A Framework for Complexity Management in Graph Visualization

Ugur Dogrusoz and Burkay Genc

Computer Eng. Dept., Bilkent Univ., Ankara 06533, Turkey
Tom Sawyer Software, Oakland, CA 94612, USA

1 Short Description

We present a comprehensive framework for development of complexity management techniques in graph visualization tools. The presented architecture is capable of managing multiple associated graphs with navigation links and nesting of graphs as well as ghosting, folding and hiding of unwanted graph elements. The theoretical analyses show that the involved data structures and algorithms are quite efficient, and an implementation in a graph drawing tool has proven to be successful.

Our architecture is based on dynamic interactive compound graphs. The definition of compound graphs is extended to efficiently handle the graph editing and complexity management operations. A navigation forest is used to keep track of navigational links among nodes and graphs, and a nesting forest is used to keep track of nesting (inclusion) relations.

2 Complexity Management Operations

The framework manages complexity via the following instruments:

Expand/Collapse: Most applications require multiple levels of abstraction, where the user would like to visualize the information with varying levels of abstraction for different parts of the drawing. A *collapse* operation allows us to cancel a nesting relation between a node and a graph, thus avoiding the drawing of the graph inside the node. The *expand* operation is defined as the reverse operation creating a nesting relation between a node and a graph.

Folding/Grouping: A *fold* operation is applied to a group of graph members, and results in a new (folder) node and its new child graph with these members. At any time, an *unfold* operation may be applied on a folder node to reverse the effects of the fold operation.
Often times, members of a graph need to be put together according to some criteria to emphasize certain *grouping*. This can be achieved through folding followed by an expand operation, enabling all the group members to be gathered in the newly created graph.

M.T. Goodrich and S.G. Kobourov (Eds.): GD 2002, LNCS 2528, pp. 368–369, 2002.

Invisibility/Hiding/Ghosting: A graph member is said to be *invisible* when it is not rendered on the display yet it is part of the graph topology. *Hiding*, on the other hand, is used to avoid any means of user interaction on a set of graph members, and temporarily removes graph contents from the graph topology as well. Any set of graph members may later be *unhidden*. *Ghosting* can be used to visually decrease the importance of a graph member by means of changing its color and/or brightness of its skin, and sending it to the background. Unlike hiding, the member is still there, both visually and topologically.

3 Implementation

Our framework has been successfully implemented and integrated into Tom Sawyer Software's Graph Editor Toolkit for Java, version 5.0.

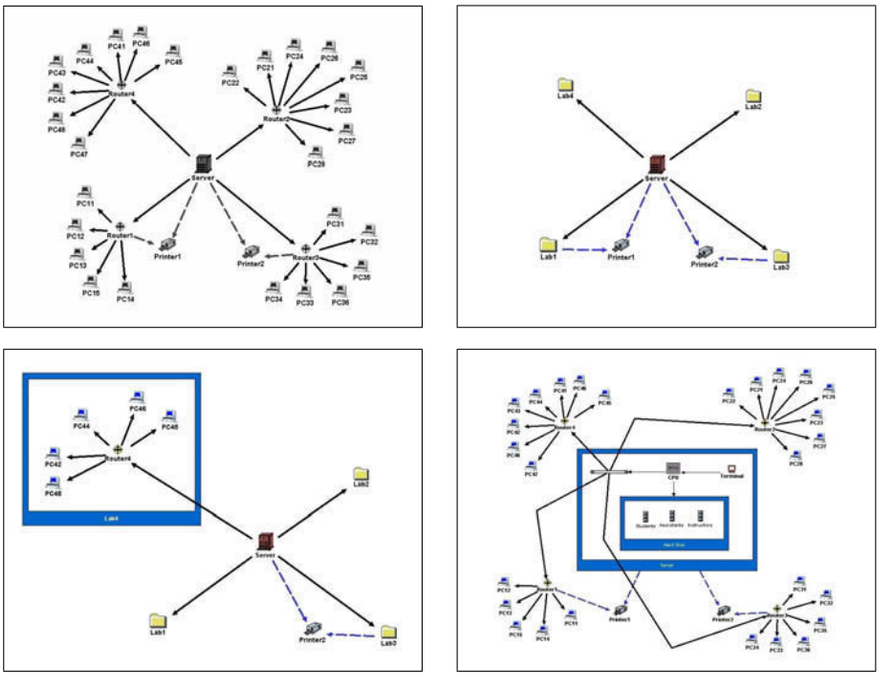

Fig. 1. *Upper Left:* Map of a small network drawing in GET for Java. *Upper Right:* All related network devices are grouped together under a folder. *Lower Left:* Lab4 has been expanded to reveal details. Additionally, Printer1, PC41, PC43 and PC47 are unavailable, so they are hidden. *Lower Right:* The varying levels of the details of the server has been input as deeply nested graphs.