non-distorted view. In most cases, this is impossible.

The user can also edit layouts of nodes. However, since this technique does not have a general inverse mapping, areas that can be edited are restricted to the inside of handles.

3.3.5 Pad++

Pad++[7, 8] is basically a single focus and pan/zoom based interface. It realizes real time animation of panning and zooming in a logically infinite space including texts and images. In Pad++, each graphical object can be scaled individually. It supports multiple foci by multiple windows called portals. Since it does not perform automatic portal management, the user has to create, delete, and arrange multiple portals manually. Pad++ also supports hyperlink navigation in a single focus view, in which a focus moves along hyperlinks.

A web browser[9] in Pad++ is also developed. It integrates a web browser with distortion views of a history tree of visited pages. When the user follows a hyperlink, the destination page is zoomed, and other nodes are shrunken. In addition, the user can see multiple pages simultaneously by explicitly zooming each page.

In addition, Pad++ suffers from the problem of “desert fog” [21], a condition wherein a view of an information space contains no information for navigation. The solution proposed in [21] is to display indicators for invisible objects. Although this is useful for navigation, these indicators impair the original views. Originally, those solutions were unnecessary for distortion views.

Editing in the Pad++ environment is also difficult. Since each graphical object has its own scale, the user can not correctly grasp the size of the object. The multiscale editor MuSE[17] solves this problem by allowing editing through space-scale diagrams (details can be found in [16]). However, the interface is still indirect for ordinary users.

3.3.6 The Continuous Zoom

First, Schaffer modified the bifocal display technique for networks that are clustered hierarchically by enclosing rectangles, and named the algorithm *the variable zoom* [50]. This technique was applied to a network supervisory control system. Shaffer compared this technique with traditional pan and zoom interfaces, and showed by user tests that the variable zoom improved user performance. Later, Dill improved the variable zoom by continuous animation of zooming in [13], then Bartram improved space utilization using the DOI approach, and named the algorithm *the continuous zoom* [5].

The continuous zoom supports efficient navigation through hierarchical networks. The user controls the level of detail in nodes by opening and closing nodes. The child nodes of an open node are visible, allowing the user to see more detailed view. An open node is allocated more space than closed nodes. Closing a node makes the child nodes invisible, and makes it smaller than any open nodes. In addition to this automatic resizing, the user can resize nodes continuously with the mouse. The size of a node increases or decreases, while the user is pointing at the node and holding down a button.

However, the continuous zoom only supports navigation between a parent and children of node hierarchy, and neither support navigation with hyperlinks nor automatic multi-focus management. Moreover, it lacks support for editing networks.

3.3.7 The Intelligent Zoom

Bartram improved the continuous zoom with intelligent support by the semantics of the network supervisory control system, and this technique was called the intelligent zoom [3, 6, 4].

This suggests opening (magnifying) a node in an alarm condition, and when the node is opened it automatically selects an appropriate representation from several aspects of the node, such as a bar chart or a trend diagram. It also adjusts the size of nodes using DOIs based on alarm conditions of the system and user interactions. Since its application is a real-time control system, Bartram puts emphasis on providing awareness of alarm conditions.

This is one of several attempts for automatic focus management. However, since it merely suggests opening and closing nodes, it does not reduce explicit focusing and defocusing.

3.3.8 3-Dimensional Pliable Surface

3-dimensional pliable surface [11] generates a distortion view using the three-dimensional gaussian curve. This transforms a two-dimensional flat surface into a three-dimensional curved surface. The user selects arbitrary-shaped regions as foci, and they are pulled toward or pushed away from the user's view. Multiple focal regions are blended smoothly. It also handles nested information. When a focal region is magnified, the details in the region are displayed.

Both the rubber sheet [49] and this technique put emphasis on interfaces that support flexible selection of a shape as a focal region. These interfaces are effective for applications such as geographical maps. However, they are less effective in navigation with hyperlinks. It is also difficult to access nodes deep within a hierarchy, because the user cannot specify a focal region within an existing focal region.

3.3.9 Dynamic Fisheye Views

Noik proposed a generalized data visualization framework, dynamic fisheye views[43], which combines dynamic queries[54] and mapping, and implemented the Graphite system. Dynamic mapping refers to the ability to dynamically assign and modify mappings from data attributes to graphic properties at run-time.

A main dynamic query widget of Graphite is the frequency distribution histogram of a single numeric attribute. The range of the attribute is divided into a user-specified number of subranges, and the user can select arbitrary subranges to highlight satisfied nodes or to mask unsatisfied nodes. In addition, Graphite also supports regular (text) expression matching. The results of matching can be assigned to an attribute of a node. Matching is performed every time the user modifies the regular expression string. However, regular expression matching is limited to the identifier or labels of either the nodes or links in the graph.

Graphite also introduces the notion of focal points and similarity attributes for the user to perform similarity queries, that is to find nodes similar to the focal points. Graphite allows the user to define customized similarity functions that determine values of similarity attributes, and generates fisheye views using similarity attributes. Every time the user changes focal points, similarity functions are recalculated, and the results are reflected to the fisheye view display.

3.3.10 SHriMP

The SHriMP[58] visualization technique integrates distortion and pan/zoom viewing and is applied to a C programming source code browser. The user can combine those two views, for example, by zooming into and then distorting a subgraph. Distortion views in SHriMP are based on the SHriMP fisheye view algorithm [57] that supports various layout adjustment strategies for nested graphs.

Though SHriMP also supports navigation by hyperlinks with animations, it supports single focus navigation only by using pan/zoom effect. When the user follows a hyperlink, it first zooms out so that both the source and the destination can be seen, then zooms into the destination. This two-step effect is important to avoid confusing the user.

Chapter 4

Analysis

Although various distortion view techniques have been proposed, as shown in Chapter 3, they have not been used for practical editing tasks. In this chapter, we first discuss about recent trends in distortion views and then explain the problems associated with using distortion views for navigating and editing nested networks. Finally we show our approach to solving the problems.

4.1 Recent Trends in Distortion Views

Recent distortion view techniques allow the user to freely arrange foci and their shapes. However user interfaces for handling foci have become complicated. Table 4.1 compares user interfaces for distortion views. The following is the transition of those user interfaces.

1. Single focus, fixed focus shape, without scaling
2. Single focus, fixed focus shape, with scaling
3. Multi-focus, fixed focus shape, without scaling
4. Multi-focus, fixed focus shape, with scaling
5. Multi-focus, free focus shape, with scaling

The first has the most simple interface, and the last has the most complicated. To use the first interface, the user only needs to select a focal point, whereas

Table 4.1: Comparison of user interfaces for distortion views

	year	# of	type of	scale	graph	hyper	query
		foci	foci		edit	link	
Information suppression							
Generalized Fisheye Views	1986	1	node				
Fisheye Presentation	1990	N	node				
Fractal View	1995	1	node				
Image transformation							
Bifocal Display	1989	1	rect				
Fisheye Views	1989	1	rect				
Perspective Wall	1991	1	rect	√			
Graphical Fisheye Views	1992	1	node	√			
Document Lens	1993	1	rect	√			
Table Lens	1994	N	rect	√			
Hyperbolic Display	1994	2	node				
CATGraph	1994	N	node	√			
FOCUS	1996	N	rect				norm
Nonlinear Magnification	1997	N	any	√			
Image transformation for nested graphs							
M-V Perspective Display	1991	N	rect	√	√		
Treemaps	1992	N	rect	√			dyn
Layout-independent Fisheye	1993	N	rect	√			
Rubber Sheet	1993	N	poly	√	√		
Pad++	1994	N	rect	√	√	√	norm
Variable Zoom	1994	N	rect	√			
Intelligent Zoom	1994	N	rect	√			
Continuous Zoom	1995	N	rect	√			
3D pliable surface	1996	N	poly	√			
Dynamic Fisheye Views	1996	N	node	√			dyn
SHriMP	1997	N	rect	√		√	

# of foci; 1: single-focus, N: multi-focus

type of foci; node: node of graph, rect: rectangular region,
poly: polygonal region, any: any shape

query; norm: normal query function, dyn: dynamic query function to use the last interface, the user must specify multiple foci and their shape, and must scale them.

This transition shows that recent distortion view interfaces require the invocation of many commands for the user to obtain a desirable layout. Note that those interfaces are intended only for navigating information spaces.

In addition to those navigation commands, various commands (e.g., creating, deleting, and moving nodes) must be provided to supporting editing. The resulting interface therefore has too many commands for users to handle efficiently.

4.2 Analysis of the Editing Task through Distortion Views

Next, we analyze the task of editing nested graphs through multi-focus distortion views. The editing task can be divided into two processes, as follows. During editing, the user repeatedly performs these processes.

1. *Navigation for editing*

   The user creates a distortion view suitable for the current or next editing situation. The user magnifies nodes that are to be edited, and shrinks nodes that are unnecessary for the current or next editing situation.

2. *Editing*

   The user edits magnified parts by creating, deleting and moving nodes in the graph. Hyperlinks are also created and deleted.

As an example, we show a simple editing task in our visual programming environment KLIEG in Figures 4.1, 4.2, and 4.3. Figure 4.1 is the initial situation, in which all modules are shrunken. The user plans to drag-and-drop a process from the **node_pools** module to the **search_framework** module.

In the navigation for editing phase, the user magnifies the appropriate modules one by one. Figure 4.2 is the situation when navigation has finished. There is a placeholder node_pool to drop a process on the top-right of **search_framework**, and there are processes for the placeholder in **node_pools** such as stack and queue.

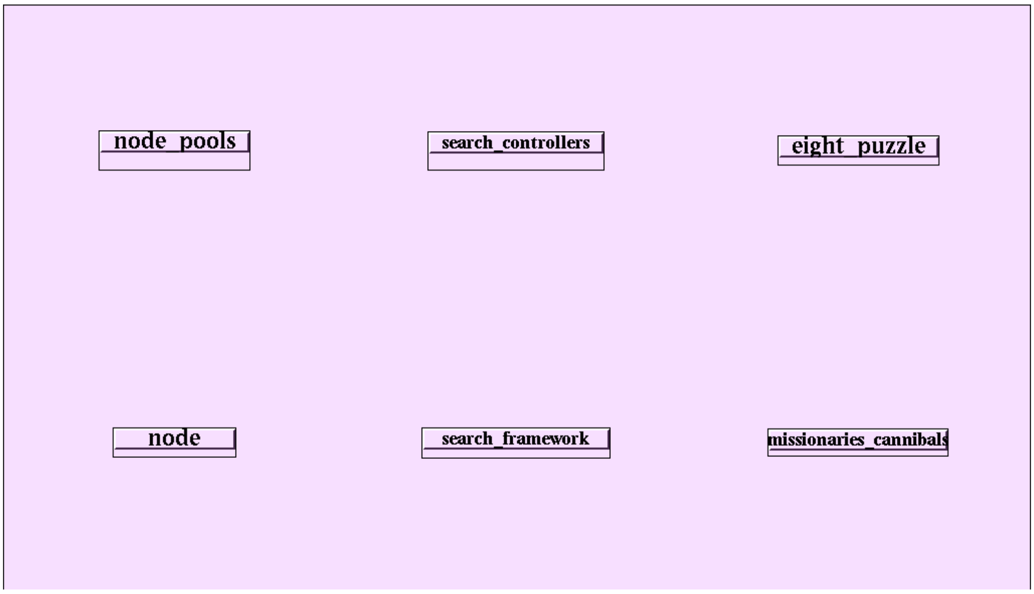


Figure 4.1: Initial situation

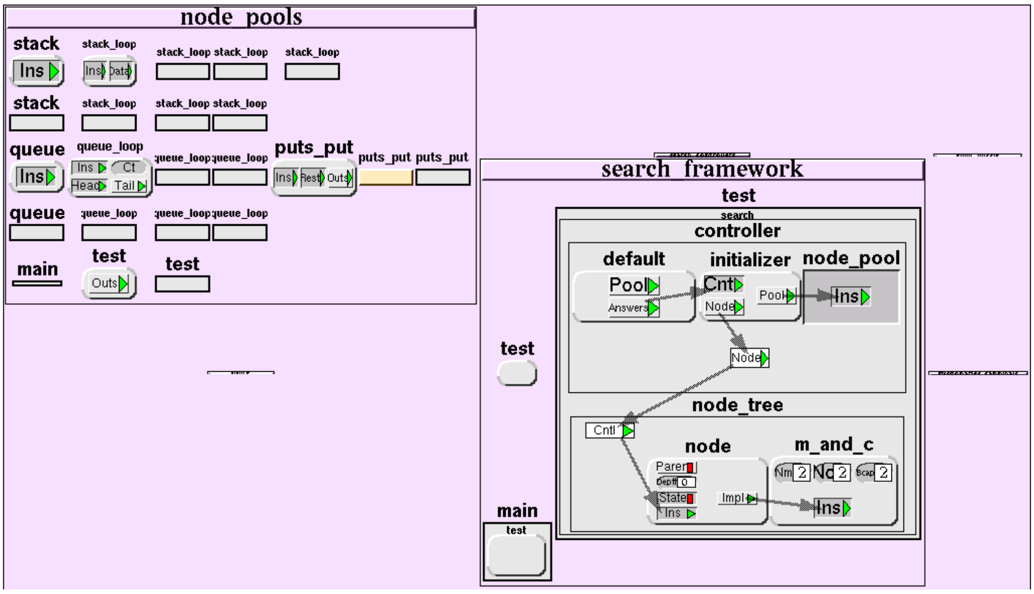


Figure 4.2: After navigation for editing

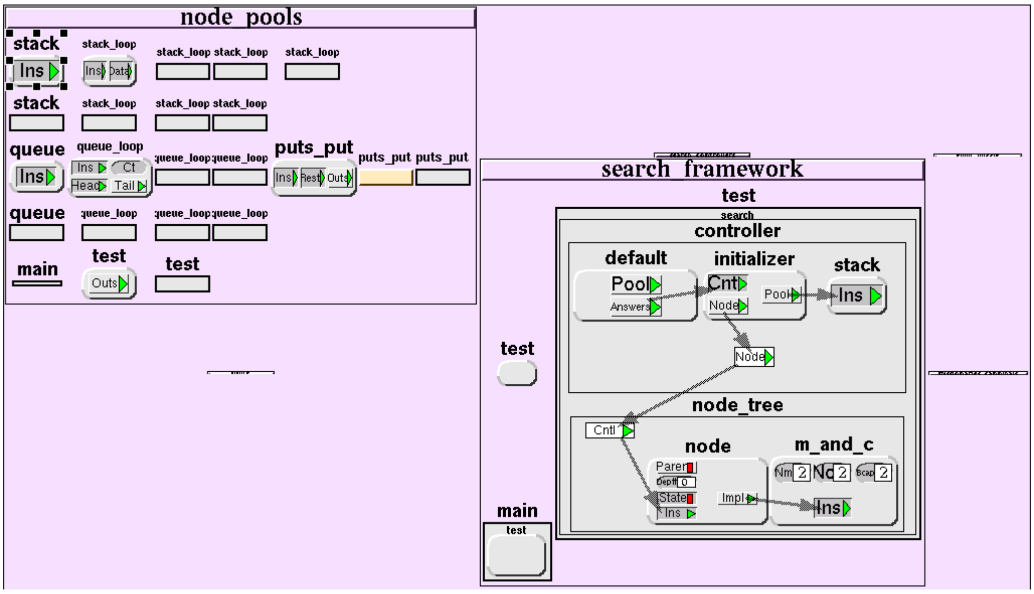


Figure 4.3: After editing by drag-and-dropping

In the editing phase, the user drags the stack process on the top-left of **node_pools** to the placeholder node_pool. Figure 4.3 shows the result.

Of these two processes, navigation for editing is more important. Issues concerning the editing process are almost the same as those in common diagram editors. It is in the process of navigation for editing that issues peculiar to distortion views arise.

During navigation for editing, it is important that the user can quickly obtain multi-focus distortion views. Multiple foci are necessary in many editing situations such as when dragging information from one node to another, editing one node comparing to other nodes, and creating a hyperlink between two nodes. Since the set of nodes required changes frequently according to various editing situations, the user has to focus and defocus multiple nodes whenever a change occurs.

To obtain an objective distortion view, the user must find the required nodes, magnify them to appropriate sizes, and shrink unnecessary nodes. In common hypertext editors, hyperlink navigation and search functions are used to find objective nodes. It is important to support these functions in editors with distortion views. Here are important issues for the support of these functions:

Hyperlink Navigation The system should allow users to navigate information space by following hyperlinks. That is, when the user follows a hyperlink by clicking on the anchor, the system should automatically focus the destination to an appropriate size.

This support is necessary, because destination nodes may be invisible or squashed, even in multi-focus distortion views, and it is difficult to locate them in such cases.

Search Functions Search functions are necessary, because it is difficult and time-consuming task to find objective data within a large information space by resizing and hyperlink navigation. The system should magnify objective nodes to appropriate sizes, so that the user can quickly edit these nodes.

4.3 Problems with Existing Distortion Views

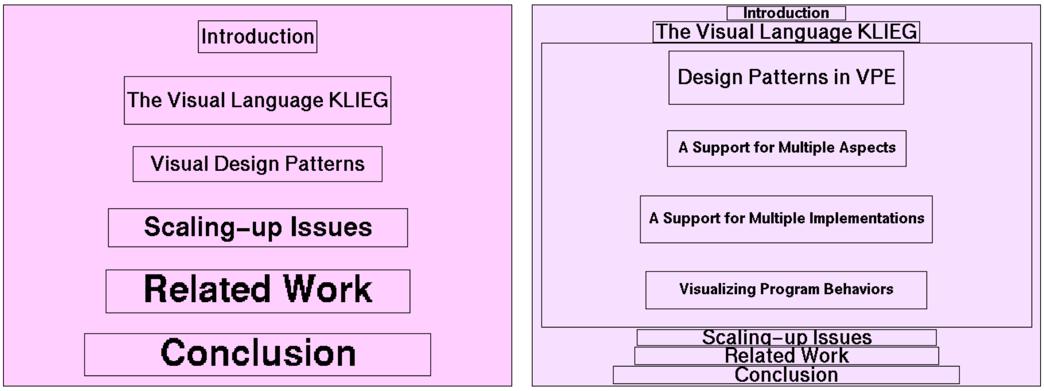
Although there have been various distortion view techniques, it is difficult to use them for editing nested networks. In the first place, early techniques, such as the perspective wall[33] and graphical fisheye views[48], are useless for editing, because they support only single focus views. Moreover, supporting multi-focus is not sufficient. In this section, we discuss general problems in existing distortion view techniques and explain what makes the editing task complicated and time-consuming.

4.3.1 Problem with Sizes of Nodes

The user can resize a node to arbitrary sizes during editing, but it is tedious to focus and defocus the node by resizing every time its contents are edited. Such repeated resizing prolongs the navigation time required when editing.

It might be useful if the system allowed the user to focus and defocus a node to appropriate sizes in simple operations, such as the open/close operations in the continuous zoom[5]. Such automatic focusing is also necessary to support hyperlink navigation. When the user follows a hyperlink, the system should focus the destination node of the hyperlink at an appropriate size.

