Automated Tool Support for a Large Scale Diagramming Tool

Aaron J. Quigley

Dept. of Computer Science & Software Engineering,
University of Newcastle, Callaghan NSW,
aquigley@cs.newcastle.edu.au

Abstract

GRACE is a large-scale diagramming tool constructed with the aid of automated tools from the Design Maintenance System (DMS). Diagramming tools as a component of meta-case tools are common in the *development* of new software engineering projects. The design and development of **GRACE** addresses the development of a large-scale diagramming tool for the *software maintenance* area. The nature of legacy systems, due to their domain specific nature and scale, imposes certain constraints on the production of new tools. GRACE and its supporting tools are described in this context.

Keywords: Visualization, Maintenance, Visual Language, and Layout

1. Introduction

Numerous software engineering diagramming schemes are available which allow software developers to model the design of a system and/or data stores. In recent years some of these schemes have been incorporated into CASE tools or more specifically META-CASE tools which aid *software development* in *new* software engineering projects.

However, there has been a general lack of attention paid to the development of advanced CASE tools for software maintenance. Tools which do address the maintenance problem, often force the user to learn different visual notations which may not be as powerful, flexible, expressive or concise as the diagramming scheme of choice in the pre-existing software environment. The salient difference between the primary development process and the maintenance of legacy systems is that legacy systems already exist and must be considered when developing any new tools [6,8,14]. The very existence of the legacy system imposes certain constraints on the production of new CASE tools.

Another aspect of the problem, for large code base systems, is the inability of most visual layout tools to adequately deal with the scale of the maintenance problem. This paper will discuss the design and development of tools which can support arbitrary diagramming schemes. The specification of the visual lexemes of the diagramming schema and the creation, layout and editing of large-scale software engineering diagrams will be addressed here.

An important aspect of producing any visual layout tool is to avoid poor information presentation. CASE tools are already hindered by bureaucratic debris in a cramped computer display. When graphical presenting information, "if a picture isn't worth a thousand words, the hell with it" [2]

2. Design Maintenance System

DMS is a software engineering toolkit for transforming and modifying domain-specific software specifications. DMS is entirely implemented in a parallel programming language PARLANCE developed by Semantic Designs Inc. It provides a parsing infrastructure, an AST builder, a transformation engine, pretty-printers and other basic tools for automatic source code modification based on program semantics. DMS deals with domain definitions, part of which comprise explicit syntax and transformations. For domains that have an associated graphical language, DMS uses visual domain editors, which aid domain engineers in specifying a description of the domain.

Underlying DMS technology is a domain independent data store called the "Hypergraph". The design of the Hypergraph aims at addressing two major problems for code modification: the scale of code base, which can range from thousands to millions of lines of code, and the computational complexity involved in any large modification.

Another key facet of DMS technology is the use of code generation in the production of other software engineering tools, either for use by DMS developers or external developers. Specifically, this paper presents work which was designed around and built using such code generating tools [1].

3. Graphical Lexeme Editor (GLE)

Visual diagramming languages such as UML, Entity Relationship diagrams, State Charts and Data Flow Diagrams are used to specify the design and aid in the development of software engineering projects. These languages are specific to software development; however, there are numerous other domain specific visual languages (DSVL) for the specification of designs such as circuit diagrams, SADT and PERT. These domain specific languages allow for more concise and easier to maintain visual descriptions, along with being easier to reason about due to the more specific nature of the language [10,12].

These DSVLs all contain visual lexicons. This visual lexicon is a dictionary, or the vocabulary, of that language and the minimal lexical unit of each language is a lexeme; examples are shown in Figure 1.

Figure 1: A Variety of Visual Lexemes

Specifying the visual make-up and visual constraints for each lexeme in a given language is the purpose of the Graphical Lexeme Editor (GLE). This tool, constructed as a part of DMS allows domain engineers to interactively specify, using some basic drawing primitives, the visual representation, visual usage constraints and connectivity of a lexeme. Once a lexeme is fully specified then it is added to the lexicon for this graphical language which in turn is a component of the domain description.

GLE is visually similar to many drawing packages but it provides more than just graphical editing. Each graphical primitive has a set of associated properties, which the user can change after the primitive has been placed. These include position, colors, filling, line thickness and a partial ordering of the graphical primitives, if one exists.

These operations mean that a variety of visual shapes can be created, which allows most of the lexemes of a visual language to be specified. An example of some UML lexemes being specified in GLE is shown in figure 2.

