NOTE TO USERS

Extending Two Drawing Frameworks to Create LaTeX Picture Environments

JIE XIAO

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

MARCH 2005

Your file Votre référence
ISBN: 0-494-04459-4
Our file Notre référence
ISBN: 0-494-04459-4

Abstract

Extending Two Drawing Frameworks to Create LaTeX Picture Environments

Jie Xiao

This thesis presents the detailed review of two graphical drawing frameworks, Graphviz and Drawlets, and the design and implementation to customize the two frameworks into WYSIWYG tools supporting LaTeX picture environment commands. With these tools, graphs can be drawn and manipulated on a GUI, and then saved into a TeX file directly. Both tools support the basic drawing commands, which allow drawing graph or connected graphs be composed of elements such as lines, vector, ellipses, rectangle, and text. Graphviz LaTeX Tool's layout algorithms produce either directed or undirected graphs in a well organized formats, whereas Drawlets LaTeX Tool imposes the constraints to the connecting lines, and still leaves the flexibility to allow drawing any kind of graphs. Graphviz tool provides Menu and Dialog Box driven GUI, and the GUI of Drawlets tool is Toolbar driven.

Acknowledgements

I would like to thank my supervisor, Professor Peter Grogono, for the most interesting project, which stimulates me to complete my study. Also, I would like to express my warmest gratitude for his patience and invaluable guidance.

I thank my husband for his patience, understanding, and support during this long study, and my son for the times not being able to stay with him.

Special thanks for my parents, their encouragement accompanied me all the time during all my studies in all the programs.

Table of Contents

List of Figures

List of Tables

Acronyms and Definitions

2D	Two-dimensional
ASCII	American Standard Code for Information Interchange
AWK	Aho, Weinberger, Kernighan (Pattern Scanning Language)
AWT	Abstract Window Toolkit
CAD	Computer-Aided Design
CGD	Clan-based Graph Drawing
DOT	A common attributed graph data language for Graphviz graph manipulation tools
dot	Graphviz layout tool for directed graph
DVI	Device-Independent (File Name Extension)
EPS	Encapsulated PostScript
ER	Entity Relationship
ERML	A language based on XML to convert an ER-diagram
FIG	A vector drawing format which can be used with programs such as xfig to produce simple figures for documents
GasTeX	A set of LaTeX macros which allows easily drawing graphs
GDL	Graph Description Language
GEF	Graph Editing Framework
GIF	Graphic Interchange Format (File Name Extension)
GML	Graph Modeling Language
GUI	Graphical User Interface
GXL	Graph eXchange Language (based on XML)
HTML	Hyper Text Markup Language (and File Name Extension)
ILE	Incremental Layout Engine
JFC	Java Foundation Class
JPIC-XML	An XML format specially tailored for jPicEdt
JVM	Java Virtual Machine
LaTeX	A macro package which sits on top of TeX and provides all the structuring facilities to help with writing large documents

lefty	A programmable editor for technical pictures in Graphviz Toolkit
Lefty	A script language
MiKTeX	An up-to-date TeX implementation for the Windows operating system
ND	Node
OS	Operating System
PDF	Portable Document Format
PGML	Precision Graphics Markup Language
PNG	Portable Network Graphics (graphic file standard/extension)
PS	PostScript
RMD	File extension for Microsoft RegMaid Document
SQL	Structured Query Language
SVG	Scalable Vector Graphics (W3C)
TeX	An extremely powerful macro-based text formatter written by Donald E. Knuth
UML	Unified Modeling Language
VCG	Visualization of Compiler Graphs
VGJ	Visualizing Graphs with Java
W3C	World Wide Web Consortium
WWW	World Wide Web
WYSIWYG	What You See Is What You Get
XML	eXtensible Markup Language

Chapter 1. Introduction

LaTeX [1] is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents, but it can be used for almost any form of publishing. Like documents prepared by other word processors, especially WYSIWYG editors such as MSWord, the complex graphs and images of data structures such as binary trees, SQL diagrams, flow charts etc. can also be included in the LaTeX documents. Documents created with LaTeX on a properly configured system can be exported to PostScript, PDF, and HTML.

There are two ways of providing LaTeX documents with pictures:

- by directly programming the pictures within LaTeX,
- by importing pictures drawn with some sort of a vector based program.

1.1. Drawing Pictures in LaTeX

The picture environment allows programming pictures directly in LaTeX. However, there are some constraints; especially, the slopes of line segments as well as the diameter of circles are restricted to a narrow choice of values. Besides these, the LaTeX picture commands are flexible enough to allow one to build many kind of drawings, including text, circles, rectangles, lines and vectors, and bezier splines.