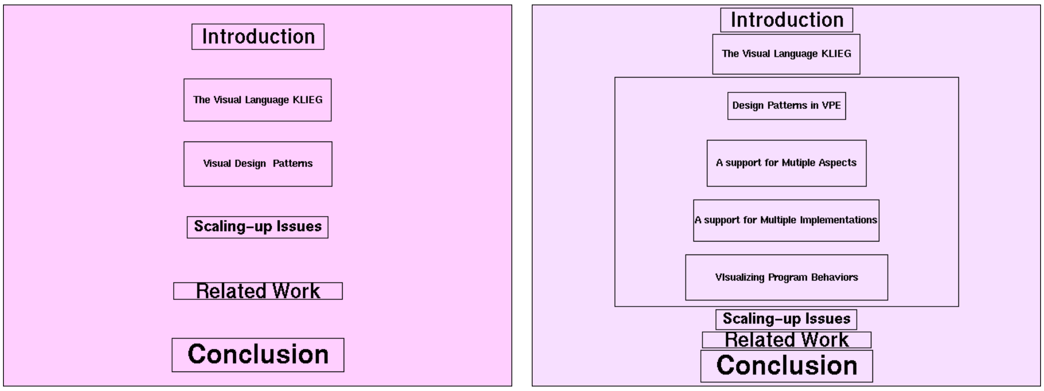
A set of appropriate node sizes for focusing and defocusing differs between nodes, according to contents of the node and the context in which the node



Introduction
The Visual Language KLIEG
Visual Design Patterns
Scaling-up Issues
Related Work
Conclusion

Introduction
The Visual Language KLIEG
Design Patterns in VPE
A Support for Multiple Aspects
A Support for Multiple Implementations
Visualizing Program Behaviors
Scaling-up Issues
Related Work
Conclusion

Figure 4.4: One sample presentation created by one test user



Introduction
The Visual Language KLIEG
Visual Design Patterns
Scaling-up Issues
Related Work
Conclusion

Introduction
The Visual Language KLIEG
Design Patterns in VPE
A support for Mutiple Aspects
A support for Multiple Implementations
Visualizing Program Behaviors
Scaling-up Issues
Related Work
Conclusion

Figure 4.5: Another sample presentation created by an another test user

is placed. These sizes are important for applications such as a presentation tool, in which the user creates a document to show other people.

It would be a tedious and repetitive for the user to explicitly designate the sizes of each node. Therefore it is desirable that the system automatically decides the size of a node from its contents and from the context. However, automatic designation is difficult, because the size also depends on the user's preference. We show the difference of node sizes caused by user's preference in Figures 4.4 and 4.5. They are two presentations with the same contents, created by two users. One of slides is focused in the right snapshot in each figure. In spite of having the same contents, the sizes of slides, both when focused and defocused, differ in the two presentations.

4.3.2 Problem with Hyperlink Navigation

The layout that is desirable after following a hyperlink may vary according to the current editing situation. When the user follows a hyperlink, it is always necessary to focus the destination of the link, but the source and other foci may or may not be necessary. On the one hand, if the user still wants to edit the source, both the source and the destination should be focused. On the other hand, after the user has finished editing the source, it is not necessary to retain the focus on the source.

As an example, we show a simple navigation task using a presentation tool in Figure 4.6. If viewing slides consecutively, the focus on the current slide will become unnecessary when the user follows a hyperlink. In Figure 4.6 (b), the system should automatically shrink the title slide when the user follows the hyperlink to the slide "Our Goal." In Figure 4.6 (c), the slide "Our Goal" should be shrunken in the same way.

Figure 4.7 shows another navigation example in a visual programming environment. In case of editing visual programs, it is necessary to retain foci on nodes that are being edited. In Figure 4.7 (a), the user is editing a data-flow diagram **master** at the center of the bottom-left module, and intends to check the behavior of the **pass_answers** component by following the hyperlink to its definition. In this case, the system should retain the focus on **master** when the user follows the hyperlink (Figure 4.7 (b).)

Although Pad++[7] and SHriMP[58] support hyperlink navigation, they only move the focus to the destination of a hyperlink, and do not address this problem. Therefore, when the source is necessary, the user must keep the previous focus on the source using tools such as portals in Pad++. If the

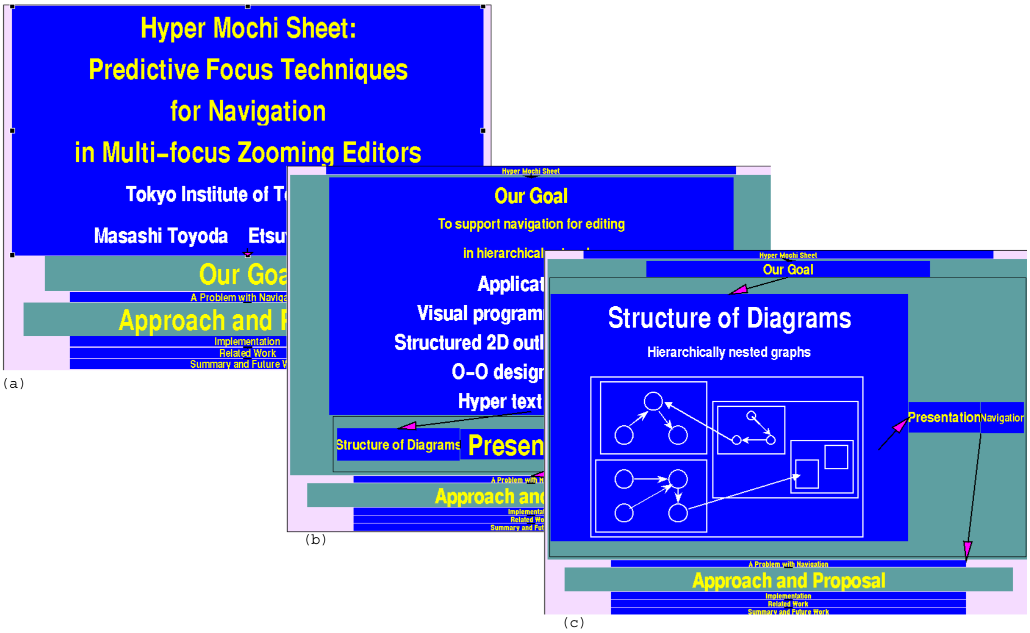


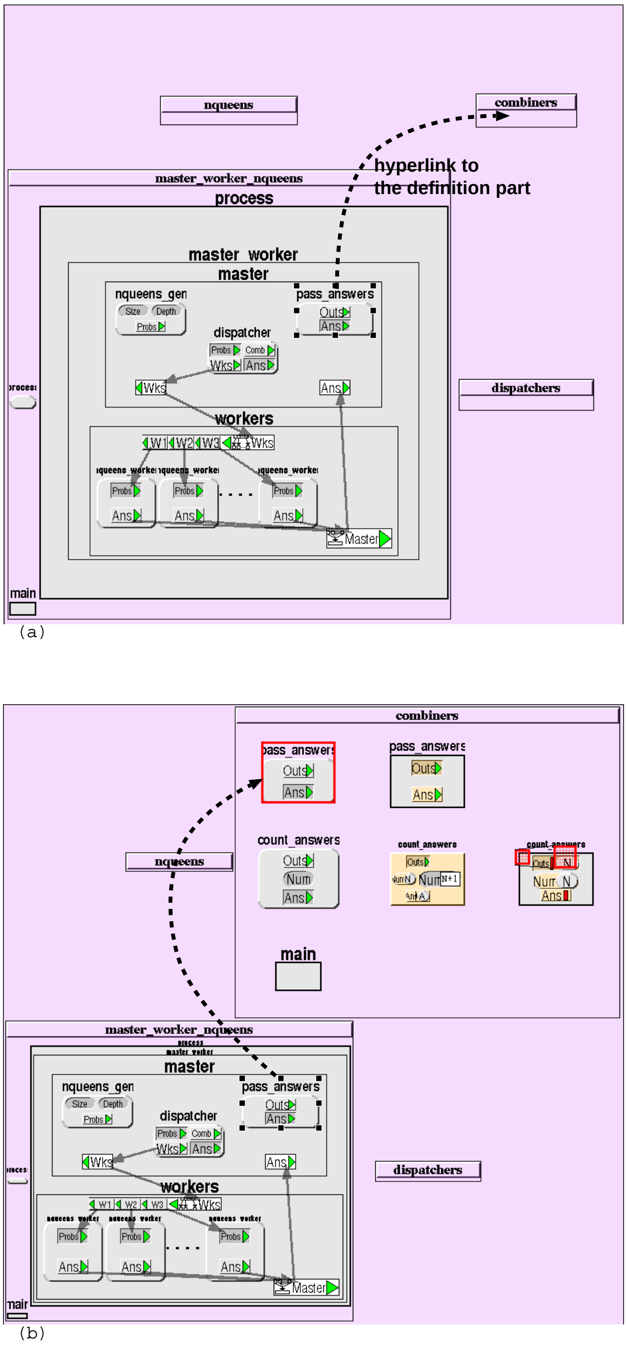
Figure 4.6: A hyperlink navigation in our presentation tool

system were to always keep the focus on the source as the default, the user would have to manually defocus unnecessary nodes after following the link. These are tedious and time-consuming tasks.

In addition, the system could provide multiple commands for following a hyperlink, so that the user can select a layout from multiple candidates. In this case, however, the user should consider which layout is appropriate for the current editing situation before following the hyperlink, and must select an appropriate command. This is cognitive overload.

4.3.3 Problem with Search

When editing, the search function is frequently used to find components to edit. It is important that the system can quickly provide distortion views in which objective data is focused. Though some distortion views support search functions, obtaining such distortion views is time-consuming for the user because of the way search results are displayed. Search results are shown using highlighting in most distortion views with search functions, such as Treemaps[20] and Dynamic Fisheye Views[43]. Therefore, to edit the result



(a)

(b)

Figure 4.7: A hyperlink navigation in a visual programming environment

of a search the user must magnify the appropriate highlighted part manually.

Another way to show search results is by list presentation. For example, in Pad++[7], the user first invokes a search dialog box and inputs keyword into a text input box; the result of the search is shown as the list of node identifiers. When the user clicks one of these identifiers, the focus is moved to the corresponding object. This is an indirect and time-consuming interface. It is also difficult to find objective data from numerous candidates.

4.4 Principles of Desirable User Interfaces

Essentially each problem concerns trade-off between freedom of distortion layouts and an increase in the number of command invocations. To solve these problems, it is important to keep distortion layout free and flexible and to reduce the increase of command invocations through automatic focus management. We show the principles of desirable distortion view interfaces in the following.

- Distortion layouts must be free and flexible

  We must avoid restricting distortion layouts (e.g., to single focus) when simplifying user interfaces. Support for multiple foci and magnification of focal points is indispensable.

- Automate boring, repetitive work

  Boring, repetitive work can be reduced by automatic invocation of commands. A history of command invocations can be used as a key to predict timing of command invocation. The prediction algorithm should be constructed based on user tests, and not on ad-hoc heuristics. This principle is relevant to the problem with sizes of nodes.

- Automate multi-focus management

  The work needed for layout adjustment should be reduced by automatic multi-focus management, which automatically maintains necessary foci and defocuses unnecessary foci. Since the need for foci may depend on application semantics, it must be easy for the application programmer to introduce the semantics to automatic management behaviors. This principle is relevant to the problem with hyperlink navigation.

- Reduce the time needed to switch from searching to editing

  The user should be able to quickly obtain a suitable editing layout from a search result. That is the search result should be not only highlighted but also automatically focused immediately. This principle is relevant to the problem with search.

Though there are some attempts to manage multiple foci automatically, the purpose of their focus management is different from ours. The Intelligent Zoom[6] and the data-driven magnification[24] manage multiple foci based on real-time data, such as network conditions and air traffic data, and uses zooming effects to alert the user to situations. It is easy to know the component to magnify in such applications. However, the purpose of our automatic focus management is to decrease user's editing tasks by automatically discarding unnecessary foci. This needs more heuristics than systems such as the network management system or air traffic control system.

Chapter 5

Interaction Techniques for Focus Management

Based on the principles in Chapter 4, we propose three interaction techniques for automatic multi-focus management in the following.

- *Focus size prediction* automatically determines a pair of node size, one being used when the node is focused and the other being used when the node is defocused. It is not necessary for the user to explicitly set these sizes, rather, our technique predicts appropriate sizes of nodes using a history of editing commands.

- *Predictive focus selection* automatically selects necessary foci and discards unnecessary foci during navigation with hyperlinks. When focusing and defocusing nodes, this technique uses sizes predicted by the focus size prediction. Since necessities of foci may depend on application semantics, Hyper Mochi Sheet provides a default focusing behavior that can be customized by the application programmer.

- *Dynamic query* Dynamic query function allows the user to easily find scattering objective data from hierarchical information, and to quickly obtain a suitable editing layout from a search result. It shows search results clearly by automatically magnifying nodes matched to a query, using node sizes predicted by the focus size prediction. The transitions of layouts occur dynamically during keyword inputs. Hyper Mochi Sheet provides a default search engine that can be customized by the application programmer.

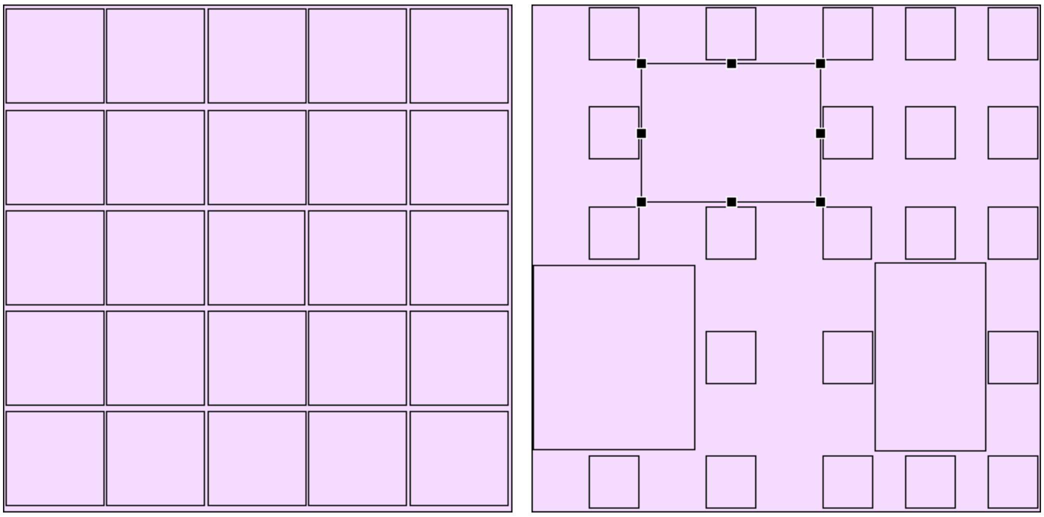


Figure 5.1: A distortion view in Hyper Mochi Sheet

In this chapter, we first introduce the basic user interface of Hyper Mochi Sheet. Then we describe three interaction techniques: focus size prediction, predictive focus selection, and dynamic query.

5.1 Basic User Interface

To generate distortion views, we use the Mochi Sheet algorithm[67, 68], which is similar to the continuous zoom[5]. Figure 5.1 displays an application of our approach to a 2D grid graph. When some nodes are magnified in the left view, it becomes impossible to display all nodes in their desirable sizes on the screen. In this case, all nodes are compressed uniformly in the horizontal and vertical directions keeping relative positions of nodes as the right view.

In addition to the continuous zoom algorithm, our algorithm avoids overlapping of nodes by simply aligning nodes in the horizontal and vertical directions during moving and resizing nodes. To use screen space more efficiently, it also meshes adjoining rows or columns together. For example, in the right view of Figure 5.1, two columns in the left are meshed together.

In the following, we describe the basic user interface of Hyper Mochi Sheet, then explain details of the Mochi Sheet algorithm.

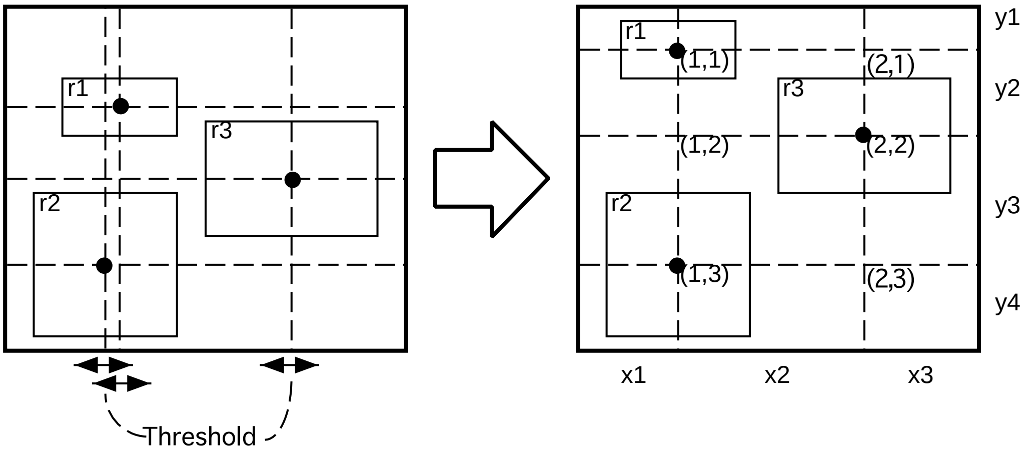


Figure 5.2: Generating alignment lines and arranging nodes

5.1.1 User Interface

The user can focus and defocus nodes by stretching and shrinking them with handles that are shown as small black rectangles in Figure 5.1. The width and height of a node can be stretched independently to each direction. Multiple nodes can be stretched simultaneously by selecting multiple nodes. All selected nodes become the same size as a node stretched by a handle. In addition the user can move nodes by dragging.

We also use semantic zooming [44], which changes an amount of information of a node according to its size. For example, in a presentation tool, when a slide is small, we can see only its title. When a slide is large enough, we can see details of the slide.

5.1.2 Mochi Sheet Algorithm

The Mochi Sheet algorithm is based on a basic algorithm that determines a layout of child nodes in one parent node. The basic algorithm takes the size of the parent node, and positions and sizes of child nodes as inputs, and outputs appropriate positions and sizes of child nodes. By recursively applying the basic algorithm from the root node, the Mochi Sheet algorithm determines a layout of an entire hierarchy. The following is the procedure of the basic algorithm.

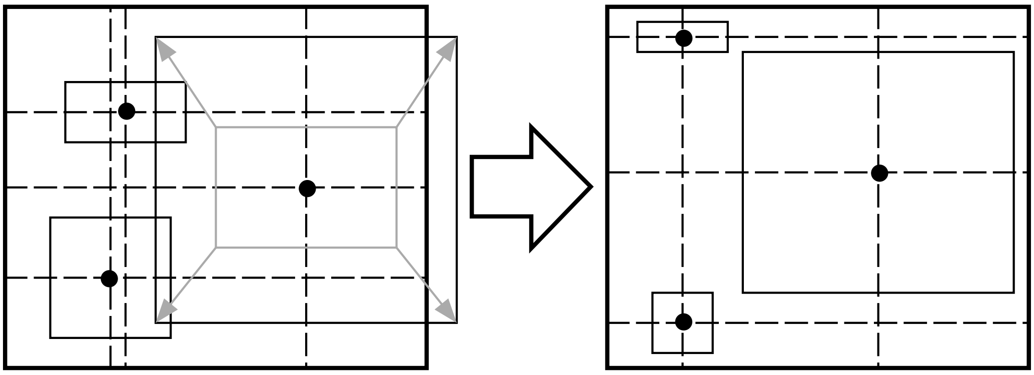


Figure 5.3: A layout when the required width is increased

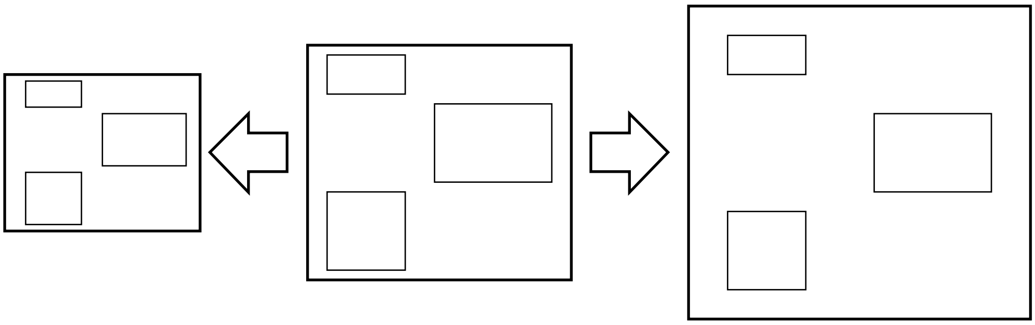


Figure 5.4: Distributing rest space

(1) Generate alignment lines First, a vertical and a horizontal lines are generated for each node. Both lines pass through the center of the node (The left hand side of Figure 5.2). Then two adjoining lines are merged, if the distance between these lines is shorter than a threshold, which can be defined for each parent node. In the right hand side of Figure 5.2, two vertical lines on the left are merged. Lines are not merged, if some nodes will be placed on the same intersection point as a result of merging.

