GraVis — System Demonstration

Harald Lauer, Matthias Ettrich and Klaus Soukup

Universität Tübingen, Sand 13, 72076 Tübingen, Germany
Email: {lauer, ettrich, soukup}@informatik.uni-tuebingen.de

Abstract. GraVis is a powerful, interactive graph visualization system, designed to be generally usable in research and practical applications. The implementation of GraVis is based on a flexible object-oriented system architecture, portable to many platforms. The intuitive and efficient user interface is completed by the ability of the base system to meet the requirements of future applications.

1 Features of GraVis

Graph drawing research and practical applications working with structured data sets represented as graphs demand a powerful visualization tool. Common for both areas is, that the graph drawing tool must be interactive, extensible and flexible to cover the wide range of application domain specific requirements found in research and praxis. Together with a robust and maintainable realization, this would result in *the* ideal tool for graph drawing. GraVis aims to fulfill as many of the above requirements as possible.

Asking practitioners using graph visualization within their applications about their needs reveals a very broad spectrum of requirements covering performance, ease of use, graphical attributes, layout tools and many more. The requirements analysis and thus the design of GraVis is based on such a "wishlist" [9], therefore the design philosophy of GraVis is to be a generally usable tool for information visualization, only restricted to data representable as graphs. The following list shows a collection of the main features of GraVis realizing this design goal:

- Highly modularized architecture.
- Complete set of graphical attributes for all graph elements.
- Easy to use program interfaces to implement extension modules.
- Dynamic loading of extension modules at runtime.
- Multiple views on the same graph structure.
- Highly optimized visualization component.
- 2- and 3-dimensional graph visualization.
- Support for hierarchical graphs.
- Intuitive graphical user interface.
- Active nodes, which perform user defined actions when pressed.
- Unlimited undo and redo.
- Zoom to arbitrary levels.
- Various input/output formats (including GML [8] and PostScript).

There is a number of other grapheditors available, created mostly in the context of research projects. An example of a commercial product is the *Graph Layout Toolkit* [10] by Tom Sawyer Software. Other systems include, but are not limited to, Graphlet [8], daVinci [6], GD-Workbench [2] and the VCG tool [15].

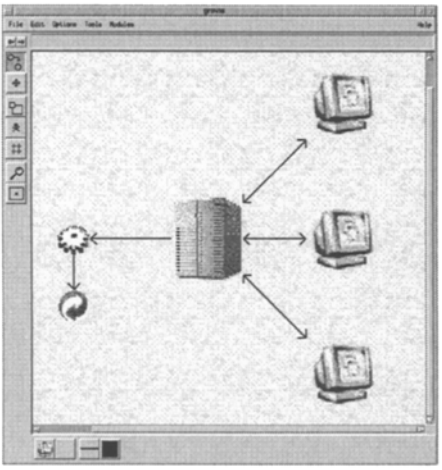

Fig. 1: The graphical frontend of GRAVIS displaying a graph using various bitmaps instead of the standard node types (circle, rectangle, rhomb, etc.).

2 Graphical User Interface

Much work has gone into the design of the user interface of GRAVIS, to produce an intuitively and efficiently usable tool for the interactive manipulation of arbitrary graphs. The philosophy of the user interface of GRAVIS is different from the traditional approach taken by tools like Graphlet [8] and the Graph Layout Toolkit [10] in one important aspect: although GRAVIS supports various input modes, the main mode for editing is *not* further split into modes for node/edge creation, attribute editing and more.

Instead, each frequently executed input or manipulation function is in the edit mode of GRAVIS directly accessible by a context sensitive mouse operation. The benefit of this approach is that there is no need for time consuming and concentration breaking mode changes, forcing the user to search for a "mode change" button if he for example wants to create nodes and edges in an arbitrary order. The *context* of the mouse event is sufficient to unambiguously decide which operation is requested by the user and is defined by

1. the mouse location (over a node, edge, bend, selection or free space) and
2. the mouse movement (a simple click or dragging the mouse after clicking).

Less frequently needed operations are accessible by the standard menu structure or by buttons arranged around the drawing area. Users of GraVis however, are seldomly forced to search for menu entries, since even functionality often found in additional panels are available directly at the object in the drawing canvas. Figure 2 for example shows how graph elements can be resized.

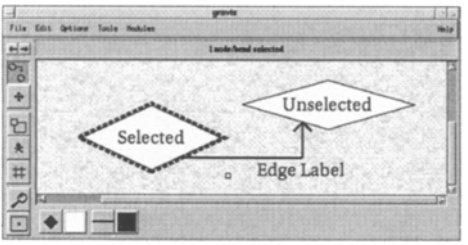

Fig. 2: Resize button in the lower right corner of the selected node, which allows resizing the node to an arbitrary height and width. The edge offsets can be moved from their default position (node center) using a similar mechanism.

