Rigi: A Visualization Environment for Reverse Engineering

Margaret-Anne D. Storey
School of Computing
Simon Fraser University
Burnaby, BC, Canada
(250) 721-6019
mstorey@csr.uvic.ca

Kenny Wong
Dept. of Computer Science
University of Victoria
Victoria, BC, Canada
(250) 721-7294
kenw@csr.uvic.ca

Hausi A. Müller
Dept. of Computer Science
University of Victoria
Victoria, BC, Canada
(250) 721-7630
hausi@csr.uvic.ca

ABSTRACT
The Rigi reverse engineering system provides two contrasting approaches for presenting software structures in its graph editor. The first displays the structures through multiple, individual windows. The second (newer) approach, **S**imple **H**ierarchical **M**ulti-**P**erspective (SHriMP) views, employs fisheye views of nested graphs. We compare and contrast these two interfaces for visualizing software graphs, and provide results from user experiments.

Keywords
Fisheye views, graph editor, nested graphs, reverse engineering, software visualization.

INTRODUCTION
Graphs are particularly suitable for visually presenting software structure. Nevertheless, as the size of software systems increase, so too do their representations as graphs. Advanced graphics and abstraction techniques are needed to manage the visual complexity of these large graphs.

The Rigi reverse engineering system currently provides two solutions for browsing software subsystem hierarchies [1]. The first approach uses multiple, overlapping windows, where each window displays a portion of a subsystem hierarchy. A second (newer) approach, the Simple Hierarchical Multi-Perspective (SHriMP) visualization technique, presents software structures using fisheye views of nested graphs.

THE RIGI SYSTEM
In Rigi, parsing the subject software system results in a flat resource-flow graph that can be manipulated using a graph editor. The next phase is semi-automatic and involves pattern-recognition skills, where the reverse engineer identifies subsystems in the flat graph that form meaningful abstractions. These subsystems are collapsed to build multiple, layered hierarchies of abstractions (see Fig. 1).

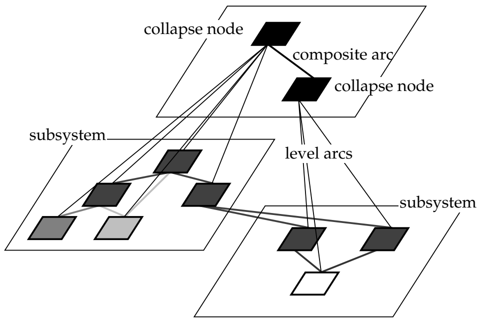
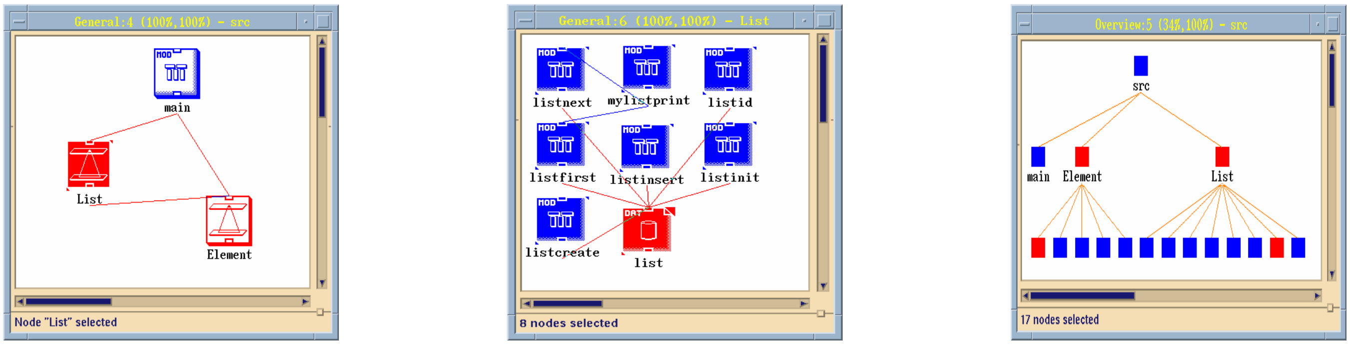


Figure 1: Rigi graph model

MULTIPLE WINDOWS
In the original Rigi approach, a subsystem hierarchy is presented using individual, overlapping windows that each display a specific slice of the hierarchy. For example, the user can open windows to display a particular level in the hierarchy, a specific neighborhood around a software artifact, a flattening of the hierarchy, or the overall tree-like structure of the entire hierarchy (see Fig. 2). However, with many open windows, users frequently become disoriented.