(2) Calculate required sizes for intervals A required size for each interval (x1 to x3, and y1 to y4 in Figure 5.2) is calculated in order to avoid overlapping. The required size for a horizontal interval xk is:

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

where $W_{(i,j)}$ is the width of a node at (i, j). If there is no node at (i, j), $W_{(i,j)}$ is 0. The required size for a vertical interval is calculated similarly.

(3) Arrange nodes Given the required size for each interval, the total amount of space requested by children nodes is calculated for each direction:

$$R_x = \sum_k R_{xk} \quad R_y = \sum_k R_{yk}$$

When R_x is longer than the width of the parent node P_x, all horizontal intervals and width of all nodes are shrunken by P_x/R_x (Figure 5.3). When R_x is shorter than P_x, the rest space $P_x - R_x$ is distributed to the intervals, so that spaces for nodes will be as equal as possible (The right hand side of Figure 5.2). The same calculation is performed for the vertical direction. In particular, the size of a parent node is changed, the layout is changed as shown in Figure 5.4.

This procedure is performed when the parent node is modified by adding, deleting or moving child nodes. When nodes are resized, recalculation is performed from procedure (2), in order to animate continuously.

We can use space efficiently by meshing adjoining rows or columns together. Meshing is performed after the procedure (2), if R_x or R_y is longer than P_x or P_y, respectively. Figure 5.5 shows two meshing examples. Two

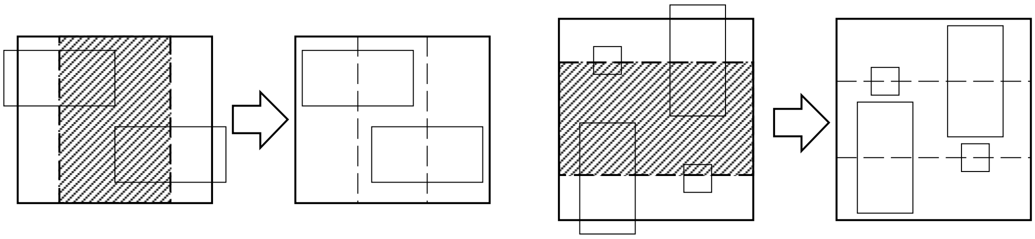


Figure 5.5: Meshing columns and rows

columns are meshed on the left example, and two rows are meshed on the right. These meshing are performed by shrinking a gray interval.

For meshing columns and rows, a shrunken size is first computed for each interval. The shrunken size of a horizontal interval xk is:

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

The shrunken size of a vertical interval is calculated similarly. Then horizontal and vertical intervals are sorted together by differences between a required size and a shrunken size in descendent order. The difference is calculated taking a scaling factor into account:

$$D_{xk} = (R_{xk} - S_{xk})\frac{P_x}{R_x} \qquad D_{yk} = (R_{yk} - S_{yk})\frac{P_y}{R_y}$$

Each interval is shrunken in this order, that is to replace R_{xk} or R_{yk} with S_{xk} or S_{yk}, respectively. During shrinking, some intervals will not be able to shrink because of causing overlap of nodes. In that case, these intervals are merely ignored.

5.2 Focus Size Prediction

Focus size prediction automatically determines a pair of node size[1] in the following.

- *Small size* is used when the node is defocused. The node area of this size is smaller than the large size. It is possible to edit inside roughly in this size.

[1] In the following, a size stands for a pair of width and height of a node.

- *Large size* is used when the node is focused. The user can edit inside details of the node. The node area of this size is larger than the small size.

The system predicts these sizes from a history of editing commands. It is not necessary for the user to explicitly set these sizes during editing. Once the small and large sizes of a node are determined, the user can easily select one of these sizes by clicking a mouse button or by using a popup-menu. Changes to the large size and the small size perform instant focusing and defocusing, respectively. These commands are useful when the user edits one node repeatedly and when the user navigates edited networks.

We do not provide any other intermediate sizes for the prediction, although they are useful in some situations. This decision simplifies size changing commands and makes the prediction easy but useful. Note that the determination of sizes is not trivial, because the user can resize nodes to arbitrary sizes in arbitrary orders during editing. For example, when the user stretches a node from its small size, it is difficult to distinguish whether the user wants to modify its small size or its large size.

5.2.1 Preliminary User Test

We performed a preliminary user test to investigate when the user determines small and large sizes during editing. We use the editor in which the user must set these sizes of each node explicitly. By tracing command histories, we tried to find out typical sequences of commands around size setting.

Method

- *System*: We used a simple editor for drawing nested nodes. The editor provides typical editing commands such as adding, removing, resizing, and moving nodes. For node size setting, it provides SetSmall and SetLarge commands that store the current node size as the small size and the large size, respectively. The editor also provides Small and Large commands for changing a node to the corresponding size.

- *Subjects*: Seven student volunteers and an instructor of computer science served as subjects in the user test. All subjects were familiar with typical window-based GUIs.

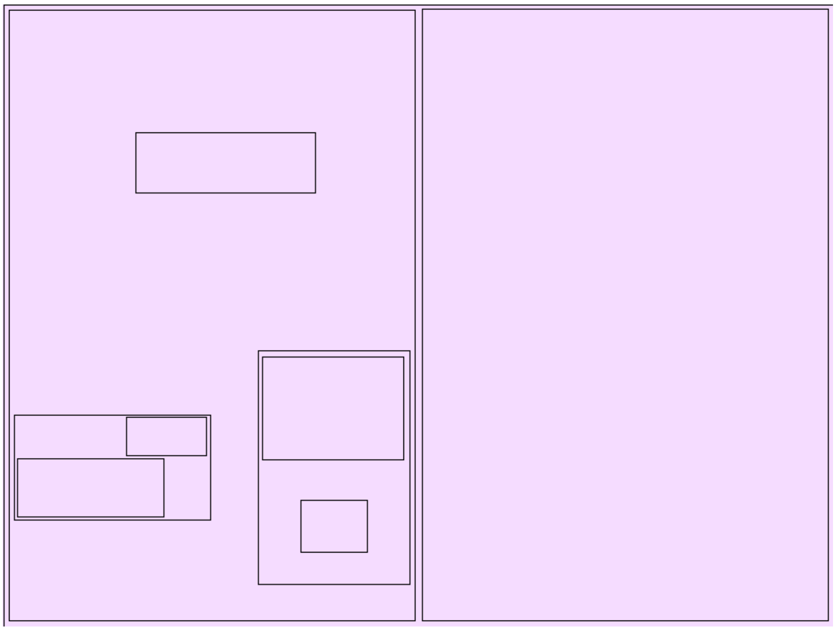


Figure 5.6: The diagram used in the user test: all rectangle sizes are set to small

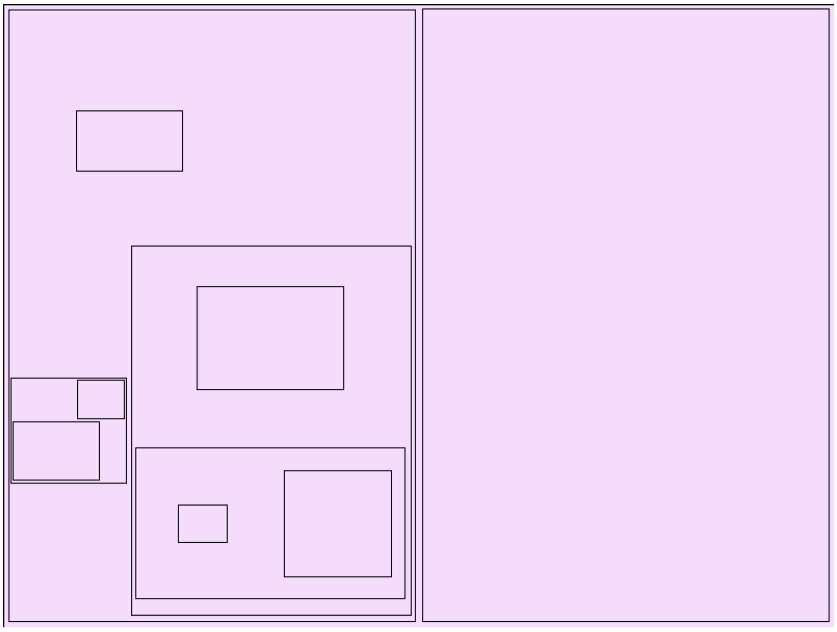


Figure 5.7: The diagram that shows the detailed view of the bottom-right rectangles

- *Task*: Subjects were required to draw a diagram, which is shown in the left hand side of Figure 5.6, on the right blank area, and to set small and large sizes of all nodes. This diagram consists of 13 nested rectangles[2], and sizes of each node have been set. The default size of each node is its small size. Figure 5.7 shows the diagram in which bottom-right rectangles are changed to its large size. Each subject was instructed to set sizes immediately when he decided sizes, and to edit without hurry. In addition, we did not limit the time for the task.

- *Procedure*: Before performing the task, subjects were given an explanation of the system and a practice trial on a part of the diagram. We spent about 10 minutes on this session.

Result and Observations

Tables 5.1 and 5.2 show patterns of command sequences around SetSmall and SetLarge respectively, and the number of times each pattern was used by each subject. A pattern begins when the node was in its small or large size after its creation[3] or changing its size. This is the initial size in the pattern and is followed by a command sequence performed on the node before the execution of set command. The pattern also includes a command performed after the set command.

We considered only resize related commands such as resizes (Shrink and Expand) and size changes (Small and Large), because we could not find distinctive regularity from other commands. Note that we treated consecutive resizes on a single node as a single resize command, since such a sequence stands for a fine tuning of the size.

We show observations of the result in the following.

1. SetSmall occurs after repeated Shrink from the small size in most cases (See the first pattern in Table 5.1).

2. SetSmall also occurs after a single Expand from the small size, and is mostly followed by Expand or Large (See the second pattern in Table 5.1). In some cases, SetSmall is followed by Small, but SetLarge occurs more frequently between Expand and Small (See the first pattern in Table 5.2).

[2] Some nodes are not displayed in Figure 5.6, because their parent nodes are too small

[3] A node is in small size at the creation time

Table 5.1: Command Sequences around SetSmall

Initial	Command Sequences	Subjects							
size	around SetSmall	1	2	3	4	5	6	7	8
small	Shrink+.**SetSmall**.except Shrink	5	6	5	5	3	5	9	2
	–	2	2	2	1	1	1		
small	Expand.**SetSmall**.Expand or Large	2	1	1	1	1	2	2	2
	Small	1	1	1					
	–			1	1	1			
large	Shrink+.**SetSmall**.Small						2		
	Large	1							
small	Expand.Shrink.**SetSmall**.Any					2			
Others							3	3	

Shrink: resize to a smaller size, Expand: resize to a larger size
Small: change to the small size, Large: change to the large size
+: one or more execution of the command
–: the node was left

Table 5.2: Command Sequences around SetLarge

Initial	Command Sequences	Subjects							
size	around SetLarge	1	2	3	4	5	6	7	8
small	*.Expand.**SetLarge**.Small	6	7	3	5	2	2	2	3
	Shrink					3			
large	Expand+.**SetLarge**.Small			1	1		6		
	Shrink+.**SetLarge**.Small			1			1	1	
Others		1		3		1	3	2	

*: alternative sequence of commands that may be empty

3. SetLarge mostly occurs after Expand and before Small (See Table 5.2). Both SetSmall and SetLarge may occur after Shrink from large size and before Small (See the third pattern in Table 5.1).

4. Among six SetLarge commands performed by the subject 5, three of them occur before Shrink. The subject first set the large size then shrunk and set the small size, though most subjects set the small size first. In the fourth pattern in Table 5.1, we can see this sequence before SetSmall.

5.2.2 Prediction Algorithm

Based on the above observations, we designed and implemented a size prediction algorithm. Our design policies are (1) to give a higher priority to patterns used by most subjects, (2) to satisfy subjects as fairly as possible, and (3) to keep the algorithm simple.

The algorithm is based on state transitions on each node that are shown in Figure 5.8. The state is changed when the user performs a command on the node, and a label on an arrow represents the command. There are three states: small, small.Ex, and large. The small and large states represent that the node is in the corresponding sizes, and small.Ex represents that the node has been expanded repeatedly from the small size. The small.Ex state is necessary, since the size is uncertain when the node is expanded from the small size (See the observation 2).

When a transition occurs, the small and large sizes (S_{small} and S_{large}) may be changed. In Figure 5.8, rectangles include actions performed after the transition. S_{pre} and S_{post} represent the sizes before and after the transition, respectively, and S_{tmp} represents the temporal store of a size. We describe the reason for each action in the following.

- *Shrink from small*: According to the observation 1, S_{small} is changed to S_{post}.

- *Expand from small*: Since S_{post} may be either size, S_{post} is stored temporary into S_{tmp}.

- *Expand or Large from small.Ex*: According to the observation 2, S_{small} is changed to S_{tmp}.

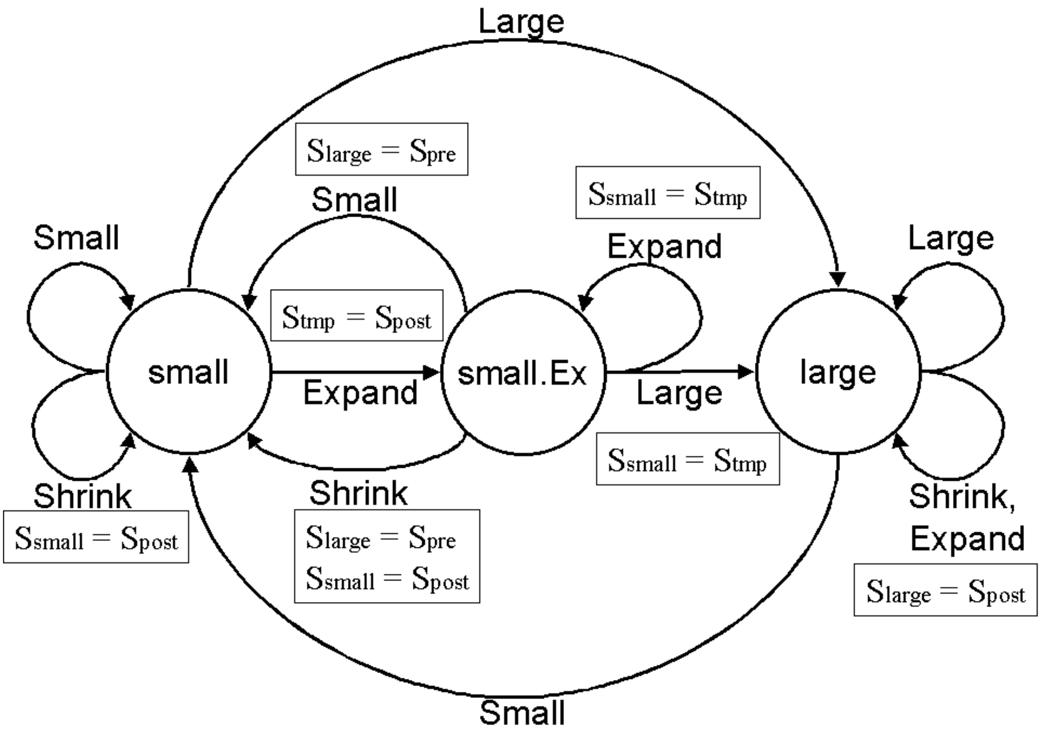


Figure 5.8: A state transition chart for predicting size

- *Small from small.Ex*: According to the observation 3, S_{large} is changed to S_{pre}.

- *Shrink from small.Ex*: We don't ignore the observation 4 to satisfy subjects fairly (This is policy 2). In fact, there are few conflicts with other observation. In this case, S_{large} and S_{small} are changed to S_{pre} and S_{post}, respectively.

- *Shrink or Expand from large*: According to the observation 3, S_{post} may be the small size in the case after Shrink from large. In this algorithm, S_{large} is changed to S_{post}, because there are three subjects who used SetLarge and is only one who used SetSmall. This decision follows the design policies 1 and 3.

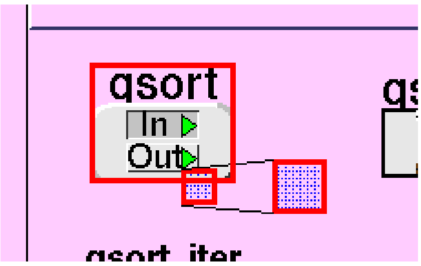


Figure 5.9: Size correction interface

5.2.3 Size Correction Interface

Since prediction may be error-prone, manual correction is necessary. We provide an interface to correct a size by choosing a size from the size history of the node. When the user performs a size changing command on a node, two buttons appear near the node (Figure 5.9). The smaller button changes the size to the next smaller size in the history and the larger button the next larger size. If the predicted size is acceptable, the user can ignore these buttons. This interface allows the user to correct size precisely to a past size rather than using handle interface.

5.2.4 Evaluation

In this section, we describe an experiment to evaluate the feasibility of the focus size prediction technique.

Method

- *System*: We used a simple presentation editor with the focus size prediction function. Differences from the editor in the preliminary user test are that a node has one line editable text inside, and that the editor does not provide size setting commands (SetSmall and SetLarge). A text in a node is not displayed when the node has child nodes and the node is large enough[4] to display its children.

- *Subjects*: Ten student volunteers served as subjects. Four of them were also ones of the preliminary user test. All subjects were familiar with

[4] A node is large enough if the width and height of the node are larger than 70 pixels.

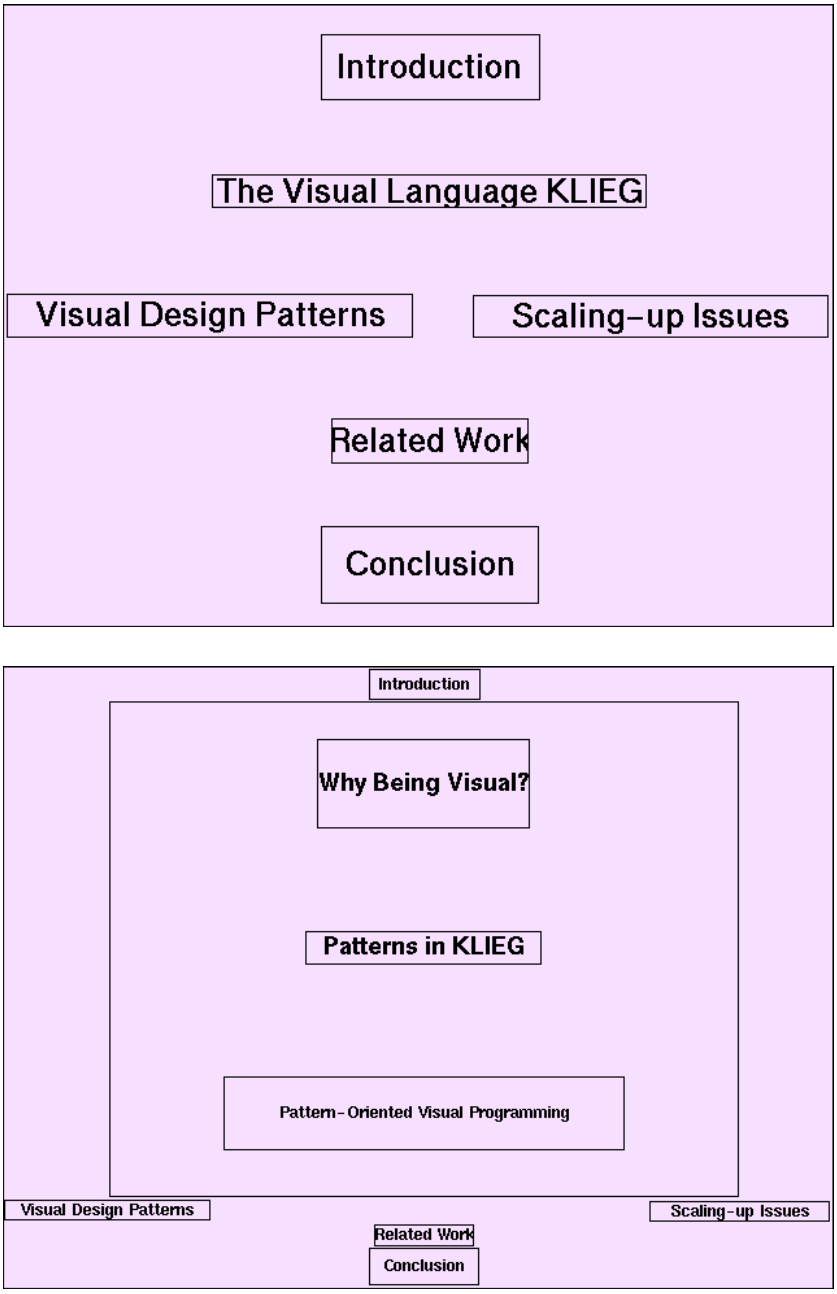
1 Introduction
2 The Visual Language KLIEG
2.1 Why Being Visual?
2.2 Patterns in KLIEG
2.2.1 Basic Usage
2.2.2 Hierarchical Constructions
2.3 Pattern-Oriented Visual Programming
3 Visual Design Patterns
3.1 Design Patterns in VPE
3.2 A Support for Multiple Aspects
3.3 A Support for Multiple Implementations
3.4 Visualizing Program Behaviors
4 Scaling-up Issues
4.1 The Zooming Interface of the KLIEG Tracer
4.2 The Zooming Interface of the KLIEG Editor
5 Related Work
6 Conclusion

Figure 5.10: The table of contents used in the experiment

typical window-based GUIs.

