End User Controlled Interfaces: Creating Multiple View Interfaces for Data-Rich Applications*

Tyson R. Henry
Department of Computer Science
University of Arizona, Tucson AZ 85721

Scott E. Hudson
College of Computing
Georgia Institute of Technology, Atlanta, GA 30332-0280

University of Arizona Technical Report 92-04

Abstract

Direct manipulation interfaces give the user the feeling that he is interacting directly with the data. This feeling is achieved by designing the interface to closely resemble the user's mental model of the data. Thus the success of the interface depends on how closely it matches the user's mental model: how closely the interface objects match and how closely the structure of the interface objects matches. Since the user's mental model may depend on his current interests and the current data set, there is not always a single "best" structure for the interface. This is especially true in interfaces to applications with many data objects—*data-rich* applications. This paper presents techniques for creating flexible interfaces that allow the user to customize the structure of the interface. These techniques allow the user to restructure the interface so it matches his current interests and his current mental model of the data.

Keywords: direct manipulation, interface layout, user controlled interface structure.

1 Introduction

Direct manipulation interfaces provide the illusion of directly manipulating data objects [9, 15]. This illusion is most often created by representing the data objects with graphical interaction objects that can be directly manipulated by the user. One of the main goals of direct manipulation interfaces is to increase the directness felt by the user—so the user feels he is interacting directly with the data and not with an interface.

There are two important aspects of a user interface that contribute to its directness; how closely the interface objects and the interface structure match the user's mental model. However, all users do not always have the same mental model. Mental models may vary from user to user and may even depend on the user's current interests and the current data set. Thus there is not always a single "best" structure for any given interface.

The importance of the structure increases in interfaces that concentrate on the manipulation and exploration of large data sets—*data-rich* interfaces—because the data begins to predominate the interface. Not only does the structure become more important, the large number of interaction objects are more difficult to lay out on the screen.

Unlike many interfaces, a data-rich interface may be used for the general exploration of data or for other poorly structured tasks that cannot be well characterized in advance. In addition, multiple aspects of the data may need to be addressed simultaneously and multiple relationships between individual data items to be depicted. Finally, data-rich applications normally cannot control the number and properties of most of the user interface components in advance—since these are derived from a dynamic data set.

One approach to this difficulty is to present different views of the data for different purposes [2, 3, 4, 6]. As a simple example, the Macintosh Finder interface normally depicts files as icons on the desktop. However, the user can optionally select from a small set of alternative textual views. Each of these views has advantages and disadvantages, and each supports some tasks better than others. By providing alternative views, the supported range of tasks increases and the interface becomes more

*This work was supported in part by the National Science Foundation under grants CDA–8822652, and IRI–9015407.

useful.

Unfortunately, presenting a small fixed set of views has limitations—particularly for exploratory tasks in which the user's focus may change as the interaction progresses. To address these issues, this paper considers a more general approach to multiple view interfaces. This approach does not simply define a fixed set of views but provides the end-user with the capability to construct additional custom views. Clearly, not all end-users want or need to construct custom views. However, by giving sophisticated end-users the ability to create new views, the interface becomes more flexible and can be expanded to meet very specific needs or particular focuses.

The basic idea behind this approach is to use a flexible layout mechanism to position the interaction objects within the interface. Providing the user with an interface to the layout mechanism allows him to create new views of the interface and customize them to meet their mental model and current interests. There are three basic mechanisms with which the user can customize the interface: choosing a specialized algorithm, adjusting parameters to the layout algorithm, and building a new algorithm.

This approach stems from the authors' previous work in interactive graph layout systems [7, 8]. In these systems, a visual programming interface allows the user to interactively compose and manipulate custom graph layouts. This framework has been extended to include general layout composition of the kind found in a number of user interface toolkits [12] (although the examples used here still make heavy use of graph layout compositions since these are the most powerful and well developed).

The next section presents related work on flexible layout and multiple view interfaces. Section 3 will illustrate the use of multiple views in a spreadsheet application and Section 4 will demonstrate how views can be customized by the user to achieve a better interface for a very specific task. Finally, Section 5 will consider the the prototype implementation being used to explore these concepts, and Section 6 will provide a brief conclusion.

2 Related work

Most user interfaces provide a static presentation of the interface—they don't provide the end-user with any control over the structure of the interface. However, there are a few systems that allow the end-user to interactively control the structure. A few of these systems provided the basis for this research and are thus presented in this section.