The LaTeX picture environment uses a Cartesian coordinate system to specify the size of a picture and the position and dimensions of objects located in the picture. The positions and dimensions of these objects are specified in the picture commands by their

coordinates. A coordinate specifies a length in multiples of the unit length. A position is a pair of coordinates, such as (2.4,-5), specifying the point with x-coordinate = 2.4 and y-coordinate = -5. The basic syntax for the LaTeX picture environment is

```
\begin{picture}(width,height)(x-offset,y-offset)
 ...
  picture commands
 ...
 \end{picture}
```

Everything that appears in a picture is drawn by either `\put` or `\multiput` command. Detailed descriptions of the picture environment commands are presented in Lamport [1] or Hypertext Help with LaTeX [2] which are extracted and listed in Appendix B in this thesis. Oswald [3] provides many interesting examples by using these commands.

1.2. Importing Pictures in LaTeX

There are several packages allowing the import of pictures in LaTeX. The package most broadly recommended is graphicx. While LaTeX can import virtually any graphics format, Encapsulated PostScript (EPS) is the easiest graphics format to import into LaTeX. For example, EPS files are inserted by specifying `\usepackage{graphicx}` in the document's preamble and then using the command `\includegraphics{file.eps}`. Optionally, the graphic can be scaled to a specified height or width, and rotated with the angle option. See the excellent document Reckdahl [4] for an exhaustive treatment of including graphics in LaTeX documents.

The EPS files are used by both LaTeX and the DVI-to-PS converter. The EPS graphics are not included in the DVI file. Since the EPS files must be present when the DVI file is

converted to PostScript, the EPS files must accompany DVI files whenever they are moved.

To include a picture in LaTeX document, the picture should be drawn in other software first, and then converted to EPS format, finally imported into the LaTeX file using the above mentioned commands. There are many graphic visualization and drawing tools available for achieving this, such as aiSee [5], Graphviz [6], Ipe [7], VGJ [8] and jfig [9], etc., as discussed in Chapter 2.

1.3. Motivations

It is obvious that the LaTeX picture environment has some disadvantages due to the limited commands for few shapes and bezier curves, etc. and the constraints for some shapes. Thus, they are not very expressive, and since there are no WYSIWYG tools, it is tedious to draw the picture by hand, such as getting all the coordinates, etc. correct. Also, it is hard to modify an existing picture.

However, there are still reasons for using the LaTeX picture environment. For example, it provides compact source file, there are no additional EPS graphics files to be dragged along. The single TeX file is self-contained and light-weight, thus the documents produced are small with respect to bytes. The example figures used by the current work show a ratio in the range of 10.7 to 32.4 in term of file sizes between EPS and TeX. In the worst case, they are 23 KB vs. 710 bytes. The figures created in LaTeX picture environment are fully standard, whereas the embedding syntax is non-standard. The

significance of "fully standard" is that, although LaTeX itself is standardized, it leaves the way in which graphics and image files are embedded to the implementer. Consequently a LaTeX file that includes EPS or other file types is not portable. However, a LaTeX file that just uses the picture environment is portable.

To facilitate using LaTeX picture environment for drawing figures in LaTeX documents, it is desirable to have WYSIWYG tools available to avoid doing it manually. The existing tools, such as JasTex [10] and jPicEdt [11] as discussed in Chapter 2, are either not for standard LaTeX, or do not support any layout algorithms to layout the drawing.

The Current work aims to apply the existing drawing framework, and customize it to be a WYSIWYG tool, which can generate LaTeX codes from drawing directly. The purpose of using the existing framework, on one side, is to save development times; also it is expected to adopt more advanced features in the framework. For example, the framework chosen should have the ability to layout the graph automatically in order to produce better looking and organized drawing. A windows version LaTeX-like typesetting system, MiKTeX [12], is chosen for the testing purpose of the current work.

1.4. Contribution of the Thesis

The contributions of this thesis fall into two areas: analysis and implementations. The analytical contribution is the thorough review through classifying the existing graphical drawing tools into different categories, and the selection of the frameworks which will be extended and customized into the graphical tools supporting LaTeX picture environment

commands. The other contribution is two implementations of WYSIWYG tools for LaTeX picture environment based on different type of framework, and the comparison of the two tools. The Graphviz LaTeX tool is built on the Bell Labs tool Graphviz. It provides a graphical user interface for drawing and saving the graphs into LaTeX picture format directly. Sections 3.2 through 3.4 provide a detailed description of the design and implementation of the GraphViz Latex tool and some examples of its use. A similar tool, the Drawlets LaTeX tool is built by extending from Drawlets framework with detailed descriptions given in sections 4.2 to 4.4 for its design, implementation, and examples of its usage. Both Graphviz and Drawlets LaTeX tools support the basic drawing elements such as lines, vector, ellipses, and rectangle, etc., and provide different ways to adjust the positions and connections between graph nodes. They both have a GUI where the drawing is created, manipulated, and saved into TeX file.