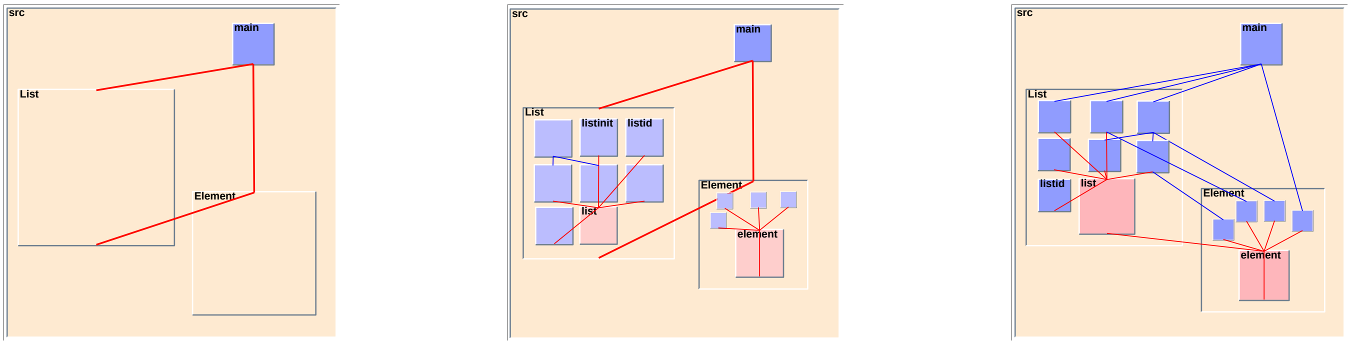
SHriMP VIEWS
The SHriMP visualization technique employs a nested-graph formalism and a fisheye-view algorithm for manipulating large graphs that provides contextual cues and preserves constraints such as orthogonality and proximity among individually resizable nodes. For Rigi purposes, the containment or nesting of nodes conveys the parent-child relationships in a subsystem hierarchy (see Fig. 3).

USER EXPERIMENTS
A small pilot study involving 12 users was conducted at the University of Victoria and Simon Fraser University according to an experiment design described in [2]. Three software browsing methods were evaluated (in this order): command-line tools (`vi` and `grep`), Rigi with multiple windows, and Rigi with SHriMP views.



(a) (b) (c)

Figure 2: (a) This window presents a main function and two subsystems List and Element which represent abstract data types. (b) The List node is opened to view its children, the list data type and access functions. (c) This *overview* window presents the subsystem hierarchy and provides context for the other windows.



(a) (b) (c)

Figure 3: (a) This window presents a main function and two subsystems List and Element as before. (b) The List and Element nodes have been opened to display their children and show an overview of the hierarchy. (c) Composite arcs have been opened to display the constituent lower-level dependencies.

Each user explored three game programs of varying size but similar complexity (in random order): Fish, Hangman, and Monopoly. Each user performed four high-level tasks (e.g., what does subsystem x do?) and four low-level tasks (e.g., find all artifacts that directly or indirectly depend on artifact x) with each interface. After the tasks, each user answered a usability questionnaire and participated in an informal interview.

Some findings found one interface less effective than another. For low-level tasks on the large Monopoly program, the command-line tools were worse than multiple Rigi windows ($P = 0.01$) and ShriMP views ($P = 0.0005$). For low-level tasks on the very small Fish program, the command-line tools and multiple Rigi windows were worse than SHriMP (by $P = 0.05$ and $P = 0.005$ respectively). Questionnaire results suggested that the users were more satisfied with the SHriMP interface than with multiple Rigi windows (at least when exploring small programs). When asked to hypothetically choose a user interface for their next software project, 8 users chose SHriMP.

SUMMARY

Rigi provides two interfaces for browsing software hierarchies. These two interfaces have recently been evaluated through some user experiments at the University of Victoria and Simon Fraser University. Early results and observations indicate that the two interfaces are effective for different types of program understanding tasks. We are currently planning further experiments to test this hypothesis.

REFERENCES

[1] M.-A. D. Storey, H. Müller, and K. Wong. Manipulating and documenting software structures. In P. Eades and K. Zhang, editors, *Software Visualization*. World Scientific Publishing Co., November 1996.

[2] M.-A. D. Storey, K. Wong, P. Fong, D. Hooper, K. Hopkins, and H. Müller. On designing an experiment to evaluate a reverse engineering tool. In *Proceedings of the 3rd Working Conference on Reverse Engineering*, Monterey, California, Nov 8-10, 1996.