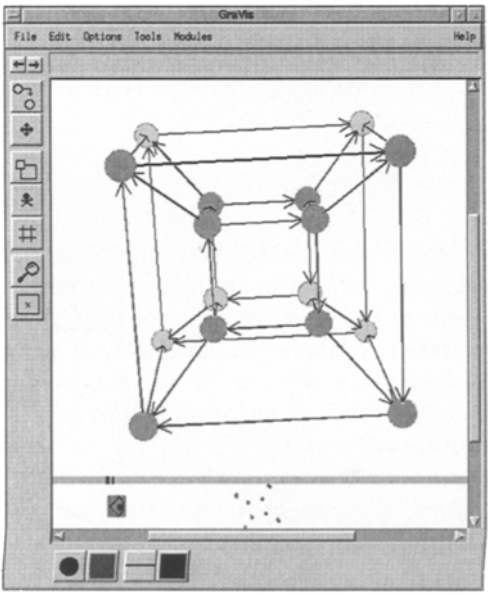

Fig. 3: The 3D graph viewer of GraVis. Below the drawing area is a mini-view of the graph, displaying the relative user position (indicated by the eye).

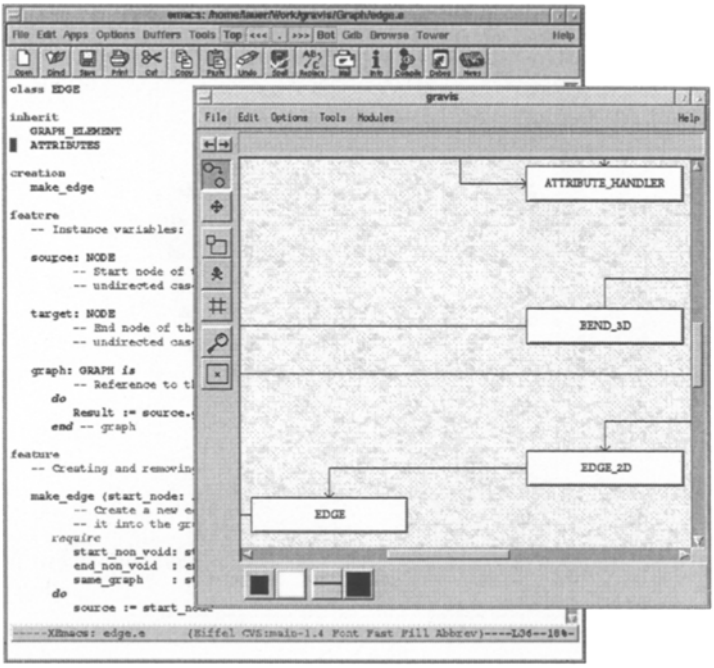

Fig. 4: Active nodes functioning as buttons with associated commands. GRAVIS can be used to layout object-oriented class diagrams and automatically load classes into an editor when the corresponding node is activated. In this case, the class **EDGE** (part of the *Graph* cluster of GRAVIS; the screenshot shows a fraction of the cluster) is now loaded in the editor after the node **EDGE** has been clicked.

3 Layout Modules

GRAVIS is dynamically extensible with new modules using well defined interfaces. Such modules can provide services of any kind, ranging from simple graph to sophisticated layout algorithms and even interfaces to external applications. A number of modules are included in the current version of GRAVIS and more are in development by the GRAVIS project group. Among these modules are:

- The layout algorithm for orthogonal, high-degree graphs *Kandinsky* [4].
- Interactive orthogonal graph layout based on [13] using extensions from [3].
- Symmetric layout (simplified version) based on [11] (see figure 5).
- Layout of series-parallel digraphs according to [1].
- Tree-based layout of general graphs (eg. for object-oriented class diagrams).
- Several graph generators.
- The Gem Springembedder [5] and a tree layout algorithm from GraphEd [7] demonstrating the interface to external applications and libraries.