An example of an interface in which the structure of the display can be modified is the Cone Tree [14]. This system is designed to visualize hierarchical data structures. It uses a 3D representation of the data that can be interactively animated. Interactive animation allows the user to not only modify the display but to create an animation sequence to display the data. Several example applications have been built in which the cone tree acts as both a visualization tool for the data set and as an interface to the data.

As mentioned in the introduction, the Macintosh Finder interface allows the user to choose between an iconic and a textual representation of files. In addition to providing these two formats, the interface allows the user to reorganize the files based on one of several key fields. When in iconic mode, a clean-up utility automatically arranges the icons to prevent overlapping.

While the Macintosh directory interface and the cone tree allow the user to modify the appearance of the interface to change focus, they do not provide general tools for interacting with the structure of the interface.

Most interface building toolkits provide the interface designer with a collection of tools for positioning a given static set of interaction objects on the screen [5, 10, 11, 13, 16]. These systems provide support for creating interfaces with fixed structures but they cannot create interfaces the end-user can restructure at runtime. They also cannot produce interfaces for dynamic data sets.

The Humanoid system [18] creates interfaces for dynamic data sets by using templates to position interaction objects. While the templates in Humanoid are fixed, a dynamic template selection algorithm examines the data set at run time and chooses an appropriate set of templates. While the Humanoid system provides a mechanism that handles dynamic data sets, it does not provide the end-user with the power to directly interact with the interface structure.

This work builds on the concepts in Humanoid by allowing the end-user to select and modify the equivalent of the Humanoid positioning templates. Unlike the Humanoid project, this work only deals with the task of positioning interaction objects within an interface and does not provide a general interface management system.

3 Multiple View Interfaces

As a sample interface, consider a spreadsheet. Each cell in a spreadsheet is an interaction object that allows the user to manipulate part of the data—usually a name, an equation and a value. Traditional spreadsheet interfaces are structured in a rectangular matrix—all the cells are positioned on the screen in a rectangular grid—as shown in the sample spreadsheet in Figure 1. The position of each cell is permanent and cannot be changed by the user.

Figure 2 shows an alternative interface to the sample

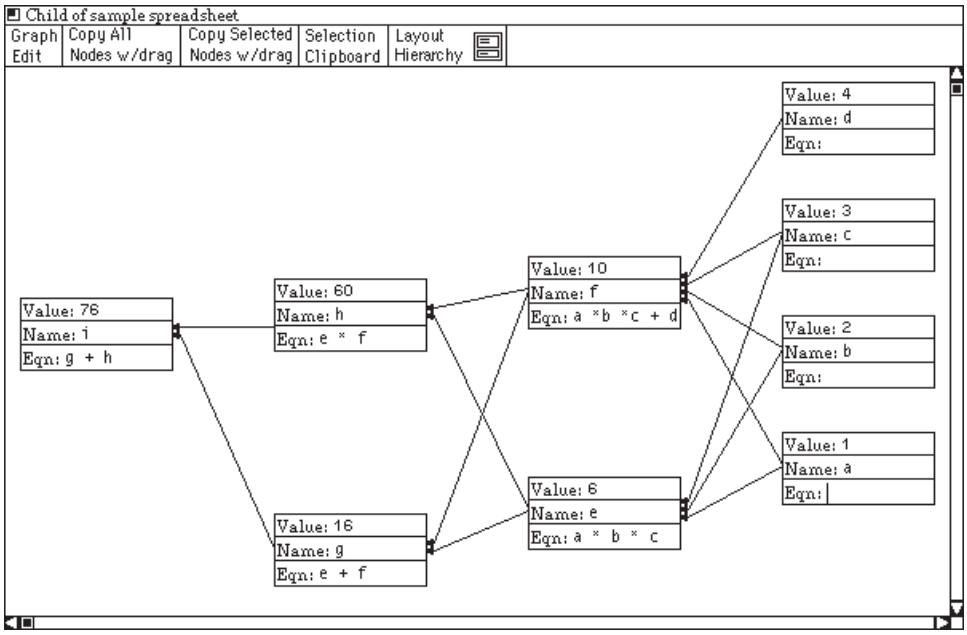

Figure 2: Spreadsheet Interface Depicting Cell Dependencies for Example in Figure 1