1.5. Organization of the Thesis

There are six chapters and several appendixes in the thesis discussing and comparing the graphical drawing tools which can generate LaTeX commands. **Chapter 1** discusses the ways to include pictures in LaTeX document and the motivations of the current work. **Chapter 2** reviews and classifies the existing graphical drawing tools and presents some of their extended applications. **Chapter 3** and **Chapter 4** first reviews the framework used, that is Graphviz and Drawlets, and then presents the detailed design and implementation of the LaTeX Tool on customizing the framework, respectively. **Chapter 5** compares the two LaTeX tools implemented regarding their features, layout algorithms or constraints supported, and the GUI. **Chapter 6** concludes the current work and

provides the suggestions for future works. Installation guide, LaTeX picture environment commands, GDL and DOT examples, Tex output from the tools, and source codes for both Graphviz and Drawlets tools are listed in the corresponding **Appendixes**.

Chapter 2. Review of Graphical Drawing Tools

Technical drawing is a very important application of computers. In many professional disciplines, engineering drawing or blue-prints are used to detail the design and exchange information. On the other hand, technical reports and technical publications require illustrations and other graphics formats to express complicated ideas and display data. CAD, Drawing & Painting Tools [13] lists many excellent software packages. This section will focus mainly on the discussion of drawing tools and frameworks related to the current work, i.e. tools running under Windows and creating figures for LaTeX documents.

Available drawing tools can be mainly classified into the following four categories:

- Graph visualization tools with layout algorithm
- Graph drawing tools creating figures for LaTeX documents
- Graph drawing tools generating LaTeX commands
- Extensible drawing frameworks for customized drawing tools

The following sections will discuss each category in detail with given examples and some applications based on the framework will also be presented.

2.1. Graph Visualization Tools

Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. Automatic graph drawing has many important applications in software engineering, database and web design, networking, and in visual interfaces for many other domains. When working with graphs and trees, visualization usually

provides for much better and faster understanding. A simple textual rendering of graphs is often confusing or even unintelligible. A graph visualization tool does not support an interactive drawing interface (GUI), but runs as a command line program or starts from the Start->Programs pop up menu. It reads a textual graph specification, prepares or layouts the graph for visualization. aiSee [5], *dot* and *neato* etc. of Graphviz [6], VCG Tool [14] and VGJ [8, 15] fall into this category. Both Graphviz and VGJ are also drawing tools with supported drawing interface and will be discussed in the later sections. This section describes aiSee and Graphviz's layout tools for their usage and output format, the graph specification language and the algorithms used to layout the graph.

2.1.1. aiSee

aiSee [5] is a graph visualization tool developed based on the VCG Tool (Visualization of Compiler Graphs). It runs under Windows, Unix, and Mac OS X (with X11). aiSee automatically calculates a customizable layout of graphs specified in Graph Description Language (GDL), which is then displayed, and can be printed or interactively explored. aiSee was developed to visualize the internal data structures typically found in compilers, but now it is widely used in many different areas, such as circuit diagrams in Hardware Design, control flow graphs in Software Development, and relationship diagrams in Database Management. aiSee can export graphs to various formats, including SVG, PNG, HTML (image maps), and colored PostScript (on multiple pages for large graphs) which can be included in LaTeX document [5].

GDL is an ASCII text representation of a graph. It describes a graph in terms of nodes, edges, subgraphs and attributes. A subgraph is described as a normal graph except that it is specified inside another graph, meaning graph specifications can be nested. There is always only one top-level graph. aiSee provides special operations for subgraphs such as folding to a summary node, boxing, clustering, or wrapping. Graphs, nodes and edges may have attributes that specify details of their appearance on the screen such as colors, sizes, shapes etc [16].

The same example used by [17] showing the dependencies of different shell programs is taken, and the graph is specified in GDL with a combination of aiSee features as listed in Appendix C-1. The visualization of the graph is shown in Figure 2.1-1.

aiSee offers 14 basic hierarchical layout algorithms for directed graphs, including a specialized version for trees; and a force-directed layout schemes (see below) for undirected graphs. All these partitioning procedures make use of heuristic techniques. The graph layout can be changed either by updating the source file directly or through the application's layout dialog window. The layout can be extensively influenced by edge, node and graph attributes or by different layout algorithms. Sections *The Effect of the Layout Algorithms* and *Tree Layout* in [16] show the same graph visualized by different hierarchical layout algorithms.

Hierarchical layout algorithms consist of the following four phases: Rank Assignment, Crossing Reduction, Coordinate Calculation, and Edge Bending.

