

Frances Newbery Paulisch

Cc 01-704

The Design of an Extendible Graph Editor

BIBLIOTHEQUE DU CERIST

Springer-Verlag

Berlin Heidelberg New York

London Paris Tokyo

Hong Kong Barcelona

Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Author

Frances Newbery Paulisch
Siemens AG, ZFE BT SE 32
Otto-Hahn-Ring 6, D-81739 München, Germany
E-mail: paulisch@ztivax.zfe.siemens.de

6342

CR Subject Classification (1991): D.2.2, I.3.4, D.1.5, G.2.2, H.2.3, D.2.m

ISBN 3-540-57090-X Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57090-X Springer-Verlag New York Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1993
Printed in Germany

Typesetting: Camera ready by author
45/3140-543210 - Printed on acid-free paper

Preface

Graphs can be used to convey information about relationships in many applications. State transition diagrams, PERT/CPM charts, call graphs, and entity-relationship diagrams are a few examples of many applications involving graphs. Typically, nodes in the graph represent items in the application (e.g. a state, an activity, a program module) and the edges represent the relationships among these items (e.g. state transition, activity duration, procedure invocation). A graph editor is an interactive tool that presents a graph to the user pictorially and allows the user to edit the graph. The recent proliferation of graph editors for particular applications indicate their effectiveness as the graphical user interface to an application. Many designers, however, are hesitant to use the graph editor model because of the high cost of developing such a graphical user interface.

This book presents the design of an extendible graph editor, which is a graph editor that can be adapted easily to many different application areas. The advantages of using a graph editor will thus be available for a minimal customization effort. Several fundamental and recurring problem areas associated with graph editors are investigated and a solution is proposed for each. The specific topics investigated are:

- **Graph layout:** How can application-specific layout requirements, individual preferences, and layout stability be integrated with layout algorithms? A layout constraint mechanism is presented which can easily be combined with various graph layout algorithms.
- **Graphical abstraction:** How can users deal with large graphs containing hundreds of nodes and thousands of edge crossings? A novel clustering technique called *edge concentration* is presented which can reduce the apparent complexity of the graph. Alternatively, a subgraph can be specified and viewed as a multi-level graphical abstraction either in the context of the graph or in a separate view.
- **Persistence:** How can the graph structures produced by the editor be kept in long-term storage, especially if the node and edge data structures have been extended for a particular application? The proposed solution uses a standardized, external format for graphs. A program generator tool reads the graph, node, and edge class declarations and automatically generates routines for reading, writing, and editing these data structures.
- **Extendibility:** How should the editor kernel be structured to be adaptable to various applications? The object-oriented design of the proposed graph editor makes it easy to adapt.

To demonstrate their feasibility, the proposed solutions have been incorporated into EDGE, an extendible graph editor prototype. EDGE has been

adapted to a number of applications including: a browser for entity-relationship diagrams, a tool for visualizing software configurations, a PERT chart editor, a call graph animator, a directory editor, and a logic simulator.

This document is a revised version of my doctoral dissertation from the Faculty of Informatics at the University of Karlsruhe (FRG) presented on May 7, 1991. I thank my advisor, Prof. Dr. Walter F. Tichy, for his invaluable guidance over the past years as well as my dissertation's co-referee, Prof. Dr. A. Schmitt.

I particularly thank Karl-Friedrich Böhringer, Stefan Manke, and Stefan Strugies for their valuable contributions to EDGE. Special thanks go to Bala Krishnamurthy, without whose encouragement I might never have made it this far. The support of my family was an important factor in completing this work.

Finally, I would like to thank the EDGE users at companies, research institutions, and universities in the United States and Europe. Their strong interest lends credence to my claim that an extendible graph editor is an appropriate graphical user interface to a wide range of applications.

