KGB
A Customizable Graph Browser

Hartmut Benz

University of Stuttgart, Department of Computer Science, Institute of Parallel and Distributed High Performance Systems, Breitwiesenstr. 20-22, D-70565 Stuttgart,
email: Hartmut.Benz@informatik.uni-stuttgart.de

Overview

This paper presents the architecture of a generic, customizable graph browser **KGB**[1]. The **KGB** has been designed to handle *very large* and *dynamically changing graphs* which are frequently used as repository management graphs in large applications. Special emphasis has been put on the *flexible presentation* of the information encoded in the graph and the *reduction of the user's workload* when adapting the presentation to his or her special needs. The **KGB** was built as a debugging and visualization tool for document hierarchies of the CASE tool **KOGGE**[2] currently under development at the "Institute for Software Engineering", University of Koblenz [3].

To automatically structure very large graphs and specifically select the subset of visible graph elements, the **KGB** implements three methods of abstraction. To achieve flexibility in the presentation, seperate visualization techniques can be applied to the attributes of each vertex and edge.

A *View into a graph* is defined by a set of applicable abstractions, presentation description and user interaction descriptions (e.g. the mapping from user generated input events to graph browser operations). A View is described as a simple rule based language. To handle dynamically changing graphs the View Description is interpreted by the **KGB** (rather than compiled into it).

Architecture

The architecture of the **KGB** is object oriented. Its primary constituents are a *source graph interface* which connects the **KGB** with the user's graph implementation [4, 6]. The *abstraction component* selects the elements of the graph to be visualized at a time and which parts of the graph can be grouped to increase readability. The *layout component* determines drawing positions as well as graphical representation for each visible graph element [5]. Finally, the *user interaction component* deals with the adequate graphical representation on the screen and the interpretation of user actions in the course of the dialog.

---

[1] German: *Konfektionierbarer Graphen-Browser*. The paper is a result of my diploma thesis at the University of Koblenz, Germany (cf. [1]).

[2] German: *Koblenzer Generator für Graphische Entwurfsumgebungen*, Koblenz generator for graphical software engineering systems

Abstraction

Common visualization techniques like windowing, zooming, fish-eye view or graph folding are not suitable to render graphs larger than a few thousand vertices and edges [7, 8, 9, 11]. Beyond this threshold either the required drawing area gets too large or the size of the graph elements approaches the size of a pixel. In any case the amount of the displayed information easily overloads human ability to successfully perceive and utilize it, so that other techniques have to be used.

Abstraction describes a method that hides irrelevant detail and by that enhances larger interrelations. It is therefore an adequate method to filter and structure large amounts of information. The **KGB** currently offers three variations of abstraction which are shortly described in the next three paragraphs.

Abstraction by Global Exclusion. This method permits to selectively exclude certain graph elements from browsing. It is based on the idea that a large and complex graph contains several semantic structures. By globally excluding those graph elements the user is currently not interested in, only the important structures are visualized. Global exclusion rules are defined intaractively as follows:

Graph elements of class $\mathcal{C}$ are hidden.

Abstraction by Local Restriction. This method permits to exclude graph elements that are irrelevant to the user's current focus of attention. The method is based on the idea of locality, the observation that semantically closely related graph elements are commonly "only a few incidences apart". Nevertheless, this abstraction method puts a high cognitive load on the user to keep a mental map. Furthermore, a positionally stable layout is necessary since moving the focus of attention introduces problems similar to those of interactive graph editing. Local restriction rules are defined interactively as follows:

Graph elements of...
class $\mathcal{C}$ have distance 1.
class $\mathcal{E}$ have distance 0.7.
Maximum distance is 3;

Abstraction by Subgraph Abstraction. This method permits to select (semantically connected) subgraphs and represent them each by a single abstraction element. The element retains the information: "there is a subgraph". Subgraph abstraction is a common feature in graph visualization and editing tools. Alas, the usual interactive selection with a mouse is inefficient for very large graphs and completely useless for frequently changing graphs.

This problem can be solved by automatic subgraph selection mechanisms. The **KGB** currently provides two rule-driven selection engines. Hard-coded algorithms, although more efficient, have been postponed due to their inflexibility. They can, of course, be used to solve specialized selection tasks. Subgraph abstraction rules are defined intaractively as follows:

Abstraction "Dir" from a Directory_Vertice with all inbound adjacencies;

The graph browser currently uses two subgraph description languages which will not be described any further in this paper. Both languages are computable in polynomial time (ref. [1, 2]).