- *Task*: Subjects were required to edit a simple presentation based on the table of contents shown in Figure 5.10. Each subject was instructed (1) to represent the presentation hierarchy as nested nodes like Figure 5.11, (2) to put some empty text boxes as contents of each leaf section such as “1 Introduction” and “2.2.1 Basic Usage,” (3) to arrange nodes as you like, and (4) to edit without hurry and we did not limit the time for the task. In addition, we did not force for subjects to check node sizes during editing.

- *Procedure*: Before performing the task, subjects were given an explanation of the system and a practice trial on a part of the presentation. We spent about 10 minutes on this session. After each subject performed task, we checked whether sizes of each section are along to the subject’s intention. In this session, we asked subjects about correctness of sizes using Large and Small commands.



Introduction

The Visual Language KLIEG

Visual Design Patterns

Scaling−up Issues

Related Work

Conclusion

Introduction

Why Being Visual?

Patterns in KLIEG

Pattern−Oriented Visual Programming

Visual Design Patterns

Scaling−up Issues

Related Work

Conclusion

Figure 5.11: A sample presentation written by a subject

Table 5.3: The number of use of the size correction interface and the number of prediction errors checked after the task

		Subjects									
		1	2	3	4	5	6	7	8	9	10
# of size corrections on 17 sections	Small	1	4	1	0	3	1	0	3	4	3
	Large	1	0	1	0	2	1	0	4	0	0
# of errors in 17 sections	Small	2	0	0	0	1	0	0	0	0	0
	Large	2	0	3	0	0	3	3	0	3	3
total	Small	3	4	1	0	4	1	0	3	4	3
	Large	3	0	4	0	2	4	3	4	3	3

Result and Discussion

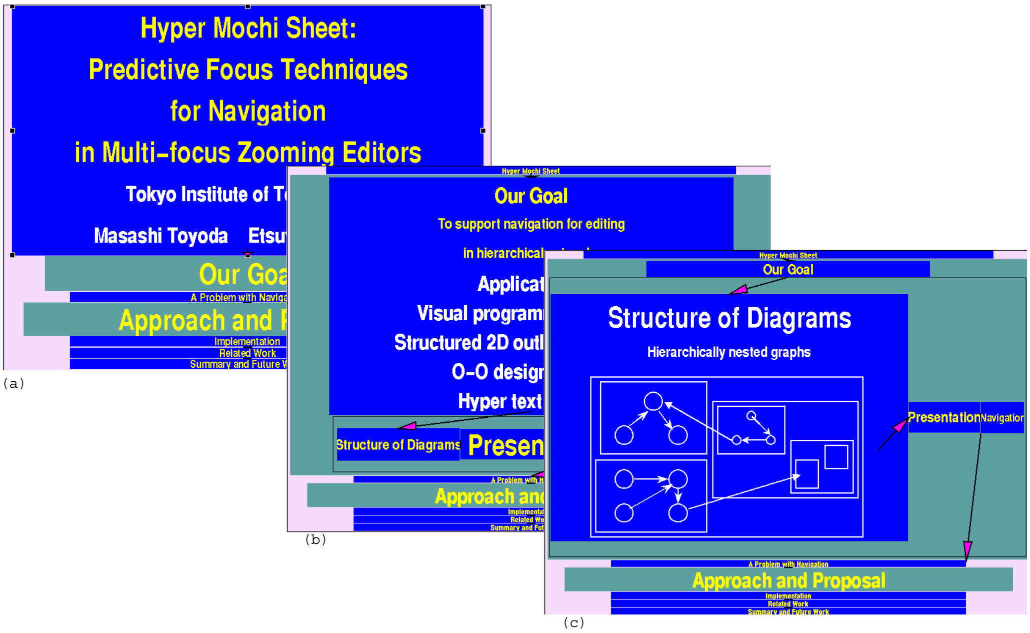
Table 5.3 shows the number of the use of the size correction interface, and the number of prediction errors. The use of the size correction interface means that a subject found and corrected a wrong node size, which was not along to the subject's intention, during editing. An error was counted when a wrong node size was found during the check session after the task. Each number was counted for each size. Subjects 1 to 4 were also ones of the preliminary user test, but there were no significant differences in the result from other subjects.

In spite of the fixed algorithm, error rates are significantly small. The average error ratio after the task is 6% (the best is 0% and the worst is 11%). Even in total error ratio, the average is only 14% and the worst is 20%.

Note that the prediction algorithm almost suits all subjects, though they edited the presentation in various manners. Some subjects resized nodes without using Small and Large command, and some subjects used Small and Large command on about half of the nodes. In addition, some subjects first decided a large size of a node, and other subjects decided a small size first.

5.3 Predictive Focus Selection

During navigation with hyperlinks, the system predicts foci that will be unnecessary, and automatically discards these foci. Focus selection enables the user to obtain almost desirable layout only by following hyperlinks.



Hyper Mochi Sheet:
Predictive Focus Techniques
for Navigation
in Multi-focus Zooming Editors
Tokyo Institute of T
Masashi Toyoda Etsu
Our Goa
A Problem with Naviga
Approach and P
Implementation
Related Work
Summary and Future W

Hyper Mochi Sheet
Our Goal
To support navigation for editing
in hierarchical
Applicat
Visual program
Structured 2D outl
O-O design
Hyper text
Structure of Diagrams
Presen
Approach and

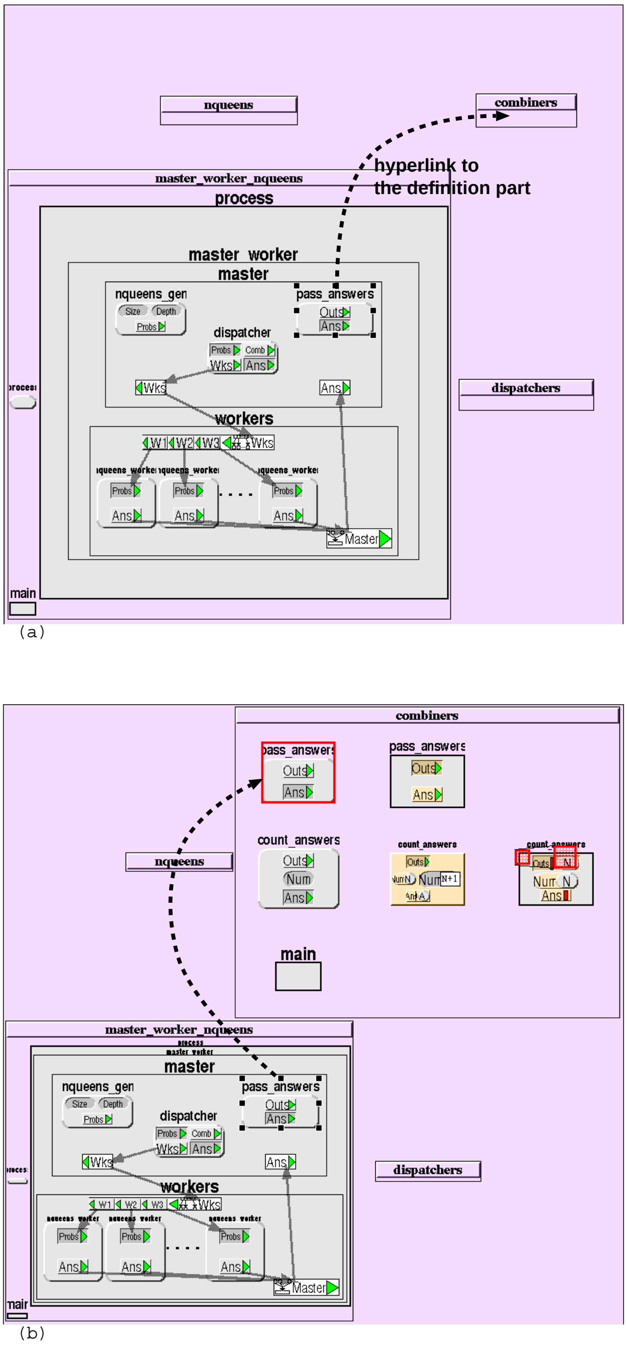
Hyper Mochi Sheet
Our Goal
Structure of Diagrams
Hierarchically nested graphs
Presentation
Navigation
A Problem with Navigation
Approach and Proposal
Implementation
Related Work
Summary and Future Work

Figure 5.12: A hyperlink navigation in our presentation tool

5.3.1 Hyperlink Navigation Examples

As an example, we show a simple navigation task using a presentation tool in Figure 5.12. In case of viewing slides one after another, the focus on the current slide will become unnecessary when the user follows a hyperlink. In Figure 5.12 (b), the system automatically shrinks the title slide when the user follows the hyperlink to the slide “Our Goal.” In Figure 5.12 (c), the slide “Our Goal” is shrunken in the same way.

Figure 5.13 shows another navigation example in a visual programming environment. In case of editing visual programs, it is necessary to retain foci on nodes that are in the middle of editing. In Figure 5.13 (a), the user is editing a data-flow diagram **master** at the center of the bottom-left module, and intends to check the behavior of the **pass_answers** component by following the hyperlink to its definition part. In this case, the system can predict that **master** is still necessary because there are unconnected components in the network. Therefore, the system retains the focus on **master** when the user follows the hyperlink (Figure 5.13 (b).)



(a)

(b)

Figure 5.13: A hyperlink navigation in a visual programming environment

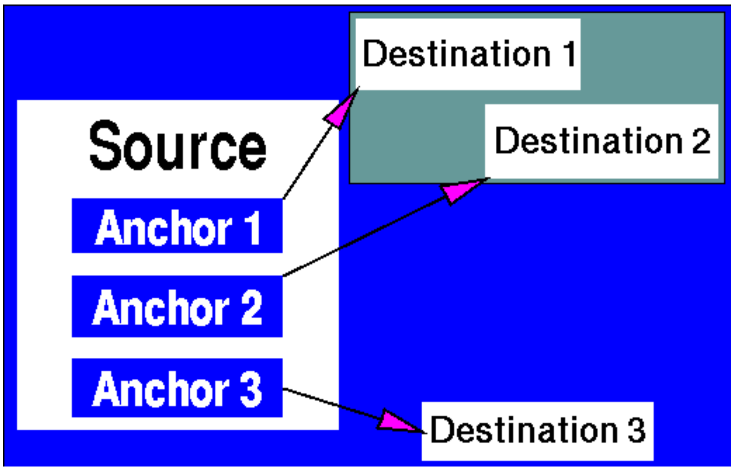


Figure 5.14: Hyperlink structure

5.3.2 Prediction Method

To realize such automatic focus management, Hyper Mochi Sheet library provides each node with a boolean function $f(F)$, which returns true if the focused (large size) node F is still necessary. Programmers can reflect application semantics in their applications by defining customized $f(F)$ for each node. For example, in visual programming editor, $f(F)$ returns true if there exist unconnected ports in the node F. The default implementation of $f(F)$ returns false if all child nodes in the F are in the small size.

When the user follows a hyperlink from an *anchor* (See Figure 5.14), the system changes the size of the *destination* to its large size, and stores the *source* and the destination in the *focus list*. Simultaneously, the system changes the sizes of all the ancestor nodes of the destination to their large size in parent-to-child order, so that the destination will be visible. After magnifying the destination, the system checks whether each node except the destination in the focus list satisfies f. If f returns false with a node in the list, the node size is changed to its small size. Then the system changes the sizes of ancestors in child-to-parent order. Before changing the size of an ancestor A, the system checks $f(A)$. If $f(A)$ returns false, A is changed to its small size, and if not, the system stops changing sizes of upper ancestors.

In addition, the system animates transition from one layout to another, so that the user is not confused even if the layout drastically changes during navigation.

5.4 Dynamic Query

The dynamic query function of Hyper Mochi Sheet allows the user to easily find scattering objective data from a set of textual data that is constructed hierarchically such as Yahoo![1], and class libraries of Java[2] and C++[59]. This shows search results clearly by magnifying nodes matched to a query. The transitions of layouts occur dynamically during keyword inputs.

The function provides following four methods to search hierarchical textual data. These methods can be performed with only two operations, typing keywords and selecting nodes.

- Exploring a hierarchy from the root.
- Scanning results of full-text pattern matching.
- Narrowing search space by exploration then performing pattern matching.
- First performing full-text pattern matching then narrowing search space to some sub-trees that includes the results. This process is performed repeatedly.

It is important that our query function facilitates the fourth method, which overcomes problems of the first three methods. The first method often causes many and deep backtracks. The result of the second method may include too numerous matches to find objective data. The third method may miss finding scattering multiple objects in a hierarchy. Using the fourth method, the user can find scattering multiple objects without both deep backtracks and scanning numerous results.

In addition, Hyper Mochi Sheet provides a default search engine for the pattern matching that can be customized by the application programmer. As a default search engine, we use a dynamic approximate string matching technique.

In the following, we use a directory editor, HishiMochi[63], as an example of our dynamic query function. Figure 5.15 shows a screen snapshot of HishiMochi. It visualizes a part of directories in Yahoo! as nested rectangles. The

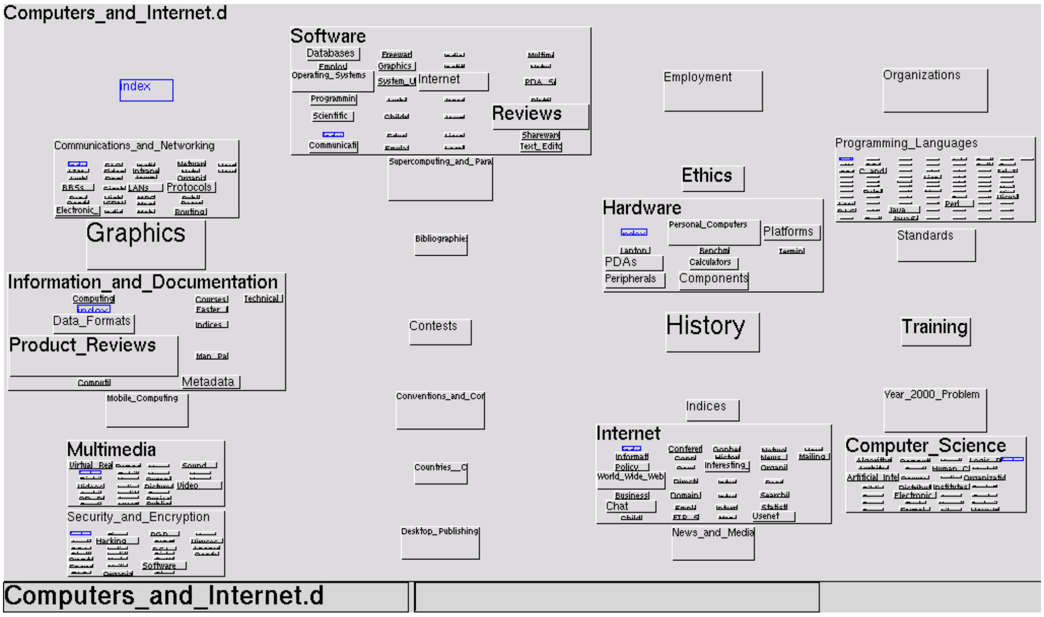


Figure 5.15: The initial screen dump of Computers & Internet directory of Yahoo!

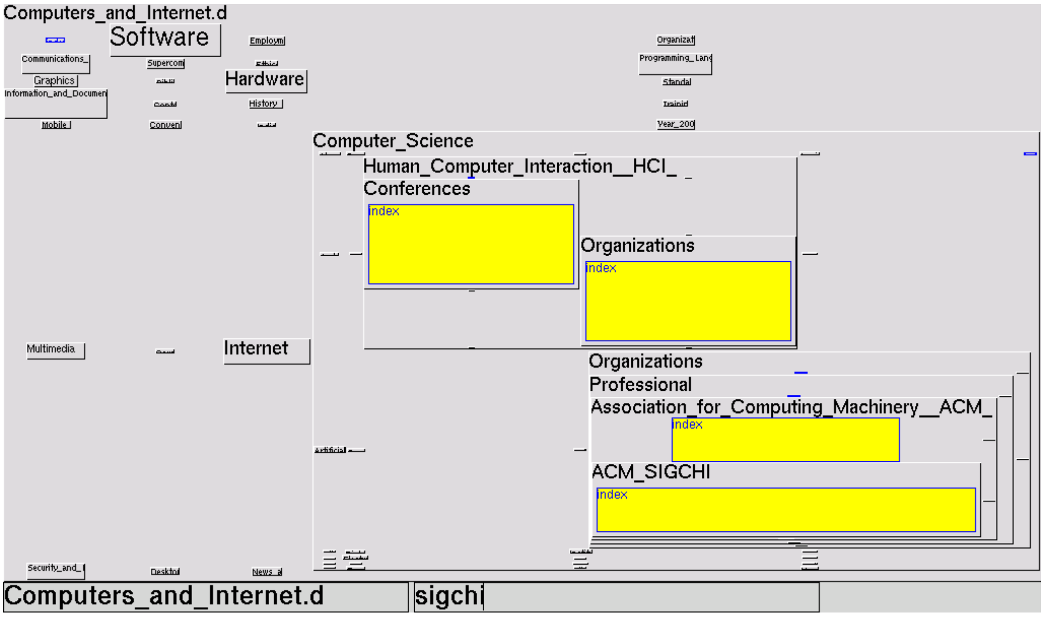


Figure 5.16: Searching with a keyword "sigchi"

size of each directory represents the number of files below the directory. The user performs queries by typing in keywords into the right-bottom text box. Whenever the user types in a character, the system matches text lines of all files with the keyword, magnifies found file with animation, and highlights matched lines. Figure 5.16 is the result of the query, “sigchi.”

5.4.1 Details of the Dynamic Query Function

Displaying Results

The sizes of nodes matched to a query are changed automatically to their large size. Simultaneously, all ancestors of matched nodes are also changed to their large sizes. Then the other nodes are changed to their small sizes. These changes of layouts occur whenever the user types in a character into the keyword box. In addition, the system animates these changes, so that the user is not confused during transition of layouts.