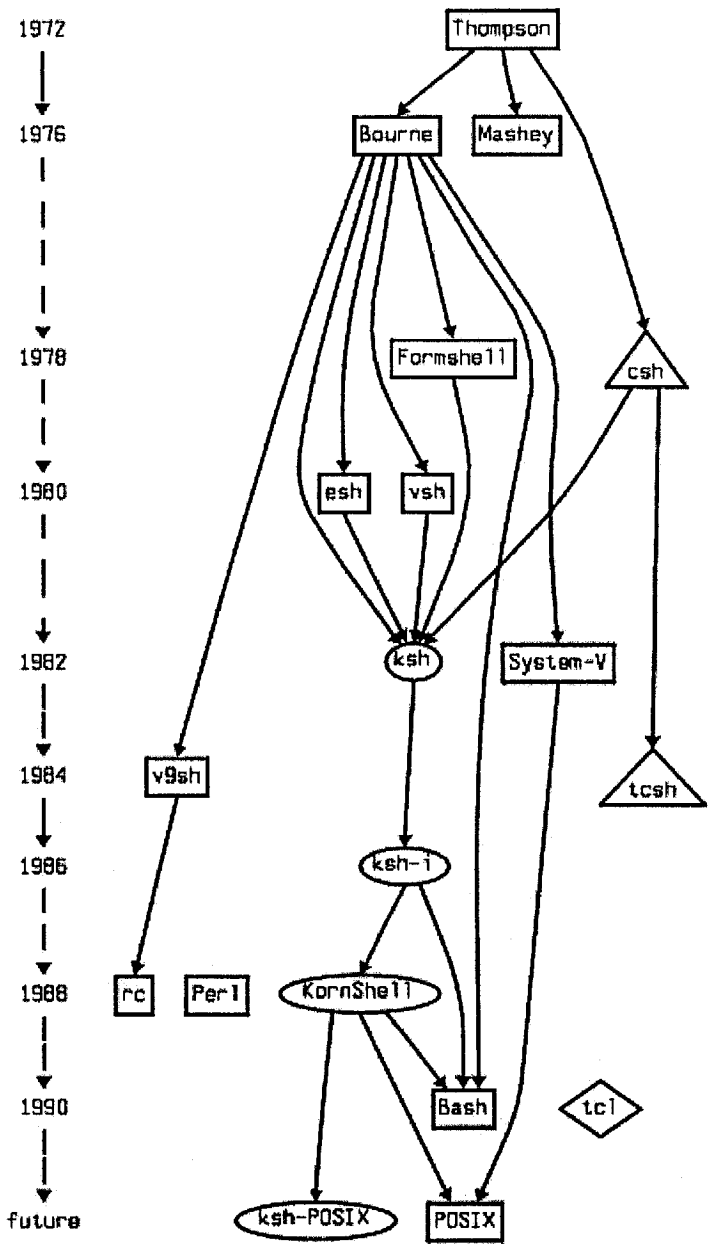


Figure 2.1-1 Visualization of Development of Shells by aiSee

aiSee combines the following four ideas in its force-directed layout algorithm: Spring forces, Magnetic forces, Gravitational forces, Simulated annealing. And the force-directed placement algorithm consists of four phases: Initialization phase, First iteration phase, Optional second iteration phase, and Final improvement phase.

The overall layout phases of aiSee include Parsing, Grouping of Nodes and Edges (Folding Phase), Hierarchical/Force-Directed Layout Phase, and finally Drawing phase.

2.1.2. Graphviz

Graphviz [6] is open source graph visualization software for Unix or Windows. It has several main graph layout programs which take descriptions of graphs in a simple text language, DOT, and make diagrams in several useful formats such as images and SVG for web pages, Postscript for inclusion in LaTeX or other documents, or display the graph specification in an interactive graph browser.

DOT is the graph specification language used by Graphviz. It describes three kinds of objects: graphs, nodes, and edges. The main (outermost) graph can be directed (digraph) or undirected graph. Within a main graph, a subgraph defines a subset of nodes and edges. An abstract grammar defining the DOT language can be found in Appendix A in [18]. Appendix C-2 lists the graph specified in DOT for the same shell programs used in section 2.1.1. As the graph is directed, it can be processed by Graphviz *dot* and the graph visualization is shown in Figure 2.1-2.

Unlike aiSee, Graphviz uses different graph layout tools to support different layout algorithms which are summarized as below [19, 20]; the detailed layout algorithm used by *dot* and *neato* will be discussed in section 5.2.1.