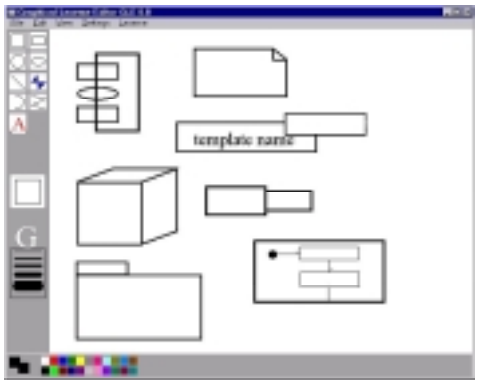

Figure 2: UML Lexemes edited in GLE

These lexemes have more than just a visual layout associated with them. Once the lexeme is visually specified, then certain usage, or layout constraints, can be applied. These constraints control, for example whether a lexeme can be scaled when drawn. Similarly, some lexemes can be rotated, usually in any rotation or just fixed set of rotations. An example circuit diagram shown in figure 3, was created using lexemes designed by GLE, This consists of the visual representations of AND, OR, EXOR and NOR gates. Conventional circuit layouts only use these lexemes in a 0^o, 90^o, 180^o, or 270^o position; if the designer of the visual lexicon wants to model this in the lexeme specification, then the rotation constraint must be applied. The lines which connect lexemes together are also lexemes, and as such must be specified and added to the visual lexicon for the language being specified. These lines also have usage constraints

Finally, for lexeme specification, some connectivity rules must be applied. Ports are logical items that can be associated with a section of a graphical primitive [13]. In certain visual languages there are some basic rules for what is allowed when connecting, e.g. in circuit logic the output gates cannot be directly wired together.

4. GRAph Creator Editor (GRACE)

The graph creator is part of the DMS toolkit. This editor provides facilities for the creation and editing of visual descriptions based on a specific visual lexicon. Using these, domain and application engineers can specify

information, in a domain specific graphical language. GRACE also comes with an API, which allows this visual information to be accessed by associated tools. These tools may mearly be interested in the logical information (i.e. syntax checker or logical search engines) but other tools may be concerned with the presentation of that information (i.e. layout engines or animation engines)

The primary principal is to maximize the information conveyed in the presentation [2]. Having layout engines access the presentation of the information either as a pre-processing step or interactively, helps avoid the visual soup some CASE tools sometimes suffer from.

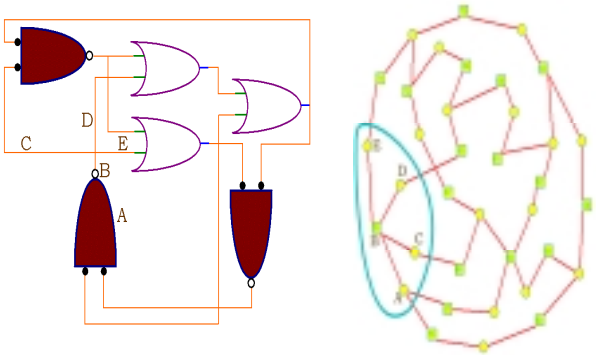

Figure 3: A circuit diagram and a *representation* of the underlying graph.

Using GRACE to layout a lexeme based visualization produces an underlying graph containing all the information which describes the visualization. This graph is in the hypergraph repository which allows other tools in DMS to interact with the graph and to alter it in a consistent manner. As a result of using this format, domain specific tools, for example a visual syntax checker based on the domain description, can be accessed from GRACE. This allows a consistent and powerful visualization tool to be built for any domain.

An example of a layout produced using lexemes specified in GLE is shown in figure 3. The gates and arcs are lexemes from GLE and are shown as circles in the abstract representation and the ports of those lexemes are shown as squares. However, the abstract graph contains no information about layout of the visualization, this information comes from the domain description. With the aid of layout algorithms, GRACE can produce information layouts specific to the domain in question.

GRACE is for domain engineers who may use different DSVLs. Although the visual lexicons may differ, and certain VLs do not facilitate associated tools interacting with them, there is a common set of operations that GRACE needs to provide to all domian engineers.

Due to the nature of software maintenance, GRACE provides generic visual navigation techniques for large amounts of information, along with standard editing facilities such as cut, copy, and paste. Visual palettes for both the lexemes and the basic syntax components are also provided to allow organizations to customize sets of visual macros that are used throughout the organization's code base.

Visual languages have an associated syntax, which describes how the lexemes can be composed to form 'visual sentences'. If the domain engineer has generated a syntax checker for this VL, then it can be accessed from GRACE. This allows the entire visualization, or a sub section, to be interactively checked for syntactic correctness. The syntax of a VL contains connectivity rules, which are visually specified in GLE. GRACE can interactively check, report, and disallow, connections between lexemes, which invalidate rules of a specified VL.

4. Implementation

GLE provided a platform for much of the initial research into GRACE. Issues on the data repository and storage mechanisms along with future tool interoperability were faced in implementing GLE. Due to the nature of the scale of the visualization problem involved both GLE and the initial work on GRACEe were implemented in PARLANCE a SD parallel language.