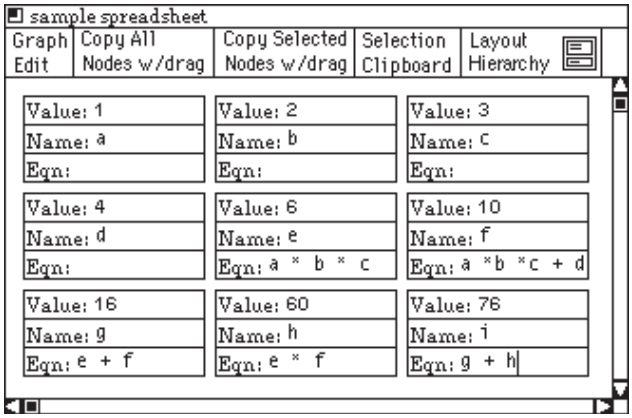

Figure 1: Traditional Spreadsheet Interface

spreadsheet shown in Figure 1. The cells in this interface are positioned in such a way as to reveal the equation dependencies between cells. The algorithm used to position the cells in this interface uses the equation in each cell to determine the dependency ordering of all the cells in the spreadsheet. It then positions the cells in a pattern that represents the dependency DAG. In addition to positioning the cells, the algorithm creates dependency edges between cells (drawn as arrows in Figure 2).

The interfaces in Figures 1 and 2 are live interfaces; they provide different views of the spreadsheet. Both are concurrently shown on the screen and the user can interact with either to change the spreadsheet data. This allows the user to look at both interfaces to understand the data and to interact with the interface that best fits his current task. In the example above, if the user wanted to edit all the cells that a given cell depends on, he might use the dependency oriented interface. On the other hand, if the user wanted to change all the cells grouped in a column, he would use the interface organized as a matrix.

4 Giving the User Control

Traditional user interface management systems have concentrated on providing the interface designer with very general tools for specify the interface. These tools provide the interface designer with the ability to precisely specify exact interfaces. The cost for this power is that the specification must be very exact—each interaction object must be considered.

The main goal of end-user controlled interfaces is to empower the user so he can customize the interface to match his current interests—which may in turn rely on the current data set and his mental model of the data. The solution is to provide the user with tools to create new views of the interface and to provide tools to customize these views.

The proposed mechanism is to format the interface using a general graph layout system that allows the user to customize the layout. Since general graph layout algorithms do not generally generate precise layouts, this approach does not provide the accuracy of traditional