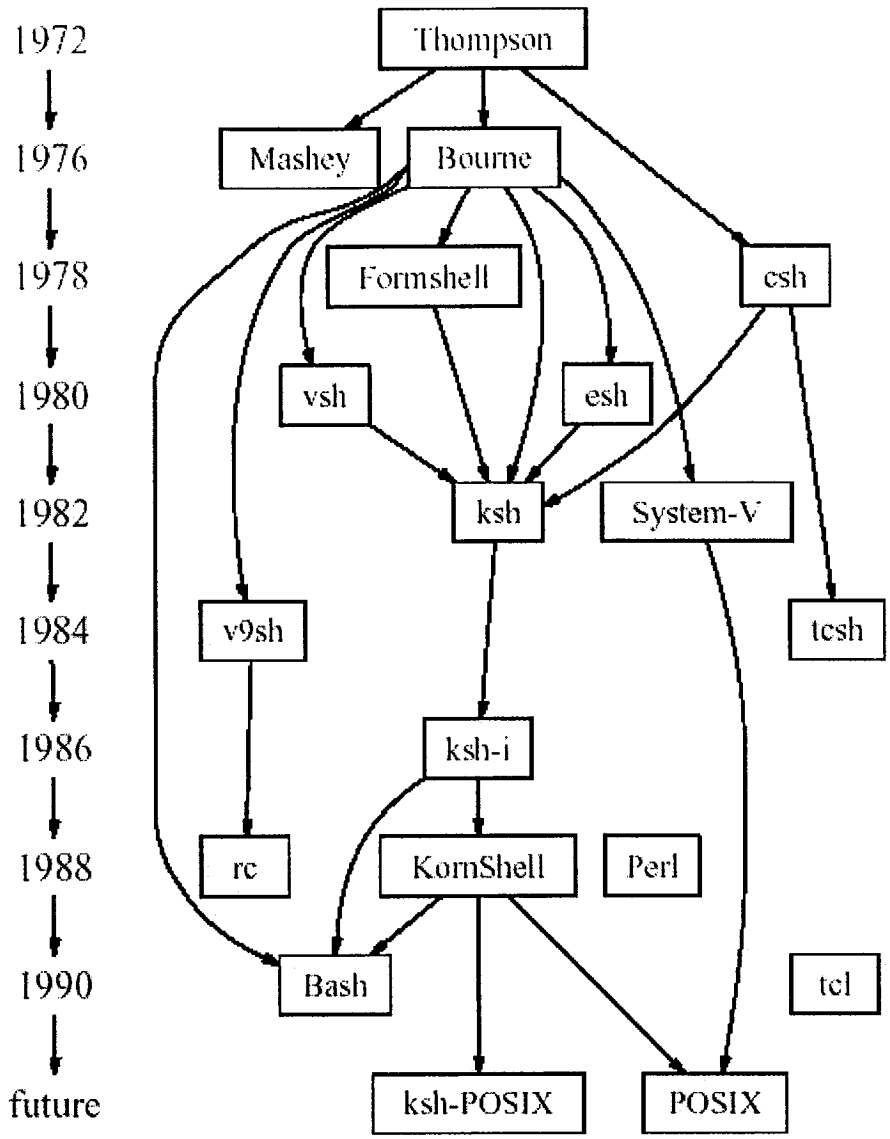


Figure 2.1-2 Visualization of Development of Shells by Graphviz *dot*

- *dot* makes hierarchical or layered drawings of directed graphs. A Sugiyama-style hierarchical layout algorithm is implemented [17, 21]. The layout algorithm aims on edges in the same direction (top to bottom, or left to right) and then attempts to avoid edge crossings and reduce edge length.

- *neato* makes "spring model" ("symmetric") layout of undirected graphs. It uses the Kamada-Kawai algorithm [22], which is equivalent to statistical multi-dimensional scaling [23, 24].
- *fdp* implements the Fruchterman-Reingold algorithm [42] for "symmetric" layout. This layout is similar to *neato*, but including a multigrid solver that handles larger graphs and clustered undirected graphs.
- *twopi* implements the radial layout as described by Graham Wills [43].
- *circo* implements the circular layout combining aspects of the work of Six and Tollis [27, 28], Kauffman and Wiese [29]. It is suitable for certain diagrams of multiple cyclic structures.

The above mentioned tools are all static layout tools, which take the input file and layout the graph statically in one shot. Static diagrams are not completely satisfactory because in many situations, the displayed graphs can change. Three common scenarios are: manual editing, browsing large graphs, and visualizing dynamic graphs. Graphviz also supports incremental layout heuristic for visualizing dynamic graphs, e.g., DynaDAG for incremental layout of directed acyclic graphs drawn as hierarchies [30, 31].

2.2. Tools Creating Figures for LaTeX Documents

There are many tools which can create figures in postscript for LaTeX document, such as daVinci, Dia, Graph Layout Toolkit, GraphEd, Graphlet, Graphviz, Ipe, Ivtools (Unidraw), VGJ and Xfig etc. (web links are available in [13]). This section discusses Ipe and VGJ, some of the remaining tools will be covered in section 2.1.4. For each tool, its

usage, the output format and the algorithms used to layout the graph if applicable are discussed.

2.2.1. Ipe

Ipe [7] is an extendible drawing editor for Unix, Windows, and Mac OS X systems. It allows creating figures in PDF format for inclusion into LaTeX documents as well as stand-alone PDF documents, for instance to print transparencies or for on-line presentations. Ipe drawings combine postscript data with LaTeX source code, that are both stored in the same file. Users can create Ipelets (Ipe plug-ins) to add their own functions to Ipe, extending it with editing functions or geometric objects for their desired tasks.

