Frances Newbery Paulisch

The Design of an Extendible Graph Editor

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

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

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

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

List of Figures

List of Tables

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 1.1[1]). 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)

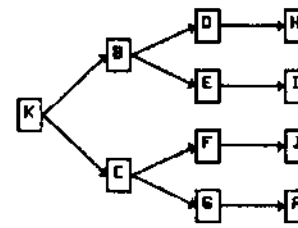

Figure 1.1: Textual and pictorial representations of a graph

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

[1]Unless 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

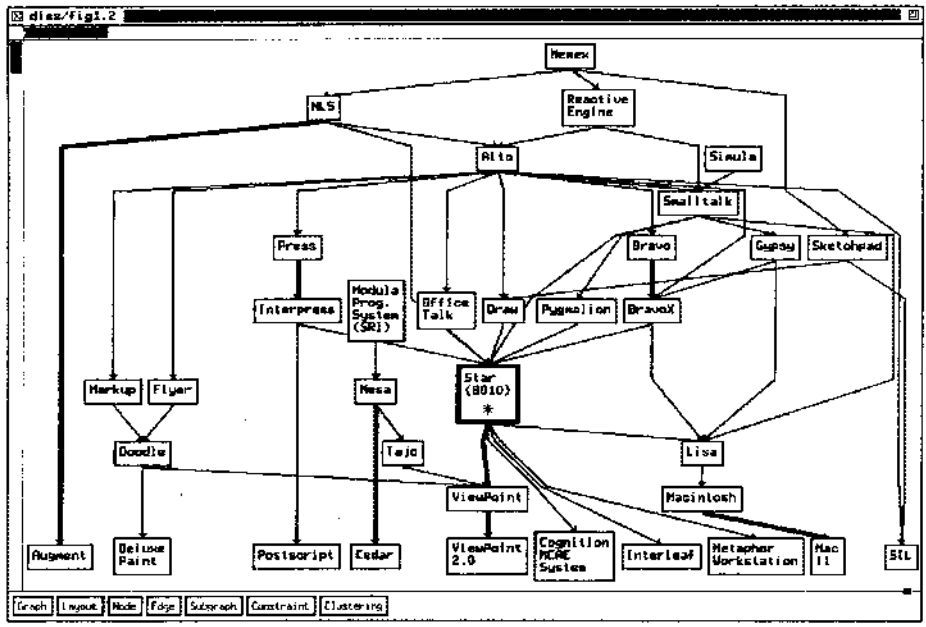

Figure 1.2: Development of the Xerox Star (EDGE graph editor)

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. Juding 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

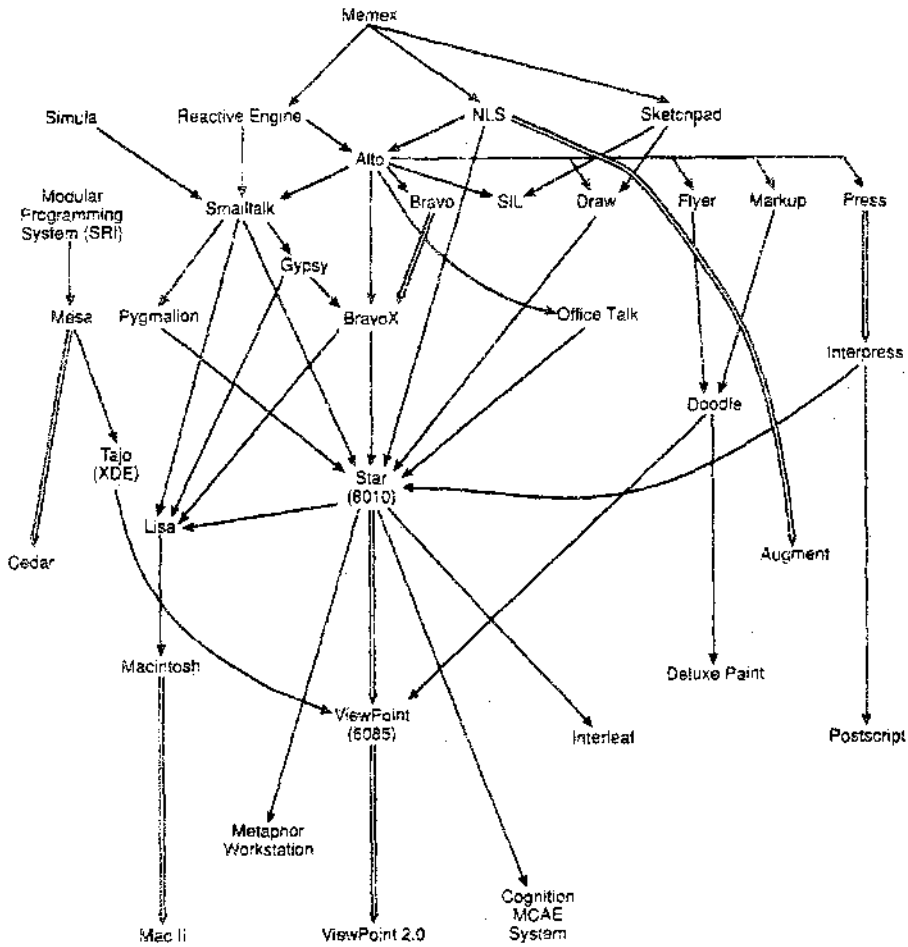

Figure 1.3: Development of the Xerox Star (from [JRV89])

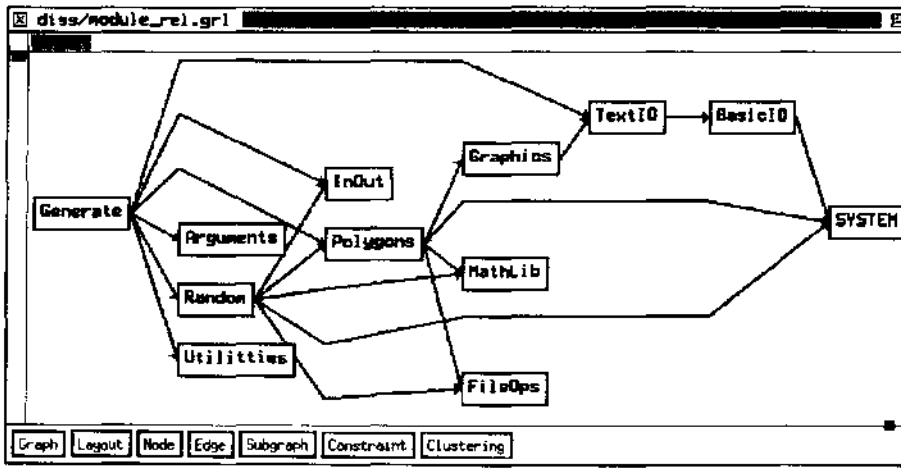

Figure 1.4: Graph editor depicting the import/export relations between modules

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