Dynamic Approximate String Matching

As a default search engine, we adopt a dynamic approximate string matching technique, which is used for searching a dictionary in the pen-based text input method, POBox[34]. First, it performs exact pattern matching. If the keyword specified by the user does not match any text lines in nodes, the system automatically performs approximate string matching that allows errors in spelling. For example, the keywords, “virtial” and “virtal” match “virtual” with the allowance of one misspelling. The allowance of misspelling is gradually increased till at least one candidate is found (the maximum allowance is three misspells.) The detail of the algorithm is described in [34].

This approximate matching feature is useful when the user does not know the correct spelling of the objective data. We also provide a feedback of misspelling. When the misspell allowance is increased, the background color of the screen becomes darker and darker. Therefore, the user can notice whether the current keyword is appropriate during typing the keyword.

In addition, this string matching feature also handles a simple wildcard character represented by the “`.*`” pattern. In order to find keywords in the middle of a line, the system attaches “`.*`” on the head and the tail of the keywords, and replaces blank characters to the “`.*`” pattern. For example, the keyword, “virtual reality” is replaced to the “`.*virtual.*reality.*`”

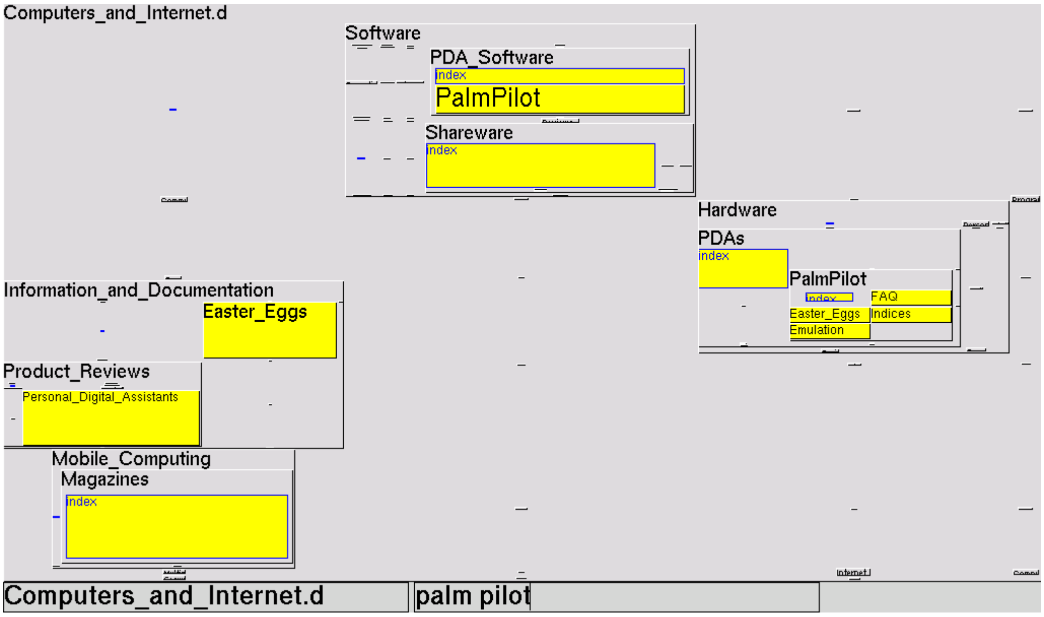


Figure 5.17: The result of search with keywords “palm pilot”

pattern.

5.4.2 Examples

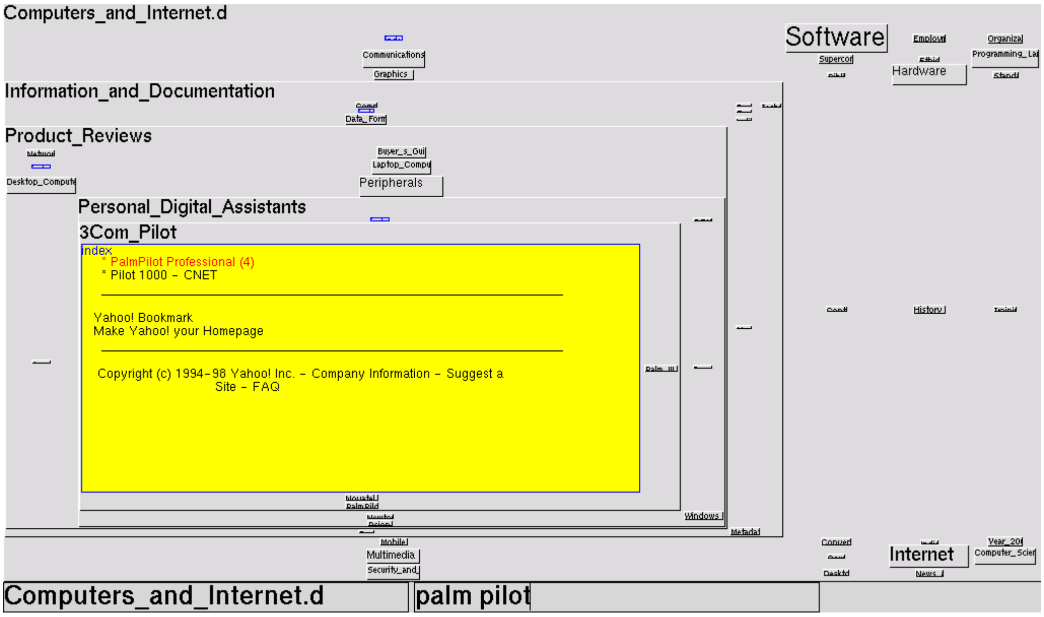
In this section, we show some examples using our directory editor with dynamic query interface.

Yahoo!

In this example, we show that our dynamic query interface allows the user to efficiently find objective data scattering about a large hierarchical information space. As an example, we use a part of directories in Yahoo!.

> **Scenario:** The user purchased a PalmPilot[5], and want to collect information to handle this device.

[5] PalmPilot is a registered trademark of 3Com Corporation.



index
* PalmPilot Professional (4)
* Pilot 1000 - CNET

Yahoo! Bookmark
Make Yahoo! your Homepage

Copyright (c) 1994-98 Yahoo! Inc. - Company Information - Suggest a Site - FAQ

Figure 5.18: Looking matched documents one by one

First, the user types in keywords, “palm pilot.” The result of this query is shown in Figure 5.17. It is clearly shown that there are matched documents in directories such as Software and Hardware.

By pushing ‘Enter’ key repeatedly, the user can look matched documents consecutively (Figure 5.18). Each leaf rectangle is a document browser, in which the user can scroll texts by cursor keys. In the browser, matched lines are highlighted with red colored texts.

The user can narrow search space by a direct manipulation technique. This interface is useful when the number of the candidates is too much to look consecutively. To narrow search space to some sub-directories, the user selects these directories using a mouse. The eight small black handles are attached on the selected directories and browsers. Selection task is easy for the user, since matched parts are magnified in the editor.

In this case, the user selects some directories in Software and Hardware directories. Product_Reviews and Magazines directories are ignored, since these have nothing to do with the user’s purpose. After the selection, the user types in ‘Space’ to search selected directories with the same keywords. Figure 5.19 is the result.

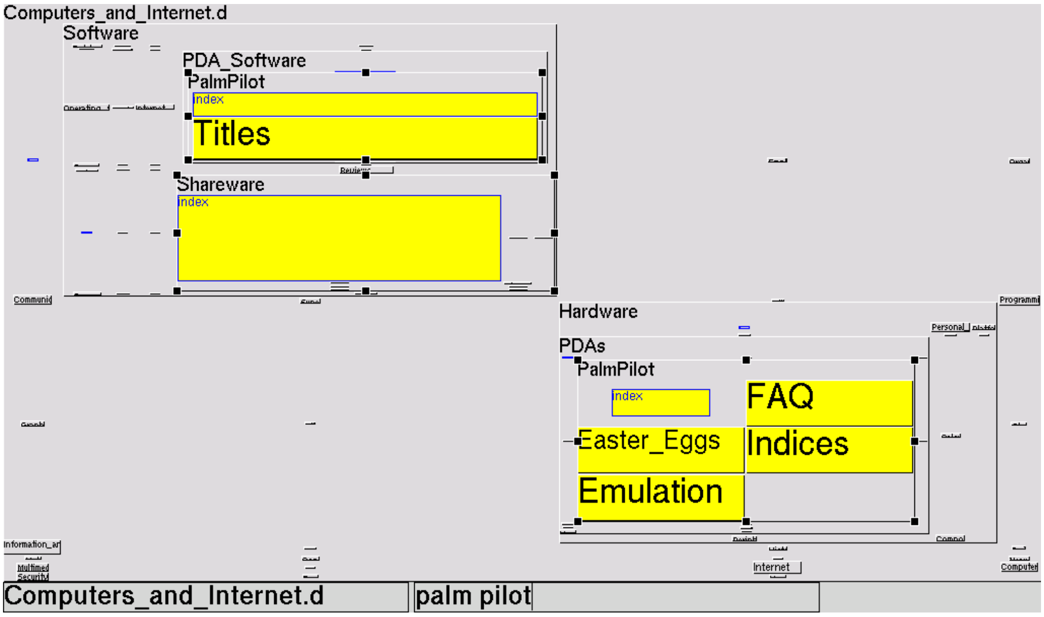
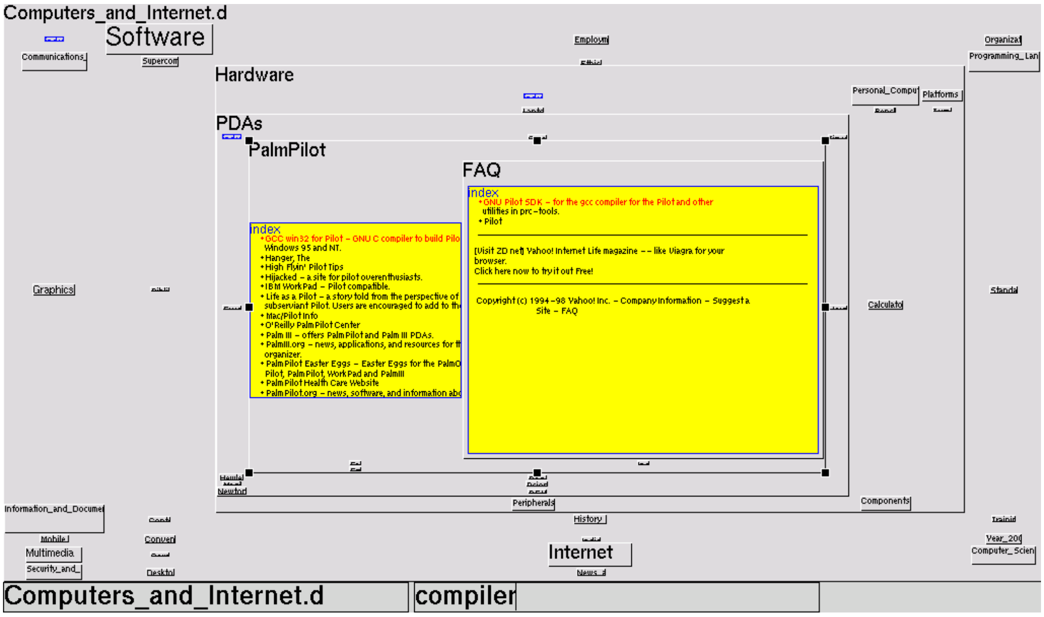


Figure 5.19: Narrowing search space



- GCC win32 for Pilot – GNU C compiler to build Pilo
  Windows 95 and NT.
- Hanger, The
- High Flyin' Pilot Tips
- Hijacked – a site for pilot overenthusiasts.
- IBM WorkPad – Pilot compatible.
- Life as a Pilot – a story told from the perspective of
  subserviant Pilot. Users are encouraged to add to th
- Mac/Pilot Info
- O'Reilly PalmPilot Center
- Palm III – offers PalmPilot and Palm III PDAs.
- PalmIII.org – news, applications, and resources for th
  organizer.
- PalmPilot Easter Eggs – Easter Eggs for the PalmO
  Pilot, PalmPilot, WorkPad and PalmIII
- PalmPilot Health Care Website
- PalmPilot.org – news, software, and information abo

- GNU Pilot SDK – for the gcc compiler for the Pilot and other
  utilities in prc–tools.
- Pilot

[Visit ZD net] Yahoo! Internet Life magazine –– like Viagra for your
browser.
Click here now to try it out Free!

Copyright (c) 1994–98 Yahoo! Inc. – Company Information – Suggest a
Site – FAQ

Figure 5.20: Search with a new keyword “compiler” keeping the selections

Since the selected directories and browsers are kept magnified, the user can search these directories with new keywords. This allows the user to perform a kind of AND search, that is to find a document that includes lines matched to previous keywords and lines matched to new keywords. For example, in Figure 5.20, the user types in the keyword "compiler" to find compilers for PalmPilot.

In addition, the user can make personal bookmarks by making a new directory and store the results of the search by drag-and-dropping browsers into the directory.

Novels

The second example is a collection of novels written by the same author. We use full-texts of Sherlock Holmes stories. We classify them by written year (Figure 5.21), and sort first top to bottom then left to right.

The user can easily perform simple data mining such as finding when a character appears, and how a case is referred from subsequent stories. Such an exploration can be done on data-bases of research papers and news articles.

Figure 5.22 shows a result of a search with keyword "moriarty". The result clearly shows which stories and when professor Moriarty appears.

Photo Browser

Figure 5.23 shows the third example, a photo browser. The user can easily find photos, and can perform arrangement of photos. In Figure 5.23, each photo is attached keywords, such as a date, and the names of persons and things in the photo. Then the user can find photos with these keywords.

5.4.3 Variation of Search Method

Since the user can perform keyword inputs and node selection in an arbitrary order, all the methods listed in the beginning of Section 5.4 are available as follows.

- Exploring a hierarchy can be done by selecting a node and typing in 'Space.' Backtrack is simply done by selecting a parent node.
- Scanning a result of full-text pattern matching can be done by typing in keywords then pushing 'Enter' key repeatedly.

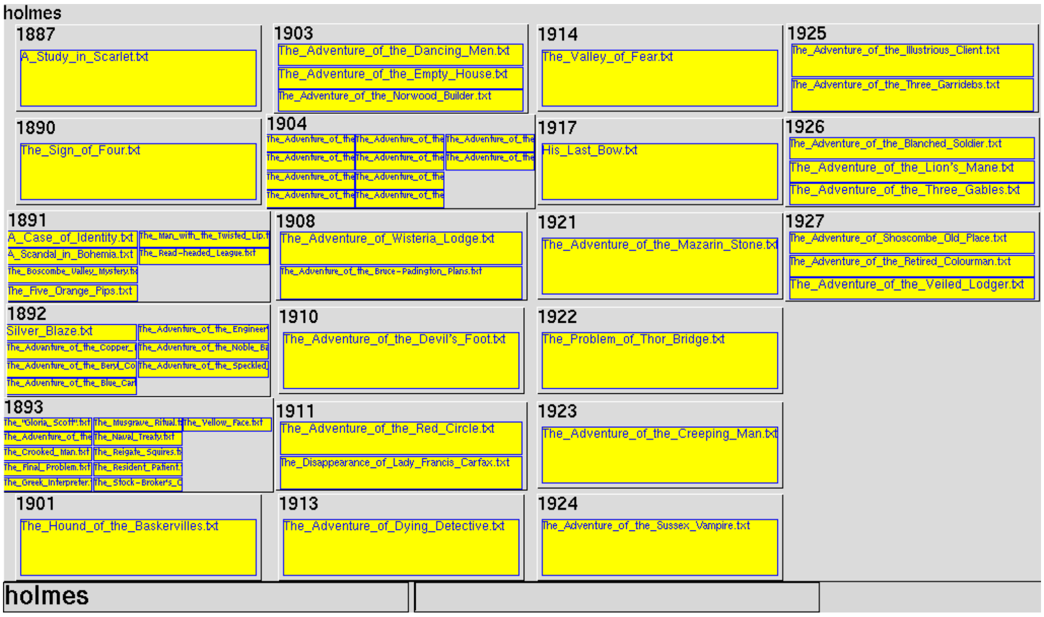

holmes
1887
A_Study_in_Scarlet.txt
1890
The_Sign_of_Four.txt
1891
A_Case_of_Identity.txt
A_Scandal_in_Bohemia.txt
The_Boscombe_Valley_Mystery.txt
The_Five_Orange_Pips.txt
The_Man_with_the_Twisted_Lip.txt
The_Read-headed_League.txt
1892
Silver_Blaze.txt
1893
1901
The_Hound_of_the_Baskervilles.txt
1903
The_Adventure_of_the_Dancing_Men.txt
The_Adventure_of_the_Empty_House.txt
The_Adventure_of_the_Norwood_Builder.txt
1904
1908
The_Adventure_of_Wisteria_Lodge.txt
The_Adventure_of_the_Bruce-Padington_Plans.txt
1910
The_Adventure_of_the_Devil's_Foot.txt
1911
The_Adventure_of_the_Red_Circle.txt
The_Disappearance_of_Lady_Francis_Carfax.txt
1913
The_Adventure_of_Dying_Detective.txt
1914
The_Valley_of_Fear.txt
1917
His_Last_Bow.txt
1921
The_Adventure_of_the_Mazarin_Stone.txt
1922
The_Problem_of_Thor_Bridge.txt
1923
The_Adventure_of_the_Creeping_Man.txt
1924
The_Adventure_of_the_Sussex_Vampire.txt
1925
The_Adventure_of_the_Illustrious_Client.txt
The_Adventure_of_the_Three_Garridebs.txt
1926
The_Adventure_of_the_Blanched_Soldier.txt
The_Adventure_of_the_Lion's_Mane.txt
The_Adventure_of_the_Three_Gables.txt
1927
The_Adventure_of_Shoscombe_Old_Place.txt
The_Adventure_of_the_Retired_Colourman.txt
The_Adventure_of_the_Veiled_Lodger.txt
holmes

Figure 5.21: Sherlock Holmes stories classified by written year

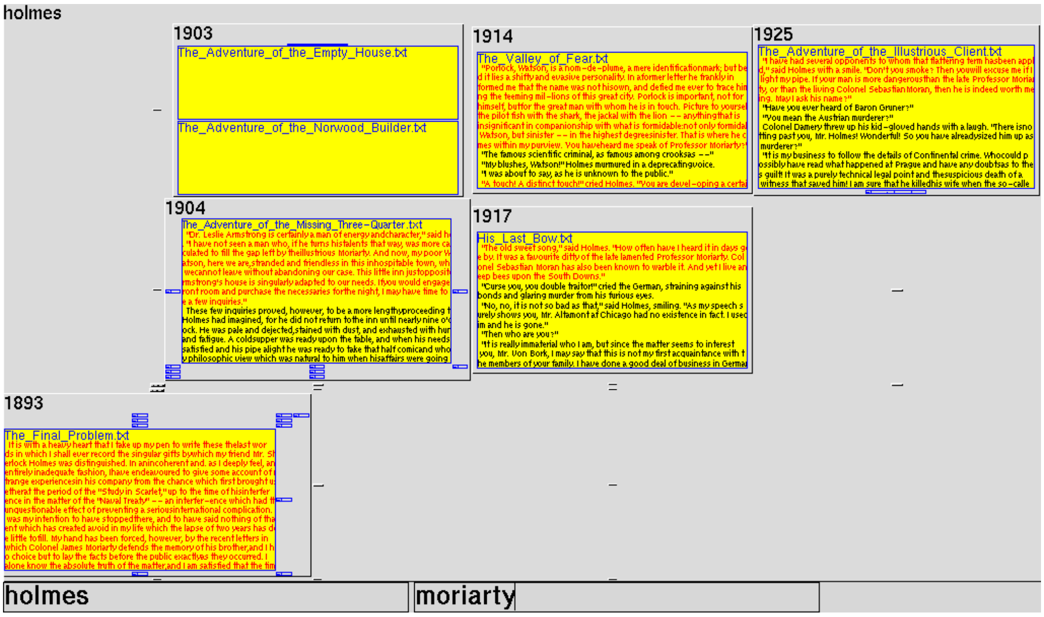

holmes
1903
The_Adventure_of_the_Empty_House.txt
The_Adventure_of_the_Norwood_Builder.txt
1914
The_Valley_of_Fear.txt
1925
The_Adventure_of_the_Illustrious_Client.txt
1904
The_Adventure_of_the_Missing_Three-Quarter.txt
1917
His_Last_Bow.txt
1893
The_Final_Problem.txt
holmes
moriarty