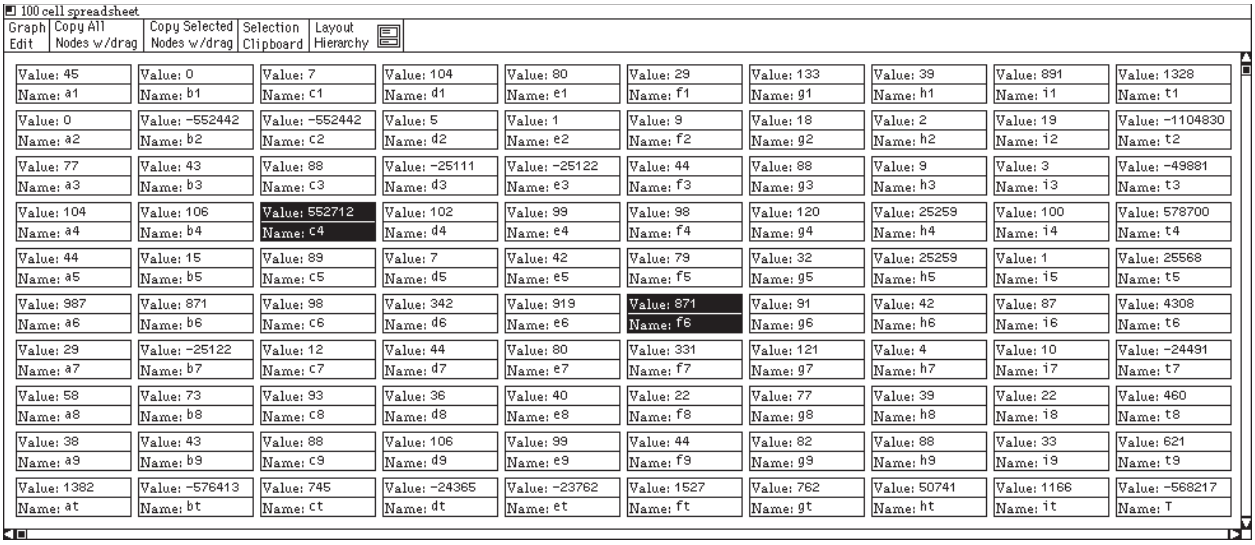

Figure 3: Spreadsheet with two Cells of Interest Highlighted

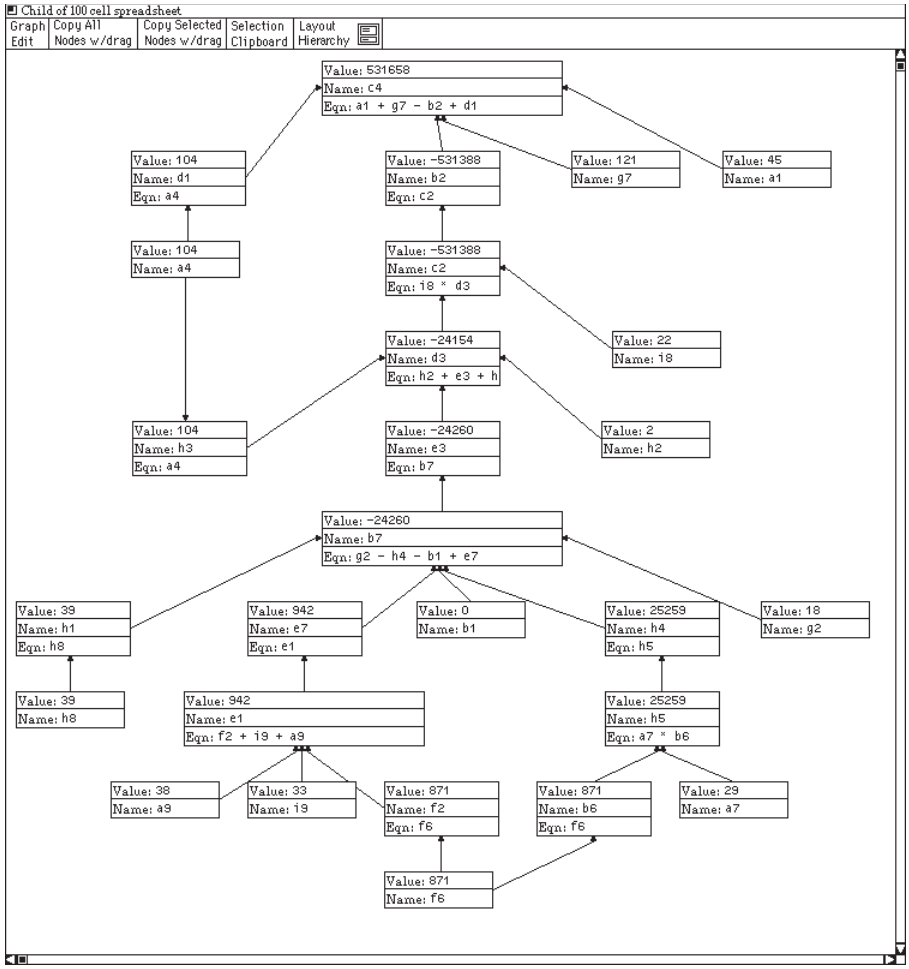

Figure 4: Dependency Graph for Cell "c4" in Figure 3

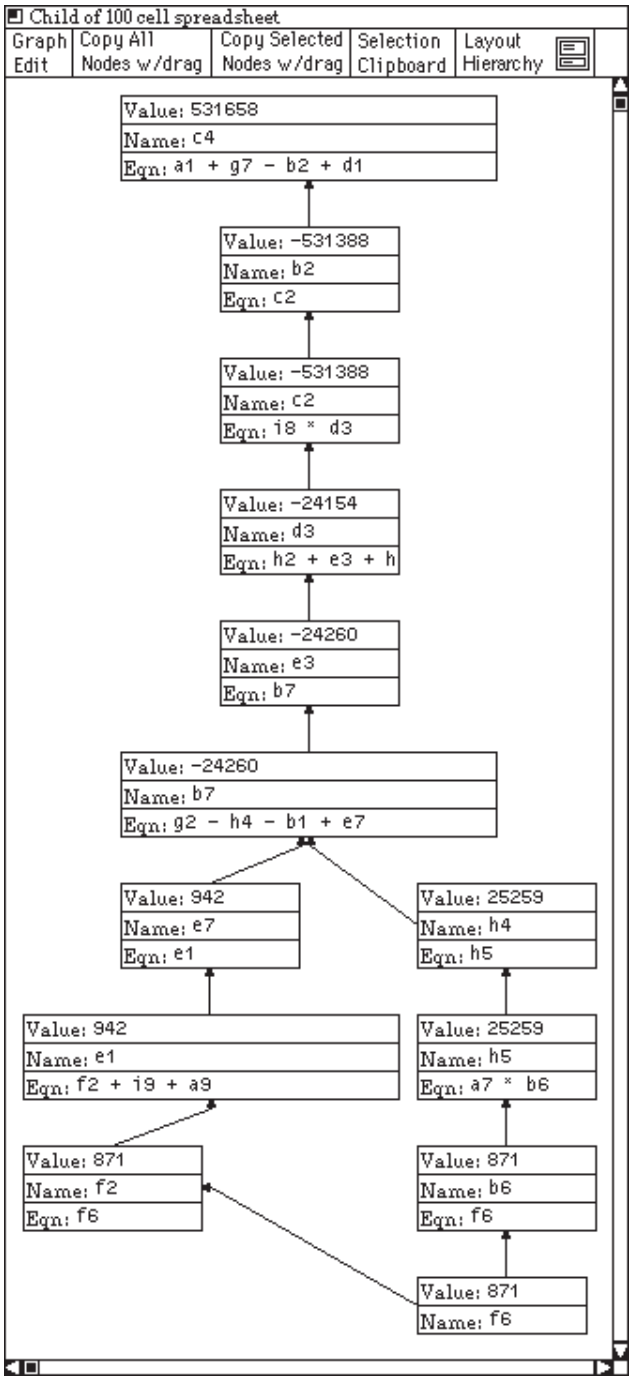

Figure 5: Pruned Dependency Graph

user interface management systems. However, it does empower the user with the tools to customize the interface.

The prototype interface layout system is built on top of the Vivid interactive graph layout system [7, 8]. Vivid extends traditional automatic graph layout algorithms to accept interactive parameters, thus allowing the user to interactively control the layout process. A direct manipulation interface allows the user to interactively create multiple custom views of a given graph.

As an example of end-user control, consider the spreadsheet shown in Figure 3. The cells in this example were interactively formatted to be drawn without their equations to save screen space. Since the layout of the interface is done algorithmically, when the interaction objects change their size—as in this example in which cells are drawn without their equation—the entire interface automatically adjusts. Thus when the user chooses a drawing options for a cell that changes its size, the layout algorithm automatically repositions all the cells.

Assume the user has discovered that when the value for cell "f6" is changed, the value in cell "c4" changes unexpectedly—these cells have been highlighted in Figure 3. The dependency graph for the spreadsheet would help the user examine the dependencies between the two cells of interest. However, the complete dependency graph is quite large and would not fit well on the screen. The solution is to allow the user to specify the portion of the spreadsheet he is currently interested in.

Figure 4 is an interface that includes only a portion of the original interface. It includes all cells in the spreadsheet that the cell "c4" depends on. Even this graph is a bit large and contains some information that does pertain to the user's current interest. Figure 5 shows another interface that includes only the cells that depend on "f6" and that "c4" depends on—all the paths between these cells. Since there is more space available, cells have been expanded in Figures 4 and 5 to include the equations allowing the user to interpret the dependency graph. Several of the cells have longer equations have been enlarged to show the entire equation. The resulting interface, while no longer general purpose, is dramatically better for the particular task at hand (debugging the connection between cells "f6" and "c4").

There are three basic mechanisms in Vivid that empower the user with the tools to customize the interface: creating multiple views, using different algorithms to format the interface, and interacting with parameters to the interactor objects and the layout algorithms.