Ipe allows preparing and editing drawings with a variety of basic geometry primitives like lines, splines, polygons, circles etc. through a graphical interface, but without providing any layout algorithm. Ipe allows adding text to the drawings, and unlike most other drawing programs, Ipe treats these text object as LaTeX source code in the picture environment. When the file is saved, it converts the LaTeX command to PDF or Postscript, thus produces one pure Postscript/PDF file. The text is displayed as it will appear in the figure. This makes it easy to enter mathematical expressions, and to reuse the LaTeX-macros of the main document. For example, if LaTeX commands that are defined in additional LaTeX packages are needed, they can be included by `(\usepackage)` in the LaTeX preamble, which can be set in *Document properties* in the *Edit* menu. After a text object is created or edited, the Ipe screen will display the

beginning of the LaTeX command. Selecting *Run Latex* from the *File* menu converts the text object at once and creates the PDF/Postscript representation, along with the TeX file containing the LaTeX source code of the text object.

Except for PDF/Postscript, Ipe supports one other file format, which is a pure XML implementation. Files stored in this format can be parsed with any XML-aware application, and you can create XML files for Ipe from your own applications. Ipe ignores all attributes it doesn't understand, and they will be lost if the document is saved again from Ipe. The Ipe Manual [32] describes this format in details.

2.2.2. VGJ

VGJ [8], Visualizing Graphs with Java, is a tool for graph drawing and graph layout under both Unix and Windows systems. Graphs can be input into VGJ in two ways: loading a textual description in GML (Graph Modeling Language) or drawing the graph using the graph editor. It supports three layout algorithms to layout the drawing automatically in an organized and aesthetically pleasing way. The graph drawn can be saved either in GML or in postscript format.

GML [33] is a textual representation language for graphs. It attempts to standardize graph input format to allow exchanging graphs between different programs. GML supports attaching arbitrary information to graphs, nodes and edges, and is therefore able to emulate almost every other format. The GML text can be edited directly in VGJ by choosing *Edit Text Representation (GML)* from the *Edit* menu. A simple graph

containing a box and an oval with connecting edge between them is drawn and saved in GML format which is listed in Appendix C-3.

VGJ graph editor contains the typical features of tools that support graph drawing, for example, it allows drawing and editing simple graphs with rectangle, oval, circle nodes and the connecting edges, zooming in and out the drawing in the graph window. Besides, it provides a two-dimensional analysis of the graph for viewing graphs in different angles, whereas the typical approach uses the single dimension, level, to arrange the nodes.

VGJ offers several layout algorithms as shown in Figure 2.2-1. It has about 48 classes organized in 5 packages: algorithm, examplealg, graph, gui and util. Except for the example random algorithm (exampleAlg2) which resides in examplealg package, the rest algorithm classes in Figure 2.2-1 and the classes related to these algorithms are all grouped in the algorithm package. These algorithms are discussed in VGJ User Manual [34]. The spring algorithm is actually a "force-directed' layout algorithm for drawing undirected graphs in the sense to reduce edge crossings and use uniform length segments to connect the edges. The tree algorithm presents the drawing to a minimum horizontal spacing and width in a tree style. It also does adjust for different sized nodes. For the directed graphs, the CGD (clan-based graph drawing) algorithm is used which has the similar four phases as used by aiSee. The Biconnect algorithm tests the graph for biconnectivity, or makes it biconnected by adding edges.