Subgraph abstraction obviously introduces a *hierarchical structure* into the graph. As opposed to the other methods, subgraph abstraction can be applied to a graph more than once. This has three important consequences:

- Subgraph abstraction has to be defined on hierarchical graphs.
- Repeated application can lead to arbitrarily deep abstractions. The abstraction hierarchy has the structure of a tree.
- The resulting tree depends on the sequence of abstractions.

Information visualization

Generally, a graph contains global and local information. *Global* or structural information consists of the graph elements (their existence) and their incidence structure. The visualization of this information is performed by placing the graph elements and is widely researched as *graph layout.*

Local information consists of the attributes that graph elements are quite regularly annotated with. The visualization of this information is often crucial to the user's ability to understand the semantics of the graph at all.

The **KGB** acknowledges this importance. The user can freely and interactively describe which (subset of the) attributes of a graph element are to be visualized and which graphical representation they should receive. The **KGB** supports a great variety of graphical representations to choose from. For example:

edges Class_A: use Name as text label;
edges: label color from Int2Col(Weight);

View into a Graph

A *View into a graph* consists of a set of rules defining possible abstractions and visual representations of attributes. In addition to these definitions, there are rules to choose one (of many) layout algorithms and to associate user actions to graph browser operations [10]. For example:

Layout Sugiyama("2,2,1");
double-click-right calls ShowDetailedInformation;
<RETURN> on Abstraction_vertex calls OpenAbstraction;

Views can be named and the user can easily create, modify or switch between them while using the **KGB**.

Genericity

The amount of source code required to adapt the tool to a new task has been minimized. The complete adaptation (including source code fragments) is described using a simple abstract language. From this description a *graph browser compiler* automatically generates a dedicated **KGB**. Thus, the **KGB** is a generic tool.

Future Work

Primary objective of the future development is the portation from NeXTSTEP to X-Windows. Future development will concentrate on improving the user interface and extending the abstraction methods. A user interface supporting more direct manipulative techniques is highly desirable. Additional abstraction methods like *fish-eye view* or *concentrator vertices* [11] should be included. Furthermore, the subgraph selection methods used in subgraph abstraction should be combined with the interactive graph layout proposed in [7]. In addition, different approaches to visualize the hierarchies introduced by the subgraph abstraction should be explored [9].

References

1. Benz, H.; "*KGB – Ein konfektionierbarer Graphen-Browser*"; Diplomarbeit an der Universität Koblenz-Landau, Abt. Koblenz, FB Informatik, 1993
2. Capellmann, C.; Franzke, A.; "*GRAL—Eine Sprache für die graphbasierte Modellbildung*"; Diplomarbeit an der Universität Koblenz, FB Informatik, 1991
3. Carstensen, M.; Meissner, A.; Rhein, U.; "*Forschungsschwerpunkt CASE. Dritter Zwischenbericht.*"; Universität Koblenz, 1991
4. Dahm, P.; Ebert, J.; Litauer, C.; "*Benutzerhandbuch EMS-Graphenlabor V3.0*"; Koblenz 1994; (Unpublished manuscript)
   `ftp://ftphost.uni-koblenz.de/outgoing/GraLab/`
5. Di Battista, G.; Eades, P.; Tamassia, R.; "*Algorithms for Drawing Graphs: An Annotated Bibliography*" `ftp://wilma.cs.brown.edu /pub/papers/compgeo/gdbiblio.*`
6. Ebert, J.; "*A Versatile Data Structure For Edge-Oriented Graph Algorithms*"; Communications of the ACM (6) 1987
7. Henry, T. R.; Hudson, S. E.; "*Interactive Graph Layout*"; Proceedings of the ACM SIGGRAPH Symposium on User Interface Software, 1991
8. Himsolt M.; "*GraphEd: An Interactive Graph Editor*"; Proc. STACS 89, Lecture Notes in Computer Science, vol. 349, pp. 532-533, Springer-Verlag, 1989.
   `ftp://ftp.forwiss.uni-passau.de/pub/local/graphed`
9. Johnson, B.; "*TreeViz: Treemap Visualization of Hierarchically Structured Graphs*"; in [12], pp. 369–370
10. Paff, G. (Editor); "*User Interface Management Systems*"; Proc. of the Workshop on User Interface Management Systems held in Seeheim, FRG, Nov. 1-3, 1983, Springer 1985
11. Paulisch, F. N.; Tichy, W. F.; "*EDGE: An Extendible Graph Editor*"; in Software–Practice and experience, Vol. 20(S1), June 1990; John Wiley & Sons, Ltd.
12. "*Proc. of CHI 1992 (Monterey, California, May 3–7, 1992)*"; ACM, New York, 1992