Additional views can be constructed by first creating a new interface with a menu selection. When a new view is created, it is an empty interface without any interaction objects. The user must select interaction objects to be copied into the new interface. There are two methods for selecting interaction objects. The first—manual

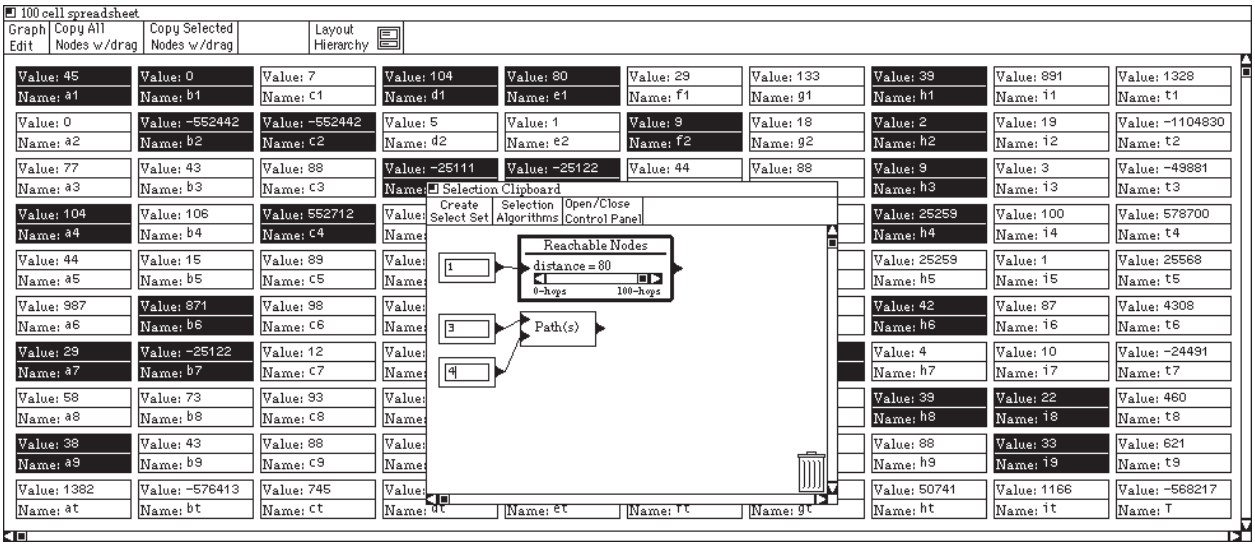

Figure 6: Selecting all Cells Reachable from "c4"

selection—is to select individual objects or groups of objects with the mouse. The second method—algorithmic selection—is to apply a selection algorithm to all the interaction objects in the interface. A visual programming language allows the user to combine multiple instantiations of selection algorithms. Figure 6 shows the selection interface and the selection programs used to select the cells pasted into the interfaces shown in Figures 4 and 5. (These cells were copied from the interface shown in Figure 3.)

The first program selects all the cells reachable from cell "c4." The object labeled "1" represents a manual selection set. These objects allow the user to manually select a set of nodes. In this example, the first manual selection set contains the single cell "c4." The next object represents a selection algorithm that selects all the cells within a given distance of the input set. Since the manual selection object is connected to the "reachable" object, its elements act as the parameters to the reachable algorithm. (The maximum traversal distance of the reachable algorithm can be set interactively through the distance slider. For this example is has been set to a value larger than any path in the spreadsheet). The user can now copy all the cells reachable by "c4" by selecting the "reachable" object (it and all the cells it represents are drawn highlighted when it is selected as shown in Figure 6) and then dragging the "Copy Selected Nodes w/Drag" button into the new view.

The second program calculates the paths between "c4" and "f6" as shown in Figure 5. This visual program consists of three objects. The first two are manual selection set objects labeled "3" and "4." In this example, the first manually selected set contains the single cell "c4." The second manual selection set contains the single cell "f6." The next object represents a selection algorithm that selects all paths between its input parameters. Figure 5 was created by selecting the "path(s)" object and dragging the "Copy Selected Nodes w/Drag" button into an empty view.

Since interfaces are formatted using general layout algorithms the user can change the format of the interface by using a different algorithm. In the example shown in Figures 3 and 4, each view uses a different layout algorithm. The interface in Figure 3 uses a grid layout algorithm that positions all the objects in a rectangular grid and the interface shown in Figure 4 uses a hierarchical layout algorithm based on ideas presented in [17] to position the interaction objects.

When the cells in this example were selected and copied into the new interface, they were automatically formatted using the grid layout algorithm. In general, when objects are pasted into a new interface, the algorithm that formatted them in the original interface is also pasted into the new interface. The user had to explicitly choose the hierarchical layout algorithm to create the interface shown in Figure 4.