Frances Newbery Paulisch
June 1993

Contents

1	Introduction	1
1.1	Motivation and Goals	5
1.2	Research Contributions	7
1.3	Organization	8
2	Definition of Terms	11
2.1	Graphs	11
2.2	Graph Editors	13
3	Related Work: Graph Editors	15
3.1	Special-Purpose Graph Editors	16
3.2	General-Purpose Graph Editors	23
3.3	Extendible Graph Editors	26
3.4	Summary	33
4	Layout Algorithms and Layout Constraints	35
4.1	Related Work	35
4.1.1	Planar Graphs	38
4.1.2	Undirected Graphs	39
4.1.3	Trees	40
4.1.4	Directed Graphs	41
4.2	Layout Constraints	45
4.2.1	Low-Level Constraints	47
4.2.2	Constraint Manager	47
4.2.3	Three-Dimensional Constraints	48
4.2.4	Integration with Layout Algorithms	49
4.2.5	Examples	50
4.2.6	Results	53
4.3	Layout Stability	53
4.3.1	Examples	55
4.3.2	Results	69

5	Graphical Abstraction	73
5.1	Related Work	73
5.2	Subgraph Abstraction	76
5.2.1	Representation	76
5.2.2	Definition	80
5.2.3	Examples	81
5.2.4	Results	81
5.3	Edge Concentration	81
5.3.1	Representation	82
5.3.2	Definition	83
5.3.3	The Complexity of the Edge Concentration Problem	84
5.3.4	An Approximate Solution	86
5.3.5	Examples	90
5.3.6	Results	99
6	Persistence	101
6.1	Related Work	102
6.1.1	Overview of Methods to Achieve Persistence	102
6.1.2	External Formats for Graph-Based Tools	104
6.1.3	External Formats for Other Tools	106
6.2	GRL: An External Representation for Graph-Based Tools	109
6.3	Language Extension	113
6.4	Examples	114
6.5	Results	117
7	Extendibility	119
7.1	Related Work	119
7.2	An Object-Oriented Graph Editor	121
7.2.1	Graph Class	122
7.2.2	Node Class	124
7.2.3	Edge Class	125
7.3	AGENT: A Tool to Automate Extendibility	125
7.3.1	Input Routines	127
7.3.2	Output Routines	127
7.3.3	Menu Routines	127
7.3.4	Constructors and Destructors	127
7.3.5	Class Hierarchy	128
7.3.6	Exception List	128
7.4	Example	129
7.5	Results	131

8	EDGE: An Extendible Graph Editor	133
8.1	Graph Layout	135
8.2	Graphical Abstraction	136
8.2.1	Subgraph Abstraction	136
8.2.2	Edge Concentration	137
8.3	Persistence	137
8.4	Extendibility	138
8.4.1	Changing Visual Appearance	138
8.4.2	Interfacing EDGE with an Existing Application	139
8.4.3	Integrating EDGE with a New Application	139
8.5	Implementation	139
8.6	Examples	140
8.6.1	A Browser for Entity-Relationship Diagrams	140
8.6.2	A Tool for Visualizing Software Configurations	141
8.6.3	A PERT Chart Editor	143
8.6.4	A Call Graph Animator	145
8.6.5	A Directory Editor	146
8.6.6	A Logic Simulator	148
8.7	Results	150
9	Summary and Future Research	153
9.1	Graph Layout	153
9.2	Graphical Abstraction	156
9.3	Persistence	158
9.4	Extendibility	159
9.5	Other Areas	160
10	Conclusion	163
	Appendix	167
A	EBNF Grammar for GRL	167
	Bibliography	171
	Index	183