Figure 5.22: Search with a keyword "moriarty"

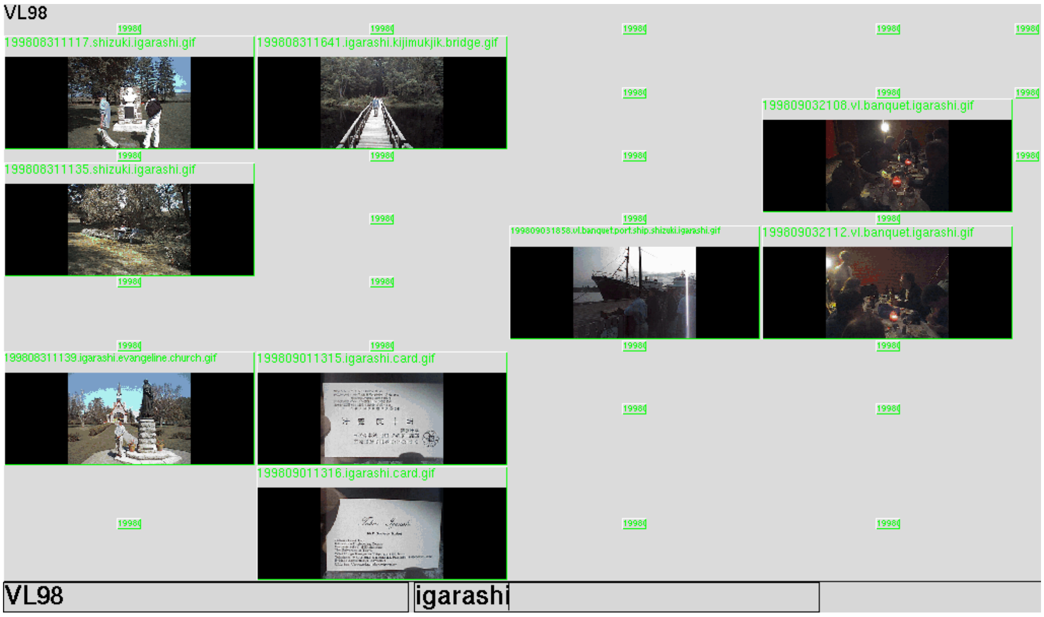

VL98
199808311117.shizuki.igarashi.gif
199808311641.igarashi.kijimukjik.bridge.gif
199809032108.vl.banquet.igarashi.gif
199808311135.shizuki.igarashi.gif
199809031858.vl.banquet.port.ship.shizuki.igarashi.gif
199809032112.vl.banquet.igarashi.gif
199808311139.igarashi.evangeline.church.gif
199809011315.igarashi.card.gif
199809011316.igarashi.card.gif
VL98
igarashi

Figure 5.23: A photo browser

- Selecting an appropriate node then typing in keywords.
- Described in Section 5.4.2.

In addition, there is an another way to explore a hierarchy. The user can explore by keywords, assuming that the path name are attached to each node. For example, to reach the directory "/graphics/3D/Software," the user types in keywords "/gra /3d /sof." It may be faster than direct manipulation, because the user can omit keywords in the middle of the path.

5.5 Implementation

We implemented Hyper Mochi Sheet as a library for the Amulet user interface development environment[39] (Version 2.0), and made it easy for Amulet users to use the library. Amulet is based on a prototype-instance object system constructed on C++[59], and provides a constraint solver. The detailed description of Amulet can be found in the Amulet home page[39].

The main component of Hyper Mochi Sheet is the MochiGroup object, which aggregates graphical objects hierarchically. Since MochiGroup inherits a group object Am_Group in Amulet, application programmers can use MochiGroup in the same way as Am_Group, except that child objects are arranged and zoomed automatically. In addition, the programmer can define a function $f(F)$, for the predictive focus selection, as a method in MochiGroup.

The following code fragment creates a new instance of MochiGroup, and adds some graphical object to the instance.

```
Am_Object new_grp = MochiGroup.Create()
  .Set(Am_WIDTH, 200)
  .Set(Am_HEIGHT, 200)
  .Add_Part(Am_Rectangle.Create())
  .Add_Part(MochiGroup.Create());
```

In Amulet, a new object is created by copying an existing object, and modifying attributes. The first line creates a instance of MochiGroup by invoking the Create method, and assigns it to the variable new_grp. Next two lines set the width and height of new_grp. Last two lines add instances of Am_Rectangle and MochiGroup to new_grp by the Add_Part method.

The Mochi Sheet algorithm is implemented as constraints between MochiGroup and its children. Therefore, positions and sizes of graphical objects are automatically recalculated, whenever the layout is modified by user interactions.

The structure of a Hyper Mochi Sheet application is almost the same as one of an ordinary Amulet application. The following is the skeleton of the main function.

```
int main(void)
{
  Am_Initialize();      // Initializing Amulet
  Initialize_Mochi();   // Initializing Hyper Mochi Sheet

  Am_Object TopGroup = MochiGroup.Create();

  Am_Object TopWindow = Am_Window.Create("TopWindow")
    .Set(Am_WIDTH,      800)
    .Set(Am_HEIGHT,     600)
    .Add_Part(propagateChildren, TopGroup)
    ;
13   Am_Screen.Add_Part(TopWindow);
14
15   // application dependent initialization code
16
17   Am_Object SelectionWidget = MochiSelectionWidget.Create()
18     .Set(Am_OPERATES_ON, TopWindow)
19     ;
20   TopWindow.Add_Part(Am_SELECTION_WIDGET, SelectionWidget);
21
22   Am_Main_Event_Loop();  // invoking the main event loop
23   Am_Cleanup();
23 }
```

To use Hyper Mochi Sheet library, the Initialize_Mochi() function must be invoked after Am_Initialize(). Lines 6 to 12 create TopGroup and TopWindow. TopGroup is the root of a nested network, and is added to TopWindow at the line 11. TopWindow is the main window of the application, which inherits a window object Am_Window. The window is displayed in the screen by adding it to the Am_Screen object (the line 13).

Hyper Mochi Sheet features are activated by adding MochiSelectionWidget to the window object as shown in lines 17 to 20. Once MochiSelectionWidget is added to the window, the user can select, move, and resize graphical objects in the nested network with the focus size prediction and the predictive focus selection. MochiSelectionWidget inherits Am_Selection_Widget, which provides editing support with handles such as selecting, moving and resizing.

To complete the application, the programmer must define graphical objects to be edited, and menus for editing operations. To define graphical objects, Hyper Mochi Sheet provides text and image objects with zooming support. Menus and tools can be implemented in the same manner as Amulet. In addition, Hyper Mochi Sheet provides widgets for the dynamic query function, such as a keyword input box, and a text browser with highlighting support.

5.6 Conclusion

We have proposed a new GUI framework to support navigation and editing through multi-focus distortion views. This framework keeps distortion layout free and flexible, while reducing the number of command invocations by automatic focus management techniques: the focus size prediction, the predictive focus selection, and the dynamic query. In the following, we show how proposed techniques satisfy our principles in Section 4.4.

5.6.1 Keeping Distortion Layouts Free and Flexible

We kept distortion layouts free and flexible, while reducing the number of command invocation. Hyper Mochi Sheet allows the user to select multiple nodes as focal points in a nested network, and to magnify and demagnify them in arbitrary sizes. Although the user cannot modify the shape of a focal point, our algorithm is sufficient for implementing our target applications, such as visual programming environments, directory editors, and presentation tools.

5.6.2 Reducing Boring, Repetitive Work

Hyper Mochi Sheet reduces boring, repetitive work using the focus size prediction technique. The user does not have to explicitly designate a set of sizes for each node, as the system predicts the sizes from a history of editing operations. This technique is based on the heuristics obtained from preliminary user test, and we showed reasonable accuracy of this technique with an experiment. The technique cut the number of size designations by about eighty percent.

5.6.3 Reducing Multi-focus Management

Hyper Mochi Sheet supports automatic multi-focus management by the predictive focus selection technique, which automatically defocuses unnecessary foci during navigation with hyperlinks. In most case, the user does not have to adjust distortion layouts before or after following hyperlinks. Application programmers can reduce the error rate of this technique by customizing prediction methods for their applications.

5.6.4 Reducing the Time for Switching from Searching to Editing

Hyper Mochi Sheet also reduce the time for switching from searching to editing by its dynamic query technique. This shows search results clearly by magnifying nodes matched to a query, so that the user can quickly edit the results of the query.

Chapter 6

Enhancing Usability with Application Semantics

This chapter shows that the use of application semantics provides more enhanced support for navigation and editing, and decreases the number of command invocations. We have made a case study using a visual programming environment (VPE) as a sample application. In VPEs, distortion view interfaces have not been used effectively though there are some VPEs with distortion views such as VIPR[12]. This study is one of the earliest attempt to use application semantics for editing visual programs through distortion views.

In the VPE, we use type information and design information as application semantics. Using the semantics, we can provide various support, such as searching appropriate components, focusing components to be modified, and automatic component linking.

Each support is realized based on the notion of *visual design patterns* (VDP), which is a visual abstraction representing design aspects in dataflow programs. VDP serves as a flexible and high-level structure of reuse for visual parallel programming.

In the following, we first describe the background of this work, and introduce the notion of VDP and its implementation in KLIEG visual programming environment (KLIEG-VDP). Then we explain how application semantics support navigation and editing in KLIEG-VDP system.

6.1 Background

Design pattern approaches have been recently proposed to describe recurring design aspects in object-oriented systems for enhancing software reusability. A design pattern is a document that describes the combination technique of abstract objects using diagrams, descriptions, and example programs. Catalogs of design patterns such as [18] have been published; with these catalogs, non-expert programmers can use well-designed combination techniques, and can construct reusable software by implementing abstract objects depicted by the patterns.

Design pattern approaches are also important in visual parallel data-flow programming (VPDP). Our goal is to formulate the notion of design patterns that is suitable for VPDP, and to make it easier to define and reuse design patterns, and also to reuse implemented programs.

It is, however, difficult to practice pattern-based design in VPDP, because normal VPDP languages such as CODE [45] do not support the notion of replaceable components. In an object-oriented design pattern, each abstract object represents a replaceable component, and a particular behavior of a program can be determined or modified by replacing components in the pattern. Such essential mechanisms to reuse designs and programs should also be supported by VPDP languages, but most lack systematic means to replace their components.

In addition, design information is important not only for documents, but also for programming environments. For object-oriented languages, design patterns, such as [10]. However, because they do not maintain design information in the generated program itself, it is difficult to learn the intention of the program design, such as which components can be modified to change a particular behavior.

6.1.1 Visual Design Patterns

Under these observations, we introduce reusable program structures based on data-flow diagrams, in which the designer can define replaceable components, and add design information directly to the structures. We call these structures *visual design patterns* (VDP). We directly support interactive definition, reuse, and even execution of VDPs in a visual parallel data-flow programming environment.

Here are some important design issues for a VDP system:

Management of multiple sets of processes A well-designed VDP will facilitate processes which can be implemented in several ways. However, it is time-consuming task to find appropriate processes from libraries and drag-and-drop them. In addition, changes to the processes must often be coordinated, i.e., a set of processes must be used at once. Therefore it is necessary for the VDP system to allow find, management, and manipulation of a set of processes, so that the designer can allow the user to select from an abstract set of process implementations such as *default*, *sample*, and *alternatives*.

Focusing support for processes editing Since most VDPs are comprised of a numerous number of nested processes, it is important to assist the user on which parts of the given VDP he should edit on adding/modifying some new functionality. Thus, a VDP system should facilitate a feature to focus the users editing actions on the particular part of VDP subject to editing.

Visualizing execution of VDPs In order for the user to capture the behavior of a VDP, it is important for the execution of each instantiated VDP to be visualized and animated. This will manifest to the user design knowledge that dynamic and otherwise difficult to document statically.

Support for consistency checking Consistency checks on process instantiation and replacement in VDP are desirable to improve usability. For example, a user may instantiate a pattern with a wrong set of processes; by incorporating consistency checking mechanism into a VDP system, such errors can be checked, or the operation is invalidated in the first place. Furthermore, such consistency information can be employed to assist in the editing, when there are ambiguities on which inputs connect to which outputs, etc.

We have implemented a VDP system based on a visual parallel programming environment KLIEG, which answers the above issues. KLIEG itself is a visual parallel data-flow language based on a parallel logic programming language, moded FGHC [69]. We refer to a VDP in KLIEG as a KLIEG-VDP. Designers can define KLIEG-VDPs that retain design information as patterns, and then users choose a pattern from a catalog of KLIEG-VDPs in their programs and implement customized processes in the pattern.

6.1.2 Existing Work

There are many visual parallel data-flow programming environments such as CODE[45], Meander[70], and SAA[40]. There are also environments based on parallel logic programming languages like Pictorial Janus[25], and PP[62]. Because these environments lack the support for VDPs, it is difficult to replace a set of components in data-flow diagrams. The user must manually delete a set of processes, create new processes, and link processes appropriately. This is a tedious and time-consuming task.

In VISTA[51], *processors* may have an internal network of processors. In particular, internal processor called *public processor* can be replaced by another processor with a compatible interface. Although similar to VDPs, public processors in VISTA are merely replaceable, thus the user must search appropriate processors from libraries. In addition, since VISTA shows only the list of public processors, it is difficult to know which processors need to be changed when multiple processors should be changed to obtain desired behavior.

Holon/VP[28] uses an object sharing technique to enhance reusability of program. This technique allows multiple networks sharing the same process as their components. This is usable for customizing existing programs by adding functions, but difficult for customizing by *replacing* the functions, which is easy with KLIEG.

6.2 Visual Design Patterns on KLIEG

In this section, we describe the details of KLIEG-VDPs. A KLIEG-VDP is a data-flow network diagram that has some replaceable and non-instantiated processes called *holes* as parameters, and that maintains the design information described in the previous section. Data-flow network diagrams are components of KLIEG programs, and consist of *processes* with input or output *ports* and *links* that connect input ports and output ports. A network diagram may be constructed hierarchically from multiple networks.

Users can reuse the topology of the KLIEG-VDP by instantiating the holes with customized processes which are appropriate for the KLIEG-VDP. A hole also has ports, and can be instantiated with processes that have at least the ports of the same type.

In the following, we first describe the features of KLIEG-VDP, then show an example of KLIEG-VDP.

6.2.1 Features of KLIEG-VDP

The features of the KLIEG-VDP are as follows:

1. The user can find appropriate processes for a hole by a process search function. This function searches processes using the types of ports in the hole. In addition, a hole is allowed to hold multiple processes, one of which is valid at a given time. This allows the designer to provide a default implementation, several alternatives, and sample code for a hole in the KLIEG-VDP. The user can select an appropriate implementation of the hole from these alternatives.

2. For highlighting replaceable processes, KLIEG-VDPs support multi-focus distortion viewing. The designer magnifies the processes which should be replaced together for changing a particular behavior, and saves the layout with an appropriate name. The user can easily determine the replaceable processes by selecting the layout.

3. To show the dynamic behaviors of processes, KLIEG provides an execution *tracer*, which visualizes and animates the execution of the program. The user can see the behaviors by executing the sample code.

4. KLIEG-VDP system facilitates type checking and inference algorithm on communication ports for consistency checking. By checking types, KLIEG presents only the appropriate processes for a hole from multiple implementation choices. KLIEG also links replaced processes as automatically as possible.

Since features 1 and 2 are closely related to navigation for editing, they are supported by Hyper Mochi Sheet with application semantics of KLIEG. Though other features also use application semantics, they are out of the scope of Hyper Mochi Sheet. We use type information and design information as application semantics. Type information provides keys for searching processes, and design information shows which processes should be focused. These features can be implemented with the dynamic query technique in Hyper Mochi Sheet.

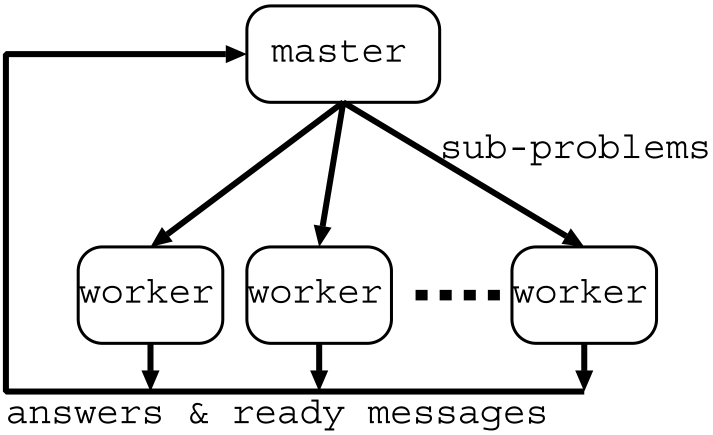


Figure 6.1: Master worker pattern

6.2.2 An Example

We show the *master-worker* pattern, which provides a simple load balancing scheme that involves a master process and a collection of worker processes. Figure 6.1 illustrates the concept of the master-worker pattern. The master partitions a problem into sub-problems and sends them to workers ready to compute. The workers are responsible for computing sub-problems. A worker returns the solution(s) of a sub-problem to the master. When a worker completes the computation of a sub-problem, it notifies the master that it is ready to compute again. By providing appropriate masters and workers, we can use this pattern for solving various parallel programming problems, such as ray-tracing and search problems.

Figure 6.2 depicts the KLIEG-VDP that represents the master-worker pattern. The master_worker pattern is a network constructed from two networks, master and workers, that correspond to the master process and the collection of worker processes, respectively. Both master and workers have some holes that represent processes, which depend on the problem to solve[1].

These networks have ports (represented by white rectangles) to communicate with each other. An arrow linking two ports represents a stream that is a continuous data-flow between the ports. For example, master has two ports Wks and Ans to communicate with workers.

[1] Master is shrunken, so the details are hidden.

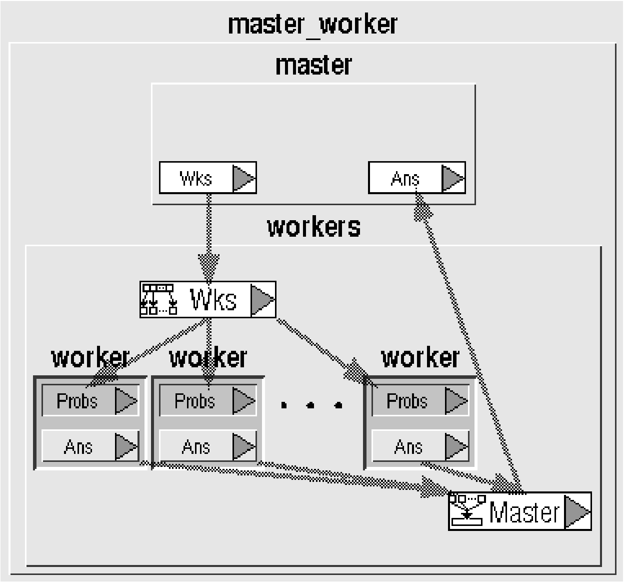


Figure 6.2: Master-worker pattern in KLIEG

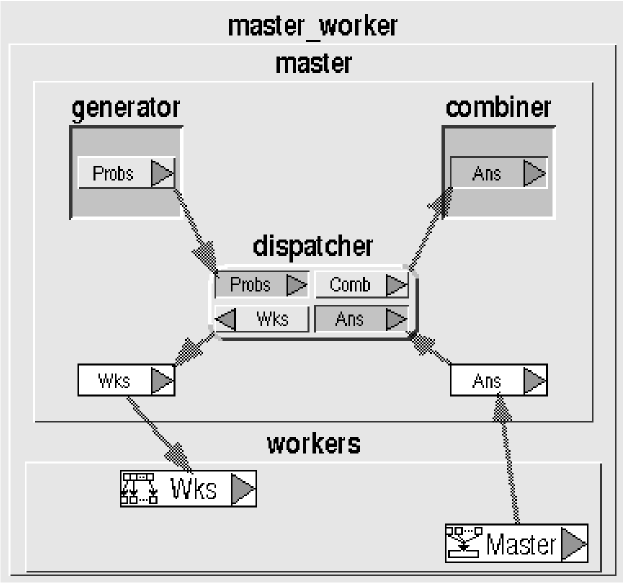


Figure 6.3: Detail of master network

The workers is defined using a *replication network* that replicates processes dynamically, and connects those processes. The replicated processes in workers are represented by three holes (recessed rectangles labeled worker), and an ellipsis, which abbreviates a set of processes. Each worker hole has an input port (the recessed rectangle labeled Probs) and an output port (the raised rectangle labeled Ans). Each replicated worker process receives sub-problems from the Probs port, solves them, and returns answers to the master via the Ans port.