In addition to creating new interfaces, the user can interact directly with the interaction objects and layout algorithms. Each layout algorithm has a set of parameters used during the layout process. A control panel is associated with each instantiation of a layout algorithm. This allows the user to fine tune the interface. For example, the orientation and spacing for the interface shown in Figure 4 was interactively changed—the default orientation is to draw the hierarchy horizontally as in Figure 2. It was changed to draw the tree vertically

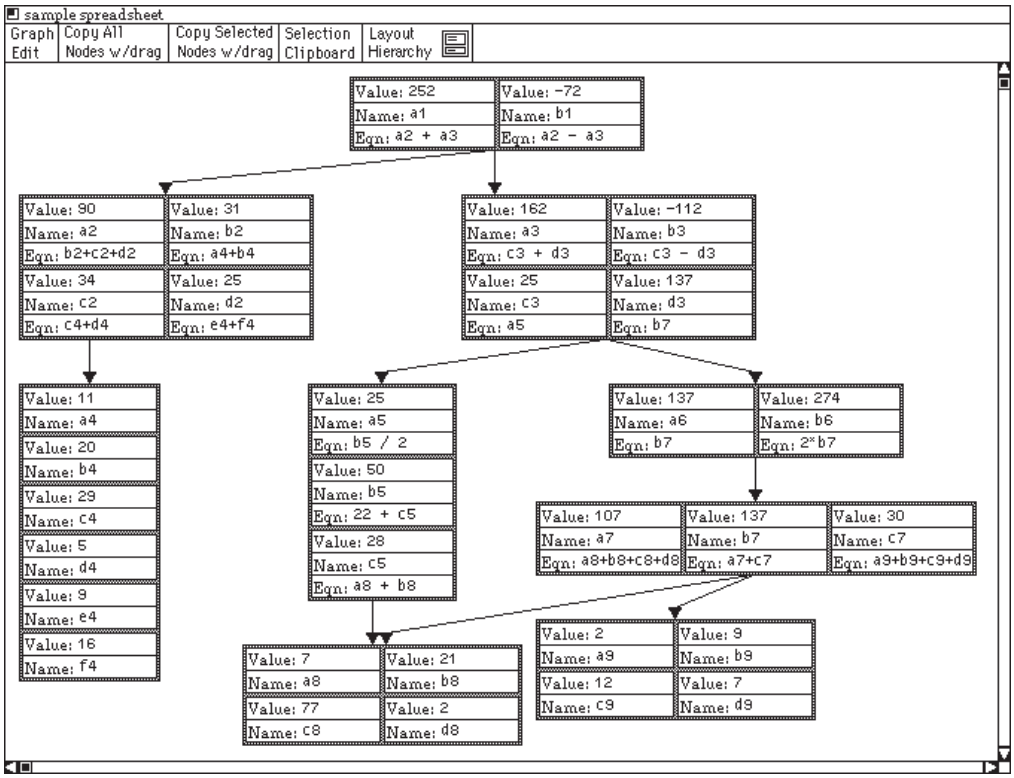

Figure 7: Interface Formatted by Hierarchy of Layout Algorithms

and increase the spacing to make the interface easier to read.

The main drawback of using layout algorithms to format the interface is that the user is constrained to using existing algorithms. Even if a large set of algorithms were introduced to the system, situations would arise in which the system could not format the interface in the manner the user desired. This problem is partially solved by allowing end-user to compose new layout algorithms out of existing algorithms [8].

New custom layout algorithms can be created by hierarchically combining existing algorithms. The basic idea is to standardize all layout algorithms so they can lay out a set of nodes and a set of layouts generated by other layout algorithms.. For example, consider the interface shown in Figure 7. It is formatted by a hierarchical collection of layout algorithms. Each group of cells is formatted by either a row, column or grid layout algorithm. All the groups of cells are in turn laid out by a hierarchical layout algorithm. The result is that the interface is laid out by a two level hierarchy of layout algorithms. The end-user creates new layout algorithms through the use of a direct manipulation editor.

5 Implementation

The example interfaces shown in this paper were built on top of the Vivid graph layout system [7]. Vivid is a general system that provides a direct manipulation interface to any given set of layout algorithms and interactor definitions.

No changes were made to Vivid to enable it to lay out interfaces—interaction objects are just a special case of Vivid graph objects. However, the spreadsheet interface did required the specification of the cell interactor. Each cell definition included functions for handling changes to the three text fields and included a general equation solving system for was accessed calculating and updating the value fields. The algorithms used to lay out the all the examples are basic Vivid layout algorithms.