List of Figures

1.1	Textual and pictorial representations of a graph	1
1.2	Development of the Xerox Star (EDGE graph editor)	3
1.3	Development of the Xerox Star (from [JRV89])	4
1.4	Graph editor depicting the import/export relations between modules	5
2.1	Level hierarchy of a directed acyclic graph	13
3.1	GINCOD editor (from [TBT83])	17
3.2	Software Through Pictures editor	18
3.3	PROSPEC editor	19
3.4	ParaGraph editor	20
3.5	GERM editor	21
3.6	VIFOR editor	22
3.7	GRAB editor	23
3.8	DRAG graph drawing program	24
3.9	DAG graph drawing program	25
3.10	ISI editor (from [Mes89])	27
3.11	GraphEd editor	28
3.12	Kb-edit editor	30
3.13	GMB editor	31
3.14	GraphView editor	32
4.1	Planar layout can eliminate crossings	37
4.2	Planar drawings: straight line, and convex (from [CON85])	38
4.3	Planar drawings: grid and visibility diagram (from [ET89])	39
4.4	Tree drawings: conventional, radial, contour (from [ET89])	40
4.5	Level assignment can affect total edge length	44
4.6	Level assignment can affect number of crossings	44
4.7	Coordinate system	45
4.8	Overview of constraint manager architecture	46
4.9	Constraint example: family tree	51
4.10	Constraint example: PERT chart	51

4.11	Constraint example: UNIX tools (without and with constraints)	52
4.12	Examples of layout constraints and instability	57
4.13	Example from 4.12, but with stability (radius 0)	58
4.14	Example from 4.12, but with stability (radius 1)	59
4.15	"World" example of layout stability: Before	60
4.16	"World" example of layout stability: Instable	61
4.17	"World" example of layout stability: Stable	62
4.18	"UNIX" example of layout stability: Before	63
4.19	"UNIX" example of layout stability: Instable	64
4.20	"UNIX" example of layout stability: Stable	65
4.21	"W2t" example of layout stability: Before	66
4.22	"W2t" example of layout stability: Instable	67
4.23	"W2t" example of layout stability: Stable	68
5.1	Display of a compound digraph (from [SM91])	75
5.2	Black-, grey-, and white-box views of a subgraph abstraction	77
5.3	Adding edge causes unnecessary edge crossing	78
5.4	Multi-level hierarchical subgraph abstraction	79
5.5	Separate view of a subgraph abstraction	82
5.6	A graph and its representation using an edge concentration	83
5.7	Comparison of two coverings by complete bipartite subgraphs	85
5.8	Counter example	89
5.9	Series of edge concentrations	91
5.10	Texchk program - includes relation	92
5.11	Calls program - define/use relation	94
5.12	W2t program - calls relation (before)	95
5.13	W2t program - calls relation (after)	96
5.14	Xcal program - includes relation	97
5.15	Fig program - includes relation	98
6.1	DRAG examples (from [Tri88])	104
6.2	DAG example (from [GNV88])	105
6.3	IDL overview (from [Lam87])	107
6.4	IDL example (from [Lam87])	108
6.5	Expression evaluation graph	113
7.1	AGENT program generator tool	126
7.2	Two bit comparator application	130
7.3	GRL input for two bit comparator application	131
7.4	Class declarations for two bit comparator application	132
8.1	Terminal session showing the EDGE graph editor	134
8.2	EDGE layout algorithms: Sugiyama, tree, undirected planar	135
8.3	A browser for entity-relationship diagrams	140
8.4	A tool for visualizing software configurations	142

8.5	A PERT chart editor	143
8.6	A call graph animator	145
8.7	A directory editor	147
8.8	A logic simulator	149
9.1	Traditional representations	155
9.2	Single- and bi-directional layout	157

List of Tables

3.1	Comparison of graph editors	33
4.1	Layout time comparison	53
4.2	Stable vs. instable layout: edge additions ("World")	70
4.3	Stable vs. instable layout: edge additions ("UNIX")	70
4.4	Stable vs. instable layout: edge additions ("W2t")	71
5.1	Effectiveness of edge concentration algorithm	99
6.1	Standard set of graph attributes	111
6.2	Standard set of node attributes	111
6.3	Standard set of edge attributes	112
6.4	Standard set of layout constraint attributes	112
8.1	Customization effort	150

Chapter 1

*"A picture is worth
a thousand words."*
- Anonymous

Introduction

The recent proliferation of high quality graphics workstations has been closely followed by interactive tools that present information to the user graphically rather than using traditional, textual representations. A graphical user interface makes tools easier to learn, use, and understand because humans recognize patterns better when they are presented pictorially. In [Rob87], Robins gives a compelling example of why "a picture is worth a thousand words". Here, two representations of a graph are given – one as a list of edges and the other as a drawing of the graph (see figure 1.1¹). Important properties of the graph – that it is a binary tree and that "K" is the root of the tree – are immediately obvious from the drawing. The list of edges contains the same information, but the user has to consider each edge and compute the transitive closure (possibly sketching a drawing in the process) to extract this information.