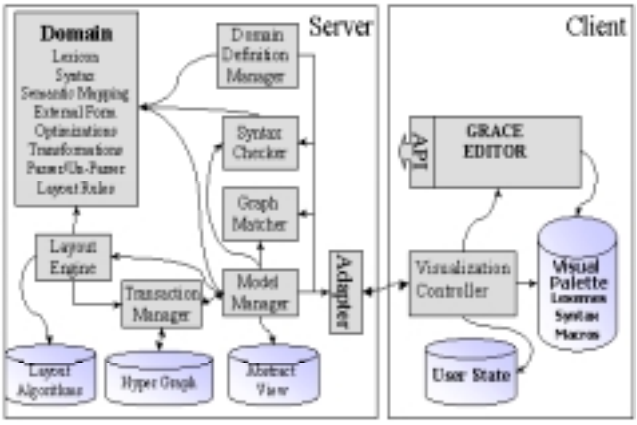

Figure 4: Architecture of GRACE.

GRACE is designed to handle the visualization of a large amount of data. It uses a parallel programming language along with space decomposition algorithms, as shown in figure 4, for the selection, editing and layout tasks. Using an SMP parallel processing architecture allows for GRACE to handle large visualization in a reasonable time.

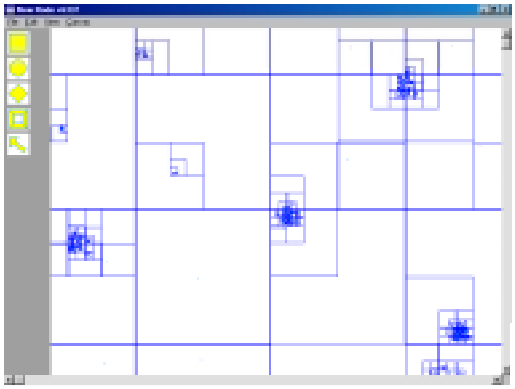

Figure 4: Large Space Decomposition.

There are numerous automatic layout algorithms available for information presentation such as Sugiyama layouts, orthogonal layout and force directed layouts [3,4,9]. The computational complexity of these algorithms usually prohibits their use in dealing with very large information layouts. In an attempt to address this problem, an implementation of the force directed layout was specified PARLANCE, which attacked the problem by fully utilizing the parallel-processing platform available. Subsequent work has focussed on using space decomposition techniques [11] to reduce the complexity of the layout problem along with using the parallel processing to speed up the layout.

5. Future Work

Further work in this area will be focussing on the problems associated with large-scale information presentation. Specifically, the nature of large scale graph layout [4, 11] clustering, and visualization of multiple levels of abstraction [15].

6. Acknowledgements

The main body of this work was undertaken when the author was a research intern at Semantic Designs Inc. in Austin Texas during 1998. The ongoing nature of this work relates to the authors Ph.D. study at the University of Newcastle, in Australia

References

[1] *Automatically Generating Visual Syntax-Directed Editors*, Computing Practices ACM Communications of the ACM Volume 33 No 3

[2] Edward R. Tufte, *Visual Explanations*, Graphics Press 1997.

[3] T. Fruchterman and E. Reingold, *"Graph Drawing by Force-Directed Placement"*, Software Practice and Experience vol, no. 11, pp1129-1164, 1991

[4] Giuseppe Di Battista, Peter Eades et al, *"Graph Drawing, algorithms for visualization of graphs*", Prentice Hall 1999.

[5] Sougata Mukherjea and James D. Foley, "*Requirements and Architecture of an Information Visualization Tool*", Database Issues in Data Visualization, IEEE Visualization '95 Workshop pp 57- 75

[6] Bruce W. Weide and Wayne D. Heym, "*Reverse Engineering of Legacy Code is Intractable*", Technical Report, Indiana University Southeast, 1994

[7] Charles Rich and Richard C. Waters, "*The Programmers Apprentice Project: A research Overview*", IEEE Expert Special Issue 1997

[8] M. Hutchins and K. Gallagher, "*Improving Visual Impact Analysis*", International Conference on Software Maintenance, 16 November 1998

[9] Karl-Friedrich Bohringer and Frances Newberry Paulisch, "*Using Constraints to Achieve Stability in Automatic Graph Layout Algorithms*", ACM SIGCHI Human Factor in Computing Systems, Seattle April 1990.

[10] J. Rekers and A. Schurr, "*Defining and Parsing Visual Languages with Layered Graph Grammars*", Technical Report Leiden University 1996.

[11] Guy Blellcoh and Girija Narlikar,"*A Practical Comparison of N-Body Algorithms*", Technical Report, Wright Laboratory 1991

[12] Wayne Citrin, Jeffrey D. McWhirter, "*Diagram Entry Mechanisms in Graphical Environments*", ACM SIGCHI 1995.

[13] Isabel F. Cruz, "*Expressing Constraints for Data Display Specification: A Visual Approach*", First Workshop on Principles and Practice of Constraint Programming, April 1993

[14] M. Storey, K.Wong and H.A. Muller, "*How do Program Understanding Tools affect How Programmers Understand Programs*", WCRE 97

[15] Survey Paper: Visualization Research Group, University of Durham 1996