A replication network has some special ports that determine the number of replicated processes and the topology of the network. For example, the Wks port in workers is a *map port* that determines the number of processes to generate by the number of received elements from the port, and maps each element to each process. Master (at the bottom-right of Figure 6.2) is a merge port that merges the output streams of all the processes. Besides these ports, we can use *broadcast ports* that broadcast received elements to all the processes.

In Figure 6.3, the structure of the master hidden in Figure 6.2 is expanded by zooming. The master network is composed of the generator and combiner hole, and the dispatcher process. Generator simply generates a stream of sub-problems. Combiner receives answers from dispatcher and computes the final answer. The dispatcher process is the default implementation of the dispatcher hole. It receives sub-problems from generator and messages from workers (ready and answer). Then it sends the sub-problems to ready workers, and sends the answers to the combiner.

In addition, we should also mention that KLIEG-VDPs can be combined hierarchically for constructing large-scale programs from smaller ones. To construct KLIEG-VDPs hierarchically, we merely instantiate a hole with an entire KLIEG-VDP. In fact, master_worker is constructed in this way, i.e., master_worker has two holes, and these holes are instantiated with the master and workers networks.

In the following sections, we will show the details of features in Section 6.2.1. Definition of network diagrams can be easily performed by normal graph editors, so we will concentrate on issues on definition and use. We first describe support with Hyper Mochi Sheet in Section 6.3, then explain other support in Section 6.4. We use the master_worker pattern as an example.

6.3 Support with Hyper Mochi Sheet

6.3.1 Management of Multiple Implementations

The designer can search appropriate processes for a hole. KLIEG searches processes using the types of ports in the hole, and show the result in a dialog box. The hole is replaced only by clicking one of processes. We found this is easier than dragging an implementation from the other module every time the user replaces an implementation. Another method for showing search results is magnifying processes using distortion views. The designer and programmer can easily access the definitions of processes for determining which process is appropriate for the hole. This feature can be easily implemented by customizing the search engine of the dynamic query function in Hyper Mochi Sheet[2].

In addition, the designer can store a hole with multiple implementations by repeated drag-and-dropping icons of appropriate processes onto the hole. Using this interface, the designer can provide a KLIEG-VDP with different kinds of implementations:

Sample implementations By executing these implementations on the tracer, the user can understand the behavior of the KLIEG-VDP.

Default implementations The most likely implementations the user is likely to use.

Alternative implementations The implementations that serve as the basis of user customization.

Implementations in a hole are chosen with a dialog box. Using the dialog box, the user can not only select an implementation, but also change a particular specification of an existing program[3].

As an example, we show how to manage multiple implementations for the master-worker pattern to solve search problems. When solving search problems with the master-worker pattern, combiner might be implemented independent of the problem to solve. Thus we provide two implementations

[2] Currently we have not yet implemented the distortion view version of the search function. The future version will incorporate this function.

[3] A well-designed VDP should be able to change its specification by replacing implementations in holes.

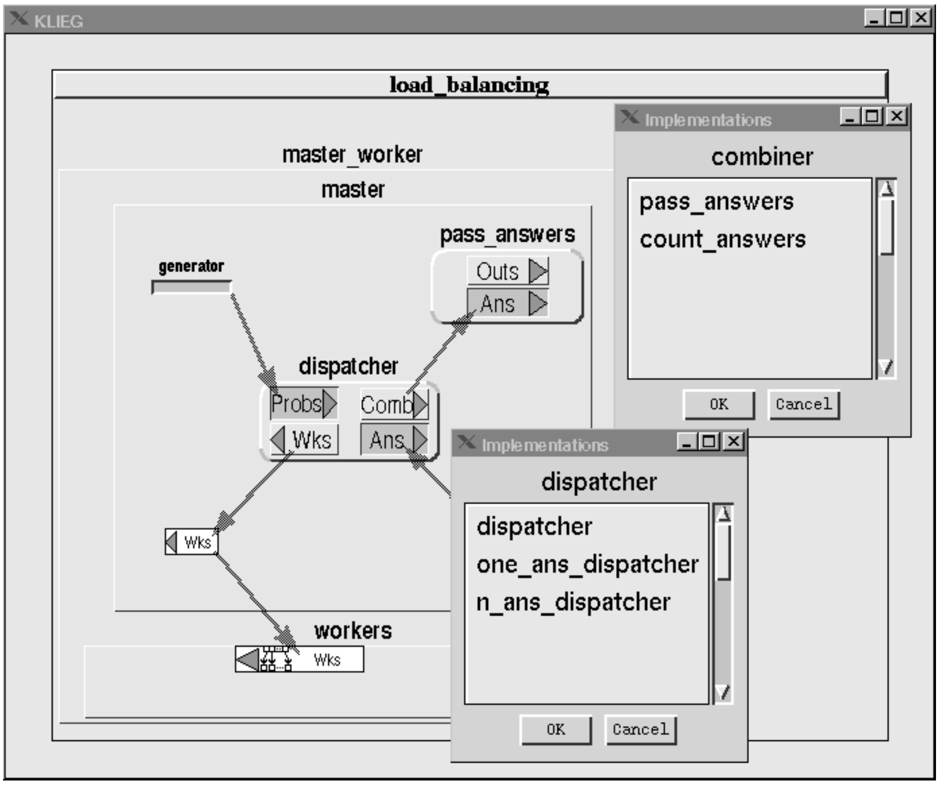


Figure 6.4: Alternative implementations

to combiner (Figure 6.4). On the top-right of Figure 6.4, the dialog box shows dropped processes. To show the result of query, we use the same dialog box. The pass_answers process passes through the answers from dispatcher to the output, and the count_answers counts the number of answers.

In addition, we can change the treatment of answers from finding all answers to finding the fixed number of answers. This is done by changing the dispatcher process to the process that terminates all the workers when it receives the fixed number of answers from workers. To perform this, we only add an implementation of a new dispatcher to dispatcher hole. On the bottom-right of Figure 6.4, a dialog box shows the one_ans_dispatcher process for finding single answer, and the n_ans_dispatcher process for finding n answers.

6.3.2 Focusing Support for Processes Editing

The designer can show which processes should be modified to change a particular behavior of a KLIEG-VDP by providing a distortion view layout of the KLIEG-VDP in which the processes are magnified. The user can select a layout by the behavior name and can easily find out which processes should be modified.

Using multi-focus distortion viewing, we can obtain a layout in which all related components are magnified in a single view, even if they are separated from each other on screen. In such a view, we can see the details of the components and the overview of the network at the same time. In addition, the editor of KLIEG animates transition from one layout to another layout, so that the user is not confused even if the layout drastically changes.

When the user intends to change a particular behavior of a program, the user selects an appropriate layout and replaces the magnified processes. The changes to a KLIEG-VDP in one layout view are reflected in other layouts.

This support uses design information as application semantics, and can be easily implemented with Hyper Mochi Sheet library. One way to implement the function uses the dynamic query technique. The programmer can find modifiable processes by dynamic queries, if the name of a behavior is attached to the processes by the designer. Another implementation uses saving and recovery of distortion view layouts. The designer should make distortion views and save them with behavior names. It is obvious to introduce layout saving and recovery functions into Hyper Mochi Sheet.

As an example, we show two layouts of the master-worker pattern for search problems. One is for changing the problem to solve, and the other is for changing the treatment of the answers. Figure 6.5 depicts two layouts and a dialog box for changing layouts. The layout on the left is for changing the problem. To change the problem, the user must change the generator and the worker magnified in the editor. Another layout on the right is for changing the treatment of the answers. Similarly, the treatment of the answers can be changed by modifying combiner or dispatcher magnified in the editor.

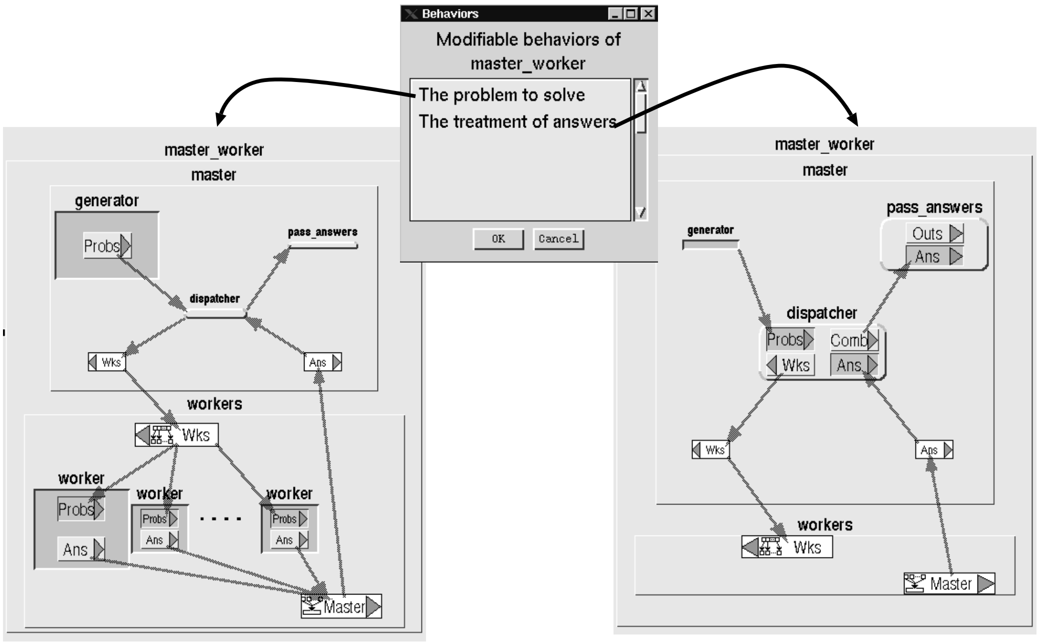


Figure 6.5: Changing layout for modifying behaviors

6.4 Other Support

6.4.1 Visualizing Execution of KLIEG-VDPs

The user can observe the behaviors of KLIEG-VDPs by executing a sample program with the tracer. As an example, we show the sample program that solves the N-Queens problem using the master_worker pattern, and its execution on the tracer. The program in Figure 6.6 was constructed by selecting the layout for changing the problem (Figure 6.5), and instantiating the holes with processes for the N-Queens problem. In the master network, the ports Size and Depth[4] are added for inputting the search parameter, and the Answer port is added to output answers.

Figure 6.7 shows a snapshot of the execution of the N-Queens program in the tracer. The tracer animates the transition of the network during the execution, while maintaining the topology of the pattern in the program. This is possible because the design information of the KLIEG-VDP can be

[4] They are singleton ports that receive only one datum

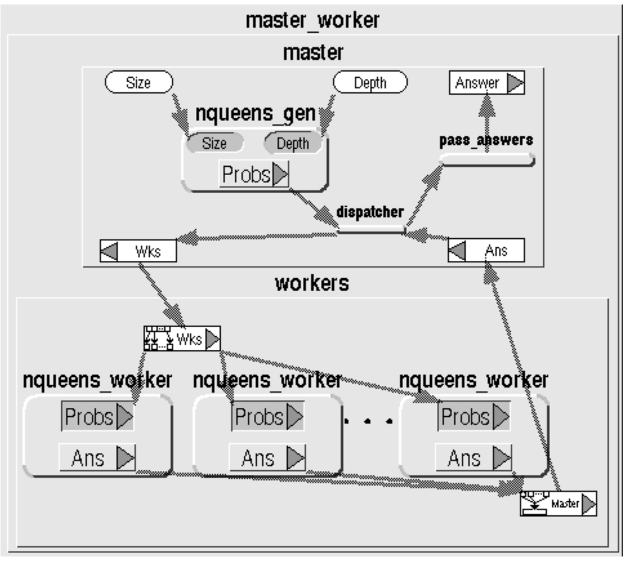


Figure 6.6: Solving N-Queens problem using master_worker

referred from the runtime system of KLIEG. The tracer can also show the contents of streams. Thus the user can easily recognize the KLIEG-VDPs used in the program, and can observe the behaviors from the animation and the contents of streams. The tracer also supports distortion viewing, so the user can navigate through large-scale networks using automated zooming of important parts of the program.

6.4.2 Checking Consistency Using Types

To further improve the user interface of KLIEG-VDPs, we employ type checking and inference of ports of processes. Using type information, KLIEG can restrict the possible candidates for a hole, and can connect ports of processes automatically.

As an example, Figure 6.8 shows the situation where the generator hole has been instantiated with nqueens_gen process that generates sub-problems for the N-Queens problem. In this case, processes that have no relevance to the N-Queens problem should be disabled when the user instantiates the worker hole or selects the worker process. To detect this, the port type of Probs of nqueens_gen is propagated to dispatcher and then to the worker holes by

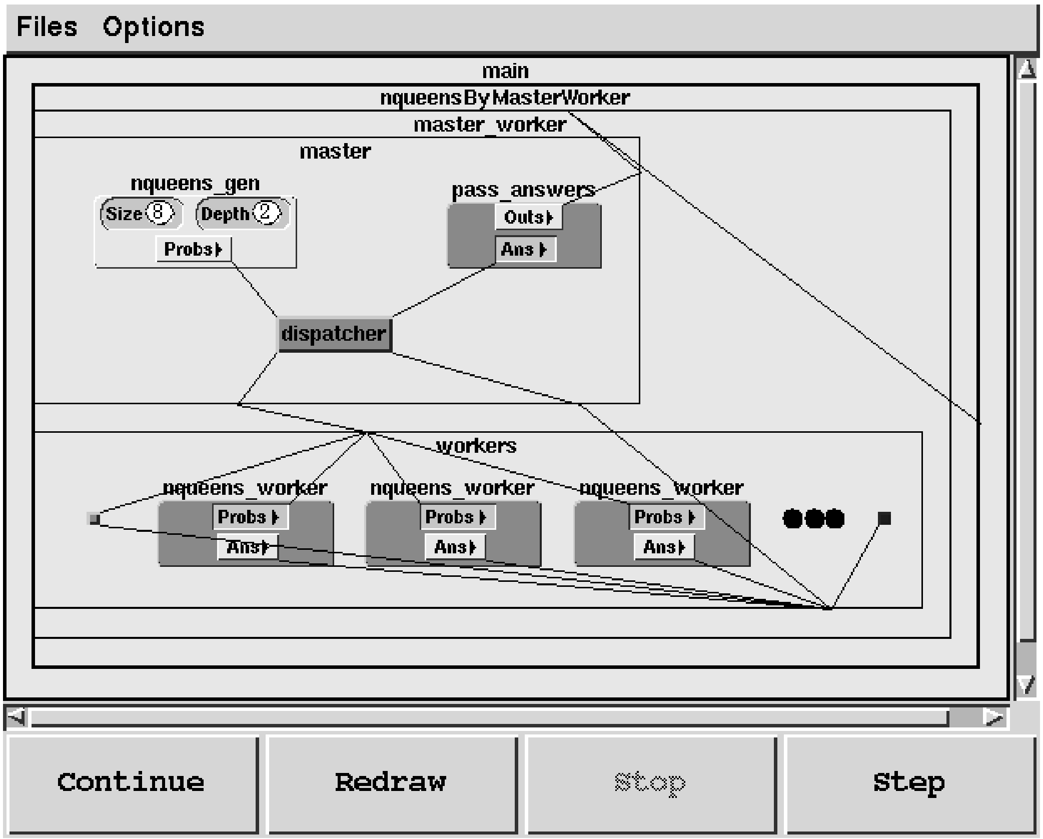


Figure 6.7: Executing the N-Queens program on the tracer

the type inference algorithm, so that processes except for nqueens_worker are disabled in the Implementations dialog box (at the bottom-right of Figure 6.8).

As another example, consider the case the user replaces dispatcher with another implementation. It would be impossible to connect ports automatically without type information, because dispatcher has two input ports and two output ports, and can be linked in different ways. In this case, KLIEG automatically connects the ports correctly if the types of all ports are detected. In this way, the user does not have to reconnect each port every time when replacing the processes.

To implement these type checkings, we use a constraint based type analysis. The base language of KLIEG is moded FGHC [69], and our analysis technique is also based on the mode-analysis algorithm of moded FGHC. We

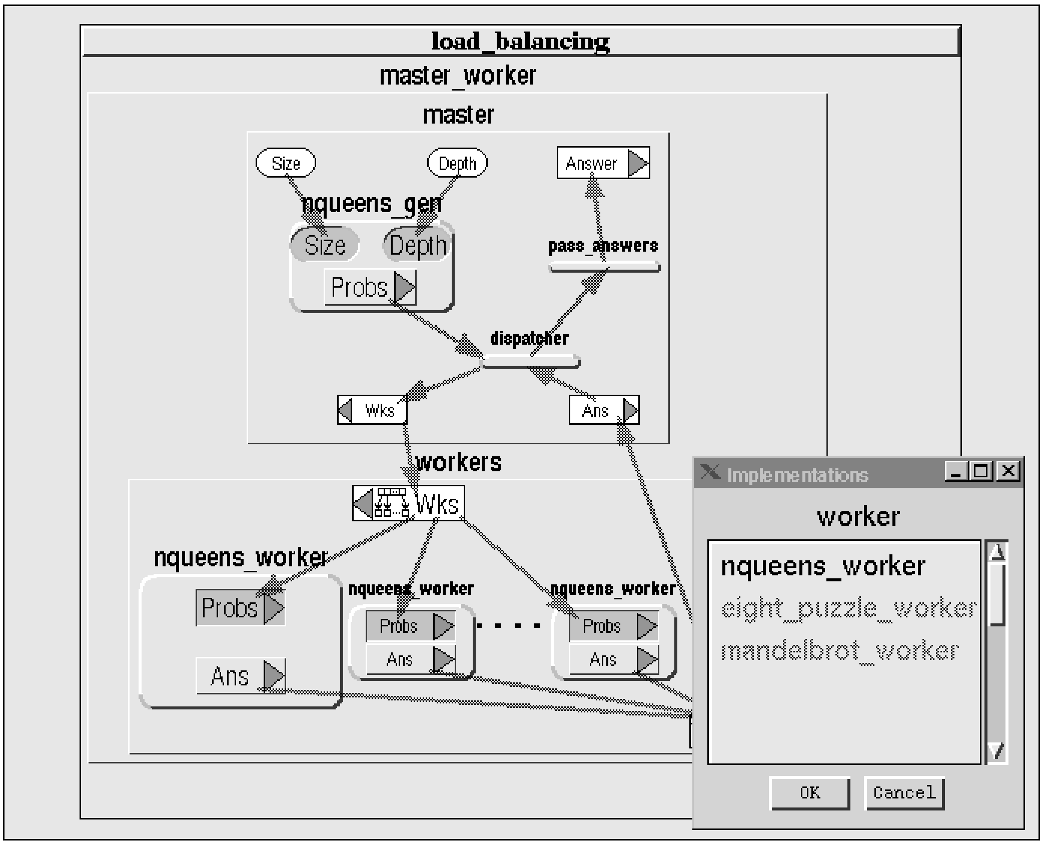


Figure 6.8: Checking port types of the worker

omit the details of the algorithm for brevity; interested readers are referred to [69].