List of edges:

$(K,B), (K,C),$
 $(B,D), (B,E),$
 $(D,H), (E,I),$
 $(C,F), (C,G),$
 $(F,J), (G,A)$

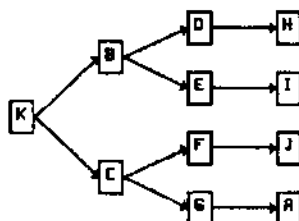


Figure 1.1: Textual and pictorial representations of a graph

There are many different ways of presenting information graphically, and one of the most general is to represent the information as a graph. Informally, a

¹Unless otherwise noted, the graphs shown as examples are drawn by the EDGE graph editor described in chapter 8.

graph consists of a set of nodes and a set of edges. Each node typically represents some object and the edges represent binary relationships between these objects. Information can be associated with the nodes and edges of the graph. Graphs are used to convey physical or conceptual information in many different application areas. The following lists a few applications of graphs in computer-related fields:

- **Software:** Graphs are used in all phases of software development from flowcharts, data structure animation, data flow diagrams, finite state automata, petri nets, and syntax graphs to call graphs and software configuration dependency graphs. They are also heavily used in the relatively new area of visual programming [Shu89, Gli90].
- **Hardware:** Computer hardware gates can be interconnected to form combinational logic networks.
- **Database:** An entity-relationship diagram [Che76, Gan90], commonly used for the conceptual design of database schemas, is a graph consisting of entities, relations, and attributes. The user interface of hypertext systems [Con87] is often based on graphs.
- **Networking:** Graphs are used to display network configurations where nodes represent machines and edges the physical connection between them. Reachability graphs are used to verify communication protocols [CL88].
- **Artificial Intelligence:** Semantic nets used to represent knowledge [Bra79].
- **Business:** PERT and CPM charts [CCP87], used in the area of project management, are graphs that help a project manager visualize the dependency relationships among various subprojects.

The terminology used in the application areas listed above indicates a wider variety than is actually the case. Syntax trees, entity-relationship diagrams, networks, semantic nets, PERT charts and the rest - all are different forms of graphs.

Just as graphs provide a general representation of information, editing is a general model of interaction for user interfaces. In [DS90] it is argued that any interactive application could present a graphical representation of its data and allow the user to edit it and to update the representation. For example, a mail program could present a graphical representation of a mailbox which the user would edit to read or send mail messages. *Direct manipulation* [Shn83], is a particular form of interaction in which the user specifies objects by selecting them "directly" on the screen using a pointing device (e.g. a mouse) rather than specifying them "indirectly" (e.g. by name).

A *graph editor* is an interactive tool that presents a graph to the user pictorially and allows the user to edit the graph. The user can add, delete, or edit