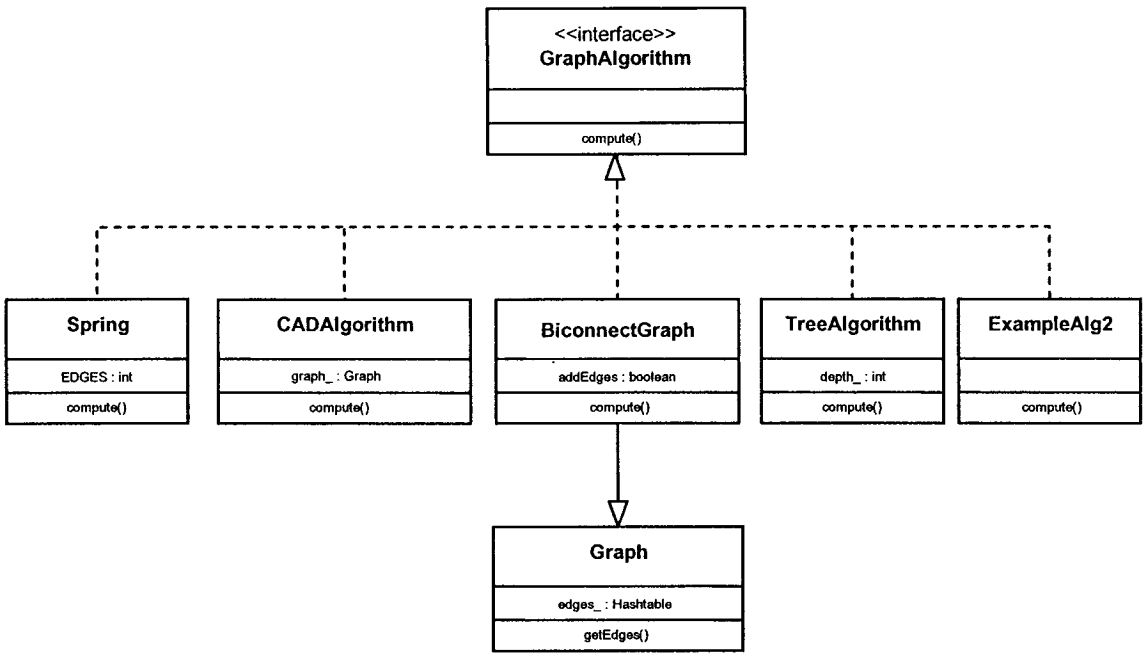


Figure 2.2-1 Class Hierarchical Diagram for VGJ Layout Algorithms

2.3. Tools Generating LaTeX Commands

As discussed in section 1.1.1, the LaTeX picture environment is a special environment for inserting simple drawing commands into a LaTeX file. This environment was obviously not intended to replace more sophisticated graphical formats such as Postscript or PDF, but was merely intended for easily incorporating small and simple drawings with the given primitives. Examples for LaTeX commands can be found in [3]. However, even preparing a simple drawing takes a lot of effort to calculate and put the graph elements in the proper positions. Some works have been done on developing the graphical drawing editor which saves the drawing into LaTeX commands directly. JasTex [10] and jPicEdt [11] are the two examples.

2.3.1. JasTex

JasTex is a Graphical tool written in Java for drawing and editing graphs in the format of GasTeX (Graphs and Automata Simplified in TeX) [35]. GasTex is a set of LaTeX macros which allows easily drawing graphs, automata, nets, diagrams, etc. in the LaTeX picture environment. A drawing with GasTex basically consists of nodes and edges. JasTex supports basic nodes like rectangle, circle, oval and edges like lines and Bezier curves without setting any constraint and layout algorithms on the drawing. Figures 2.3-1 and 2.3-2 shows the sample drawing and the corresponding LaTeX (GasTex) commands generated in the two frames. It is obvious that all the commands related to nodes and edges, such as `\drawrect` and `\drawline` are GasTex specific commands and they require having the GasTex macros installed by `\usepackage{gastex}`. This restricts the JasTex to limited usage.