6.5 Programming with KLIEG-VDPs

For users, programming using KLIEG-VDPs is simple and easy. It can be performed using the same interfaces for definition, namely dragging and dropping interface and dialog boxes:

1. Select a necessary KLEG-VDP from a library, and drag and drop it on to one's program.

2. If a sample code exists, execute the code with the tracer, and confirm the behaviors of the KLIEG-VDP.

3. Select a layout corresponding to the behavior to be modified from the dialog box.

4. Leave the default implementation for a hole as is, if it is sufficient. If not, select an appropriate implementation for each hole from its dialog box. If there are no appropriate implementations, implement necessary processes for unspecified holes, and instantiate the holes with the processes using drag and drop.

5. Add necessary ports and links that are not defined in the KLIEG-VDP.

6. When changing the other behaviors of the program, repeat the steps 3 to 5.

7. Execute the program and for change in behavior go back to step 3.

6.6 Conclusion

We have shown that the application semantics can provide further support for navigation and editing in a visual programming environment. We use type information and design information as the semantics for the support. The management of multiple implementations and the focusing support for process editing can be easily implemented in our framework of Hyper Mochi Sheet. We also show some other support without Hyper Mochi Sheet, and proposed the notion of visual design patterns for integrating all support.

Chapter 7

Summary

We have proposed a new GUI framework to support navigation and editing through multi-focus distortion views. This framework keeps distortion layout free and flexible, while reducing the number of command invocations by automatic focus management techniques.

Hyper Mochi Sheet enhanced navigation for editing using three automatic focusing techniques: focus size prediction, predictive focus selection, and dynamic query. The focus size prediction automatically determines appropriate sizes of nodes, and reduces boring repetitive work. The predictive focus selection automatically defocuses unnecessary foci during navigation with hyperlinks, and reduces layout adjustment tasks before and after following hyperlinks. The dynamic query function allows the user to find scattering objects with simple operations and quickly create layouts for editing search results.

We have also shown that the application semantics can provide further support for navigation and editing. We have made a case study using a data-flow visual programming environment (VPE) as a sample application. We use type information and design information as the semantics for the support, and showed that navigation support can be easily implemented in Hyper Mochi Sheet.

7.1 Discussion

We have shown that the combination of proposed techniques is effective in navigating and editing nested network through multi-focus distortion views.

There are relationships between those techniques. In short, the focus size prediction technique provides a set of appropriate node sizes for automatic focusing in the predictive focus selection and the dynamic query techniques. In the following, we discuss generality and applicability of each technique.

7.1.1 Focus size prediction

The focus size prediction technique itself is independent of the other techniques. Basically, a set of node sizes determined by the technique is used for opening/closing the node, which is the most simple automatic focusing. The other techniques only reuse the set of sizes.

The technique is also independent of nested networks and the Hyper Mochi Sheet algorithm, since the prediction algorithm concerns only the history of resize and open/close commands on each node. Therefore, this technique can be applied to other distortion views that focus on each node, such as the layout-independent fisheye view[42], and the continuous zoom[5]. The technique is also applicable to pan/zoom based interfaces, such as Pad++[7, 8] and SHriMP[58], because they scale each node independently.

In addition, the technique is most effective in applications such as presentation tools and hypertext editors, in which the representation of each node is important and the user should adjust node sizes according to their preferences. Otherwise, the technique is less effective, because the user often compromises on a set of node sizes determined automatically.

7.1.2 Predictive focus selection

The predictive focus selection technique uses results of the focus size prediction for focusing necessary nodes and defocusing unnecessary nodes to appropriate sizes. Therefore, the user can quickly obtain an appropriate distortion view after following a hyperlink.

Although the technique uses results of the focus size prediction for focusing selected nodes, the method of the selection itself can be used independently of the other techniques. The technique can be applied to multi-window and multi-buffer systems for automatically closing unnecessary windows or buffers. Therefore, the technique is also independent of distortion views and nested networks.

In addition, the technique is more effective in editor applications than in browser applications. In editors, the system can obtain keys for the prediction from editing status and operations, in addition to navigation operations. In practice, useful semantics is often obtained from editing status and operations as shown in Chapter 5.

7.1.3 Dynamic query

The dynamic query technique uses results of the focus size prediction to show results of dynamic queries. Therefore, the user can quickly edit results of queries through an appropriate distortion view.

The technique relies on hierarchical structures. Reliable information hierarchy is necessary for the user to narrow search space correctly. Especially, it is important that each parent node is labeled appropriately, so that the user can guess subtrees by labels and paths from the root node.

In addition, the technique is applicable to various applications by changing the search engine. Hyper Mochi Sheet performs dynamic approximate string matching as the default search method, and allows the programmer to provide search engines customized for the application as shown in Chapter 6.

7.2 Limitations

The focus size prediction technique can not predict a node size that is not in the size history of the node. The user must manually resize the node when he or she cannot find an acceptable size in the history. One solution is that the system pre-calculates appropriate sizes according to application semantics.

The predictive focus selection technique is still a preliminary design and implementation, though it works well in our applications presented in this thesis. To introduce more application semantics as prediction keys, it is necessary to implement more applications, such as an object-oriented design tool, and to analyze requirements for automatic support.

Improvement of the user interface is also required. The size correction interface is still indirect for the user. A correction interface for the predictive focus selection is also necessary. It should allow the user to select a proper layout from multiple candidates.

Hyper Mochi Sheet also has an implementation related limitation, though it provides enough scalability for practical use. It is capable of visualizing about 6,000 nodes with smooth animation (2 – 5 frame per seconds) using PC/AT (CPU: Pentium 133MHz MMX, Video Chip: Chips&Technology

65554, and Memory: 64MB). The frame rate of animation depends mainly on how many nodes are visible and are resized at once. With more efficient graphics hardware and larger memory, Hyper Mochi Sheet can apparently display and interact with nested network containing up to 20,000 nodes. To obtain more scalability, we should introduce indexing and filtering of nodes, so that the system can load nodes to memory on demand.

7.3 Future Work

Extending the concept of Hyper Mochi Sheet to desktop environments is one of the most interesting directions for future work. The hierarchical visualization of a file system, and windows of executed applications would be seamlessly integrated in the desktop, using distortion views. Pull-down menus in applications can be also integrated into Hyper Mochi Sheet as zooming menus.

Applying Hyper Mochi Sheet to large (30 inches or more) and high resolution (with 3,000 or more pixels) screens is also an interesting direction. Since the user cannot see overall screen space at once, new distortion view algorithms will be required, so that the user can keep arbitrary sets of nodes at hand, while displaying overall context on the screen.

7.4 Conclusion

Although multi-focus distortion views provide flexible layouts for editing multiple parts, it is complicated and time-consuming task to navigate and edit structured information. The user's most important task is management of multiple foci during navigation for editing, in which the user creates a distortion view suitable for the next editing situation. We analyzed the general problems of navigation for editing, and show that the most important issue is trade-off between flexibility of distortion layouts and the number of command invocations.

We have proposed a new GUI framework for distortion views, which automatically manages multiple foci, and reduces the number of command invocation, keeping distortion layouts free and flexible. We implemented the framework as a general GUI library, Hyper Mochi Sheet. We have also shown that the application semantics can provide further support for navigation and editing.

Bibliography

[1] Yahoo!. http://www.yahoo.com.

[2] K. Arnold and J. Gosling. *The Java Programming Language.* Addison Wesley, 1996.

[3] Lyn Bartram and Tom Calvert. Evaluating the Role of Intelligent Support in User Interfaces to Supervisory Control Systems. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, pages 717–722, October 1994.

[4] Lyn Bartram, Frank Henigman, and John Dill. The Intelligent Zoom as Metaphor and Navigation Tool in a Multi-Screen Interface for Network Control Systems. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, pages 3122–3127, October 1995.

[5] 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*, pages 207–215, November 1995.

[6] Lyn Bartram, Russell Ovans, John Dill, Michael Dyck, Albert Ho, and William S. Havens. Contextual Assistance in User Interfaces to Complex, Time Critical Systems: The Intelligent Zoom. In *Graphics Interface '94*, pages 216–224, 1994.

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

[8] Benjamin B. Bederson, James D. Hollan, Ken Perlin, Jonathan Meyer, David Bacon, and George Furnas. Pad++: A Zoomable Graphical

Sketchpad For Exploring Alternate Interface Physics. *Journal of Visual Languages and Computing*, 7(1):3–31, March 1996.

[9] Benjamin B. Bederson, James D. Hollan, Jason Stewart, David Rogers, Allison Druin, and David Vick. A zooming web browser. In *SPIE Multimedia Computing and Networking*, volume 2667, pages 260–271, 1996.

[10] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu. Automatic code generation from design patterns. *IBM Systems Journal*, 35(2), 1996. – Object technology.

[11] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. 3-Dimensional Pliable Surfaces: For the Effective Presentation of Visual Information. In *Proceedings of UIST '95*, pages 217–226, November 1995.

[12] Wayne Citrin and Carlos Santiago. Incorporating Fisheying into a Visual Programming Environment. In *Proc. 1996 IEEE Symposium on Visual Languages*, pages 20–27, 1996.

[13] John Dill, Lyn Bartram, Albert Ho, and Frank Henigman. A Continuously Variable Zoom for Navigating Large Hierarchical Networks. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, pages 386–390, October 1994.

[14] Steven Feiner. Seeing the Forest for the Trees: Hierarchical Display of Hypertext Structure. In *Proceedings of the Conference on Office Information Systems*, pages 205–212. ACM, March 1988.

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

[16] George W. Furnas and Benjamin B. Bederson. Space-scale Diagrams: Understanding Multiscale Interfaces. In *Proceedings of CHI'95 Human Factors in Computing System*, pages 234–241, 1995.

[17] George W. Furnas and Xiaolong Zhang. Muse: A multiscale editor. In *Proceedings of UIST '98*, pages 107–116, November 1998.

[18] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[19] J. G. Hollands, T. T. Carey, M. L. Matthews, and C. A. McCann. Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views. In *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, pages 313–320, 1989.

[20] Brian Scott Johnson. *Treemaps: Visualizing Hierarchical and Categorical Data*. PhD thesis, University of Maryland, 1993.

[21] Sussanne Jul and George W. Furnas. Critical Zones in Desert Fog: Aids to Multiscale Navigation. In *Proceedings of UIST '98*, pages 97–106, November 1998.

[22] Naftali Kadmon and Eli Shlomi. A Polyfocal Projection for Statistical Surfaces. *The Cartographic Journal*, 15(1), June 1978.

[23] Karlis Kaugars, Juris Reinfelds, and Alvis Brazma. A Simple Algorithm for Drawing Large Graphs on Small Screens. In *Proceedings of Graph Drawing '94*, pages 278–281, 1994.

[24] T. Alan Keahey and Edward L. Robertson. Nonlinear Magnification Fields. In *Proceedings of the IEEE Symposium on Information Visualization, IEEE Visualization*, October 1997.

[25] Vijay A. Saraswat Kenneth M. Kahn. Complete Visualizations of Concurrent Programs and their Executions. In *Proc. 1990 IEEE Workshop on Visual Languages*, October 1990.

[26] Hideki Koike. Fractal Views: A Fractal-Based Method for Controlling Information Display. *ACM Transactions on Information Systems*, 13(3):305–323, 1995.

[27] Hideki Koike and Masayuki Inoue. A Distortion-Oriented Approach for Automatic Simplification of the 3D Scene based on the Scene Graph. IS Technical Reports UEC-IS-1997-10, Graduate School of Information Systems, University of Electro-Communications, 1997.

[28] Yuichi Koike, Yasuyuki Maeda, and Yoshiyuki Koseki. Enhancing Iconic Program Reusability with Object Sharing. In *Proc. 1996 IEEE Symposium on Visual Languages*, pages 288–295, 1996.

[29] John Lamping and Ramana Rao. Laying out and Visualizing Large Trees Using a Hyperbolic Space. In *Proceedings of UIST '94*, pages 13–14, November 1994.

[30] John Lamping and Ramana Rao. The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–55, March 1996.

[31] Y. K. Leung. Human-Computer Interface Techniques for Map Based Diagrams. In G. Salvendy and M. J. Smith, editors, *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, pages 361–368. Elsevier Science Publishers, 1989.

[32] Y. K. Leung and M. D. Apperley. Review and Taxonomy of Distortion-Oriented Presentation Techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, June 1994.

[33] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The Perspective Wall: Detail and Context Smoothly Integrated. In *Proceedings of ACM CHI'91*, pages 173–179, 1991.

[34] Toshiyuki Masui. An Efficient Text Input Method for Pen-based Computers. In *Proceedings of ACM CHI'98*, pages 328–335, April 1998.

[35] Kazuo Misue and Kozo Sugiyama. Multi-viewpoint perspective display methods: Formulation and application to compound graphs. In *Human Aspects in Computing: Design and Use of Interactive Systems and Information Management*, pages 834–838. Elsevier Science Publishers, 1991.

[36] Kazuo Misue and Kozo Sugiyama. How Does D-ABDUCTOR Support Human Thinking Processes? In *Proceedings of CG International '94*, 1994.

[37] Kazuo Misue and Kozo Sugiyama. Evaluation of a Thinking Support System from Operational Points of View. In *Symbiosis of Human and Artifact*. Elsevier Science Publishers, 1995.

[38] Deborah A. Mitta. A Fisheye Presentation Strategy: Aircraft Maintenance Data. In *Human-Computer Interaction – INTERACT '90*, pages 875–880, 1990.

[39] Brad A. Myers. Amulet Project Home Page. http://www. cs.cmu.edu/˜ amulet/.

[40] Keng Ng, Jeff Kramer, Jeff Magee, and Naranker Dulay. A Visual Approach to Distributed Programming. In A. Zaky and T. Lewis, editors, *Tools and Environments for Parallel and Distributed Systems*, chapter 1, pages 7–31. Kluwer Academic Publishers, February 1996. (ISBN: 0-7923-9675-8).

[41] Emanuel G. Noik. Exploring Large Hyperdocuments: Fisheye Views of Nested Networks. In *ACM Conference on Hypertext and Hypermedia*, pages 14–18, 1993.

[42] Emanuel G. Noik. Layout-independent Fisheye Views of Nested Graphs. In *Proc. 1993 IEEE Symposium on Visual Languages*, pages 336–341, 1993.

[43] Emanuel G. Noik. *Dynamic Fisheye Views: Combining Dynamic Queries and Mapping with Database Views.* PhD thesis, Department of Computer Science, University of Tronto, 1996.

[44] Ken Perlin and David Fox. Pad: An Alternative Approach to the Computer Interface. In *SIGGRAPH 93 Conference Proceedings*, pages 57–64, 1993.

[45] P.Newton and J.C.Browne. The CODE 2.0 Graphical Parallel Programming Language. In *Proc. ACM Int. Conf. on Supercomputing*, July 1992.

[46] Ramana Rao and Stuart K. Card. The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. In *Proceedings of ACM CHI'94*, pages 318–322, 1994.

[47] George G. Robertson and Jock D. Mackinlay. The Document Lens. In *Proceedings of UIST '93*, pages 101–108, November 1993.

[48] Manojit Sarkar and Marc H. Brown. Graphical Fisheye Views of Graphs. In *Proceedings of ACM CHI'92*, pages 83–91, 1992.

[49] 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*, pages 81–91, November 1993.

[50] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating Hierarchically Clustered Networks through Fisheye and Full-Zoom Methods. *ACM Transactions on Computer-Human Interaction*, 3(2):162–188, June 1996.

[51] Stefan Schiffer and Joachim Hans Fröhlich. Visual Programing and Software Engineering with Vista. In Adele Goldberg Margaret Burnett and Ted Lewis, editors, *Visual object-oriented programming: concepts and environments*, chapter 10, pages 199–227. Manning Publications Co., 1995.

[52] Buntarou Shizuki, Masashi Toyoda, Etsuya Shibayama, and Shin Takahashi. Visual Patterns + Multi-Focus Fisheye View: An Automatic Scalable Visualization Technique of Data-Flow Visual Program Execution. In *Proc. 1998 IEEE Symposium on Visual Languages*, pages 270–277, September 1998.

[53] Buntarou Shizuki, Masashi Toyoda, Shin Takahashi, and Etsuya Shibayama. A Visual Parallel Programming Environment KLIEG: Reuse of Program Components and Visualization of Program Execution by Process Network Pattern. In *Workshop on Interactive Systems and Software'96*, pages 81–90, December 1996. in Japanese.

[54] Ben Shneiderman. Dynamic Queries for Visual Information Seeking. *IEEE Software*, pages 70–77, November 1994.

[55] Robert Spence and Mark Apperley. Data-base navigation: An Office Environment for the Professional. *Behaviour and Information Technology*, 1(1):43–54, 1982.

[56] Michael Spenke, Christian Beilken, and Thomas Berlage. FOCUS: The Interactive Table for Product Comparison and Selection. In *Proceedings of UIST '96*, pages 41–50, November 1996.

[57] M. A. D. Storey and H. A. Müller. Graph Layout Adjustment Strategies. In *Proceedings of Graph Drawing 1995*, pages 487–499, September 1995.

[58] M. A. D. Storey, K. Wong, F. D. Fracchia, and H. A. Müller. On Integrating Visualization Techniques for Effective Software Exploration. In *Proceedings of the IEEE Symposium on Information Visualization, IEEE Visualization*, October 1997.

[59] B. Stroustrup. *The C++ Programming Language Second Edition.* Addison-Wesley, 1991. ISBN 0-201-53992-6.

[60] Kozo Sugiyama and Kazuo Misue. “Good” Graphic Interface for “Good” Idea Organizers. In *Human-Computer Interaction – INTERACT'90*, pages 521–526, 1990.

[61] Kozo Sugiyama and Kazuo Misue. Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, July/August 1991.

[62] Jiro Tanaka. Visual Programming System for Parallel Logic Languages. In *The NSF/ICOT Workshop on Parallel Logic Programming and its Program Environments*, pages 175–186. the University of Oregon, 1994.

[63] Masashi Toyoda, Toshiyuki Masui, and Etsuya Shibayama. HishiMochi: A Dynamic Search System with Nonlinear Zooming. In *Workshop on Interactive Systems and Software '98*, pages 143–152, December 1998. in Japanese.

[64] Masashi Toyoda and Etsuya Shibayama. Hyper Mochi Sheet: A Predictive Focusing Interface for Navigating and Editing Nested Networks through a Multi-focus Distortion-Oriented View. In *Proceedings of ACM CHI'99*, May 1999. Accepted and to be published.

[65] 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*, pages 76–83, September 1997.

[66] Masashi Toyoda, Buntarou Shizuki, Shin Takahashi, and Etsuya Shibayama. KLIEG: A Visual Parallel Programming Environment Using Process Network Patterns as Flexible Reuse Units. In *Technical Report*

of IEICE. COMP95-91, SS95-46 (1996-03), pages 25–30. The Institute of Electronics, Information and Communication Engineers, March 1996.

[67] Masashi Toyoda, Buntarou Shizuki, Shin Takahashi, and Etsuya Shibayama. Mochi Sheet: Integration of Zooming and Layout Editing. In *Proceedings of Interaction '97*, pages 79–86. Information Processing Society of Japan, February 1997. in Japanese.

[68] Masashi Toyoda, Shin Takahashi, and Etsuya Shibayama. Mochi Sheet: A Zooming Interface witch Supports Efficient Editing of Large Visual Programs. *Transactions of Information Processing Society of Japan*, 39(5):1395–1402, 5 1998. in Japanese.

[69] Kazunori Ueda and Morita Masao. Moded Flat GHC and Its Message-Oriented Implementation Technique. *New Generation Computing*, 13(1):3–43, 1994.

[70] Guido Wirtz. Modularization and Process Replication in a Visual Parallel Programming Language. In *Proc. 1994 IEEE Symposium on Visual Languages*, pages 72–79, 1994.