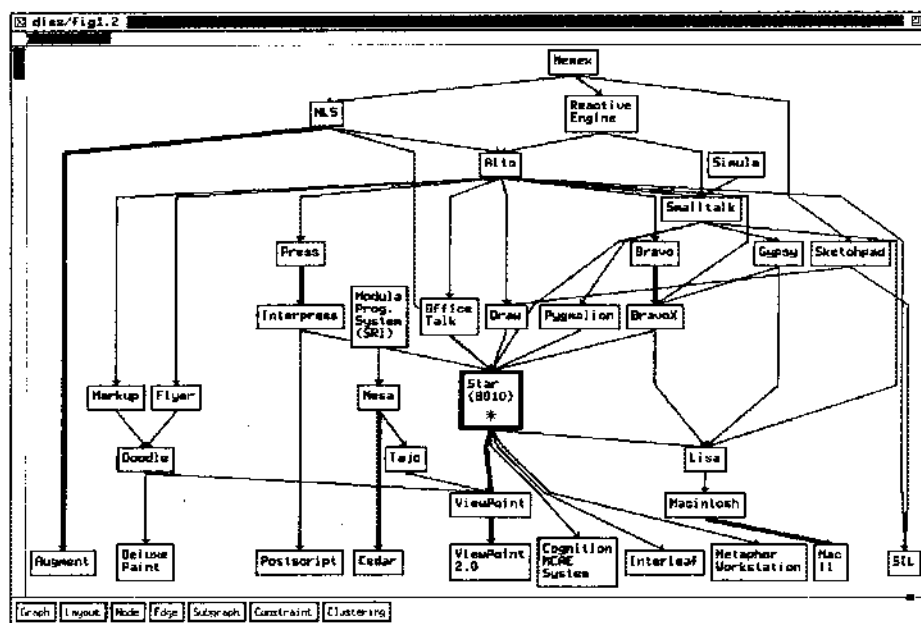


Figure 1.2: Development of the Xerox Star (EDGE graph editor)

nodes and edges in the graph and the changes will be reflected in the display of the graph. A graph editor is a powerful and widely-applicable tool because it combines a general graphical representation of information (a graph) with a general model of interaction (an editor).

Graph editors can support a wide range of user interaction. Figures 1.2 and 1.4 show examples of two extremes – one with little or no editing of the graph and the other with frequent and continuous updates.

The graph editor shown in figure 1.2 depicts the development of the Xerox Star, a personal computer designed for use by business professionals in an office environment. This information was extracted from a (presumably manually-drawn figure) given in [JRV⁺89] and shown in figure 1.3. The graph shows how related systems influenced each other (the thick lines represent direct successors of a system). In this example, the information being displayed is relatively static and little or no editing of the graph is required. The placement of the nodes and edges in figure 1.2 is done automatically as opposed to being positioned manually by the user. The layouts are of comparable quality. Judging by one of the often-used concrete measures of layout quality, the number of edge crossings, the layouts are equally good (both have 24 crossings). However, reliance on the automatic layout of the graph (which takes 15 seconds on a Sun 3 workstation) is surely faster than a manual layout. This example shows the benefit of using