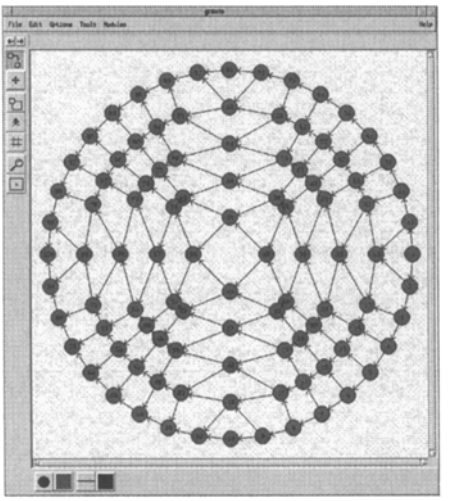

Fig. 5: A 10 × 10 grid drawn using the symmetric layout algorithm from [11].

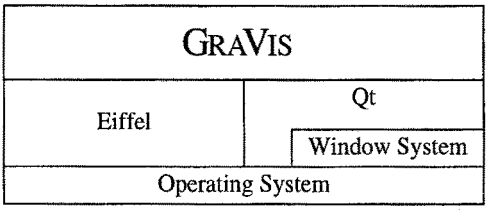

Fig. 6: Dependencies of GRAVIS on the programming language (Eiffel) and user interface toolkit (Qt). Details of the operating and the window system (eg. X Windows) are hidden from the implementation of GRAVIS.

4 Platforms and Portability

Realizing the object-oriented paradigm used in the design phase of GRAVIS, the system itself is implemented in the object-oriented programming language *Eiffel* [12]. Since one of the features of Eiffel is the abstraction of hardware or operating system specific issues and a well defined standard for kernel libraries, GRAVIS is portable to any platform supported by a compiler vendor, which covers most Unix systems and Microsoft Windows 95/NT.

Limiting the portability of graphical applications is often the access to window system functions for display purposes. GRAVIS however, uses the *Qt* library [14] for all user interface related parts of the system, which itself is available for many systems including the platforms mentioned above. The combination of the programming language and the user interface toolkit hides hardware and OS-specific details from the implementation of GRAVIS (see figure 6).

5 Conclusion

GraVis is freely available for research purposes and individual, non-commercial use. The most recent and all subsequent versions are available at our ftp site

`ftp.informatik.uni-tuebingen.de` in `/pub/PR/gravis`

Updated and more general information about the ongoing GraVis project can be found at the GraVis homepage located at the URL:

`http://www-pr.informatik.uni-tuebingen.de/Research/GraVis/`

Helpful comments and suggestions have been provided by Michael Kaufmann and Ulrich Fößmeier. This research was partially supported by DFG-Grant Ka812/4-1, "Graphenzeichnen und Animation".

References

1. P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. How to draw a series-parallel digraph. In *Scandinavian Workshop on Algorithm Theory*, volume 621 of *LNCS*, pages 272–283, 1992.
2. L. Buti, G. Di Battista, G. Liotta, E. Tassinari, F. Vargiu, and L. Vismara. GD-workbench: A system for prototyping and testing graph drawing algorithms. In *Graph Drawing*, volume 1027 of *LNCS*, pages 111–122, 1995.
3. U. Fößmeier. Interactive orthogonal graph drawing: Algorithms and bounds. To appear in Graph Drawing '97.
4. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In *Graph Drawing*, volume 1027 of *LNCS*, pages 254–266, 1995.
5. A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Graph Drawing*, volume 894 of *LNCS*, pages 388–403, 1994.
6. M. Fröhlich and M. Werner. Demonstration of the interactive graph visualization system daVinci. In *Graph Drawing*, volume 894 of *LNCS*, pages 266–269, 1994.
7. M. Himsolt. Graphed: A graphical platform for the implementation of graph algorithms (extended abstract and demo). In *Graph Drawing*, volume 894 of *LNCS*, pages 182–193, 1994.
8. M. Himsolt. The graphlet system (system demonstration). In *Graph Drawing*, volume 1190 of *LNCS*, pages 233–240, 1996.
9. H. Lauer. Grapheditoren — eine Wunschliste. Technical Report WSI–95–5, Universität Tübingen, 1995.
10. B. Madden, P. Madden, S. Powers, and M. Himsolt. Portable graph layout and editing. In *Graph Drawing*, volume 1027 of *LNCS*, pages 385–395, 1995.
11. J. B. Manning. *Geometric Symmetry in Graphs.* PhD thesis, Purdue University, 1990.
12. B. Meyer. *Eiffel: The Language.* 1992.
13. A. Papakostas and I. G. Tollis. Issues in interactive orthogonal graph drawing. In *Graph Drawing*, volume 1027 of *LNCS*, pages 419–430, 1995.
14. Qt. Information about Qt is available at http://www.troll.no.
15. G. Sander. Graph layout through the VCG tool. In *Graph Drawing*, volume 894 of *LNCS*, pages 194–205, 1994.