6 Conclusion

The structure of a user interface serves as a presentation model of the underlying data set. While this model is often adequate, it forces the user to view the data in a particular form. Providing the user with the power to directly manipulate the structure of the interface allows him to create an interface that matches his individual

mental model, his current interests and changing data sets.

Using interactive graph layout algorithms to format the interface provides a flexible mechanism that empowers the end-user with the tools to customize the structure of the interface. In addition, graph layout algorithms can handle interfaces to applications with dynamic data sets.

The result is that the end-user can create interfaces that more closely match his expectations. This reduces the gap between the user and the actual data and increases the directness of the interface.

There are two main directions of future work. The first is to develop new layout and selection algorithms that are especially well suited for interfaces. This will provide the end-user with additional power to customize interfaces. The second direction is to develop a means by which the end-user can specify layout and selection algorithms. Once the user can create arbitrary algorithms, he will be able to create interfaces that meet his exact interests.

References

[1] Apple computer Company, *Inside Macintosh*, Addison-Wesley Publishing Company, Inc., (1982).

[2] Asente, P. J., Editing Graphical Objects Using Procedural Representations, *Digital Equipment Corporation Western Research Laboratory Research Report*, 87/6, (November 1987).

[3] Brooks, K. P., A Two-view Document Editor with User-definable Document Structure, *Ph. D. Dissertation, Department of Computer Science, Stanford University*, (May 1988).

[4] Card, S. K., Robertson, G. G., Mackinlay, J. D. The Information Visualizer, an Information Workspace, *Proceedings of ACM/SIGCHI*, pp. 181-188, (April 1991).

[5] Cardelli, L., Building User Interfaces by Direct Manipulation, *Proceedings of the ACM/SIGGRAPH Symposium on User Interface Software and Technology*, pp. 152-166, (October 1988).

[6] Henderson, D. A. Jr., Card, S. K., Rooms: The use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface, *ACM Transactions of Graphics*, vol. 5 No. 3, (July 1986).

[7] Henry, T. R., Interactive Graph Layout: the Exploration of Large Graphs, *Ph. D. Dissertation, Department of Computer Science, University of Arizona*, in preparation.

[8] Henry, T. R., Hudson, S. E., Interactive Graph Layout, to appear in *Proceedings of the ACM/SIGGRAPH Symposium on User Interface Software and Technology*, (November 1991).

[9] Hutchins, E. L., Holland, J. D., Norman, D. A., Direct Manipulation Interfaces, in *User Centered Systems Design,* Norman, D. A., Draper, S. W. (eds.), Lawrence Erlbaum Associates, Hillsdale, New Jersey, pp. 87-124, (1986).

[10] Myers, B. A., Creating User Interfaces by Demonstration, *University of Toronto Technical Report,* Ph.D. Thesis CSRI-196 (May 1987).

[11] Myers, B. A., *et. al.* The Garnet Toolkit Reference Manuals: Support for Highly–Interactive, Graphical User Interface in Lisp. Technical Report CMU-CS-98-196, School of Computer Science, Carnegie Mellon University, (November 1989).

[12] Linton, M. A., Vlissides, J. M., Calder, P. R., Composing User Interfaces with InterViews, *IEEE Computer,* vol. 22, no. 2, pp. 8-22, (February 1989).

[13] Olsen, D. MIKE: The Menu Interaction Kontrol Environment. *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 43-50, (1986).

[14] Robertson, G. G., Mackinlay, J. D., Card, S. K., Cone Trees: Animated 3D Visualizations of Hierarchical Information *Proceedings of ACM/SIGCHI*, pp. 189-194, (April 1991).

[15] Shneiderman, B., The Future of Interaction Systems and the Emergence of Direct Manipulation, *Behaviour and Information Technology*, vol. 1, pp. 57-69, (1982).

[16] Singh, G., Green, M. Chisel: A system for Creating Highly Interactive Screen Layouts, *Proceedings of the ACM/SIGGRAPH Symposium on User Interface Software and Technology*, pp. 86-94, (November 1989).

[17] Sugiyama, K., Tagawa, S., Mitsuhiko, T., Methods for Visual Understanding of Hierarchical System Structures, *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-11, No. 2, (February 1980).

[18] Szekely, P., Template–Based Mapping of Application Data to Interactive Displays, *Proceedings of the ACM/SIGGRAPH Symposium on User Interface Software and Technology*, pp. 1-9, (October 1990).