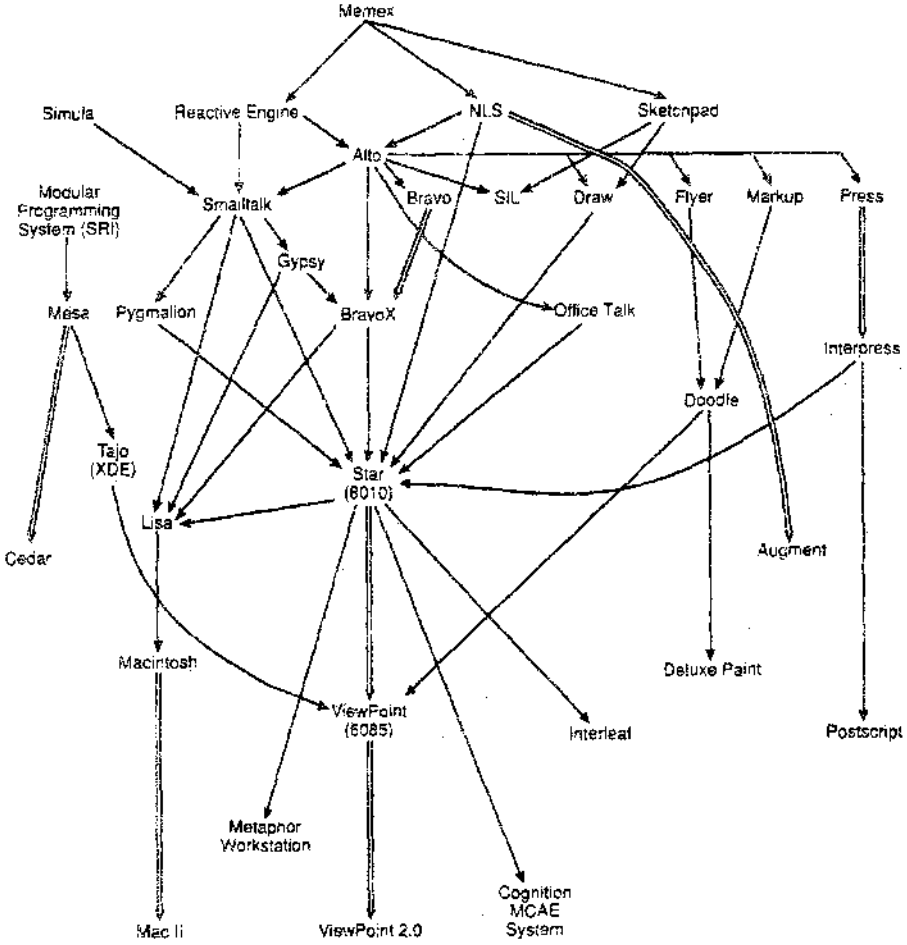


Figure 1.3: Development of the Xerox Star (from [JRV89])

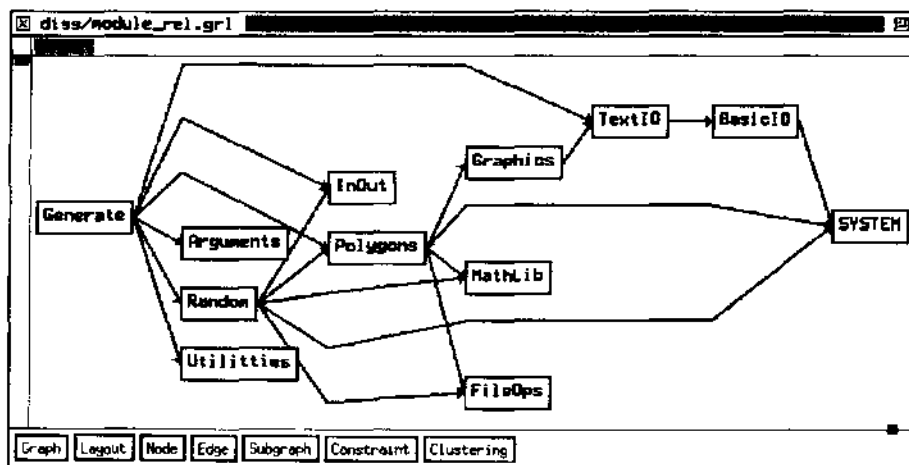


Figure 1.4: Graph editor depicting the import/export relations between modules

a graph-based tool to present information to the user, and, in particular, the benefits of automatic graph layout.

At the other extreme, the information being displayed may be changing rapidly. Consider the graph editor shown in figure 1.4 depicting the import/export relationship among modules of a program [Luc90]. The graph in this case is automatically generated from the source code of a Modula program and thus the frequency of changes is on the order of minutes rather than years. Conceivably, the user could access the source code through the editor and changes would be reflected in the graph's representation immediately.

This dissertation presents the design of an *extendible graph editor* which is a graph editor that can easily be adapted to a wide variety of applications. Changes made by the user will not only be reflected in the graphical representation of the graph, but also in the application itself. The next section presents the motivation and goals of this work and points out the shortcomings of existing graph editors. The subsequent section presents an overview of the main research contributions of the thesis. This chapter closes with an overview of the organization of the rest of this presentation.

1.1 Motivation and Goals

Graph editors have been developed for numerous applications [BNT86, WP86, CL88, BCL90, Bru88, RDLK90]. When using a graph editor for a particular application, application-specific actions are associated with the editing of the graph. For example, consider a graph editor for project management which