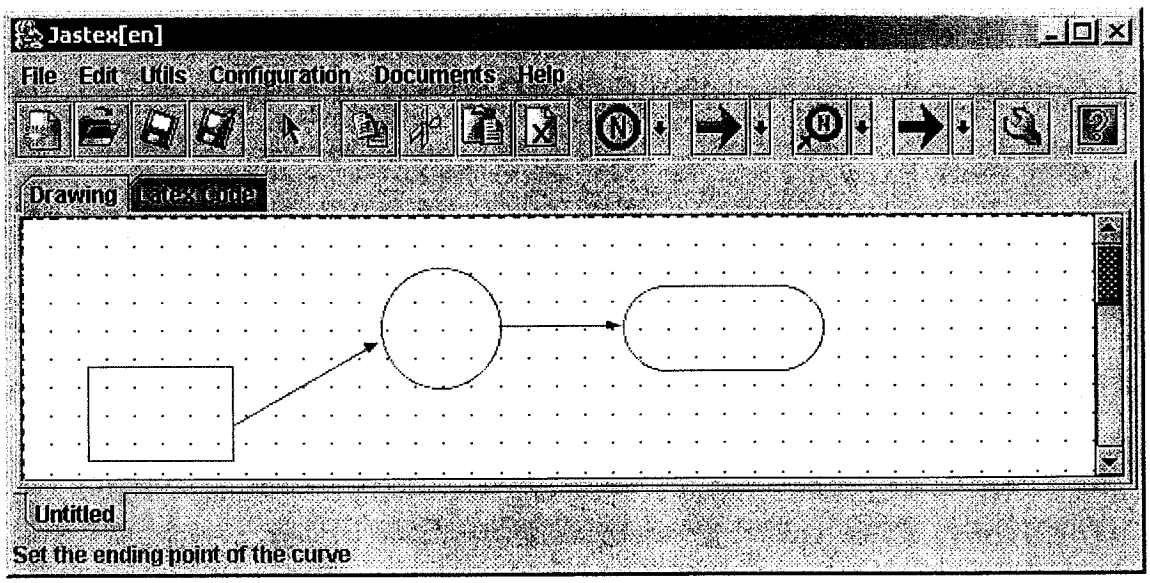


Figure 2.3-1 JasTex Drawing Frame

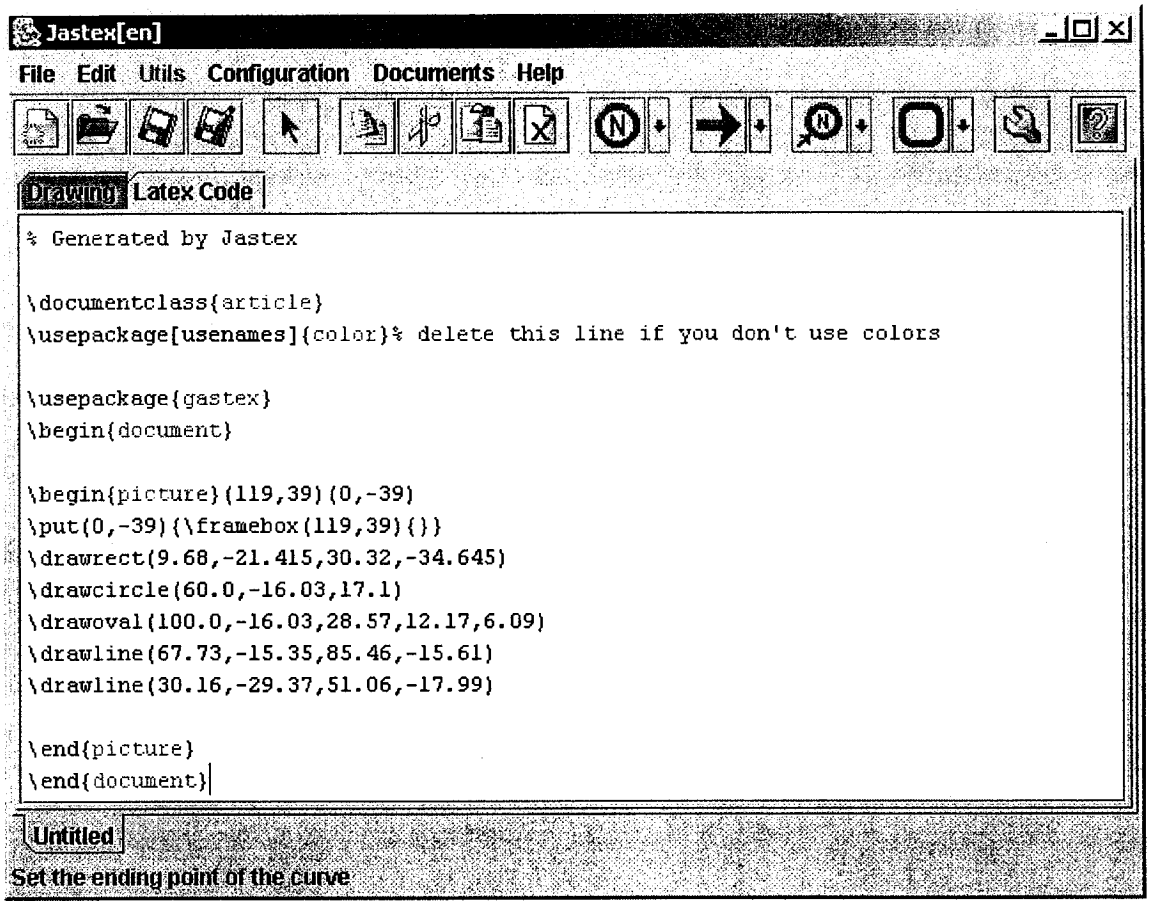


Figure 2.3-2 JasTex LaTeX Code Frame

2.3.2. jPicEdt

jPicEdt [11] is a picture editor for LaTeX written in Java. Thus, it can run on any platform where a JVM (Java Virtual Machine) is installed. It supports drawing every graphical element allowed by the LaTeX picture environment commands, such as lines, arrows, circles, boxes; and the emulated elements, for example lines of any slope, circles of any size by using the `\multiput` command. It supports also drawing most of the objects allowed by the *epic/eepic* packages, e.g., dashed lines filled ellipse; and every object allowed by the *pstricks.sty* package including filling with colors etc. jPicEdt can

generate the LaTeX, eepic and PsTricks commands from the drawing directly and save them in the corresponding format, which can be then loaded again for editing.

jPicEdt saves and loads the drawings in a special way. It writes the drawing content twice into one file, first in a special format called JPIC-XML, which is an XML format specially tailored for jPicEdt; then in the LaTeX format (LaTeX picture environment, eepic or PsTricks). The purpose to adding those XML codes is to allow jPicEdt to reload the drawing without losing even the tiniest piece of information. They are commented out so that the file can be properly compiled by LaTeX. jPicEdt can open files in the three LaTeX formats prepared by other software by delegating to an embedded LaTeX parser.

jPicEdit always set unit length to 1 mm at the beginning of the saved file which results in big drawings, especially when loading a file with default unit length setting (i.e., one big point, about 1/72 in). By default, the maximum LaTeX circle diameter in jPicEdt is set to 14 mm, which corresponds to 40 big points in the default unit length setting. If loading a LaTeX file containing a circle with 15 points in diameter (e.g. the bigger circle in Figure 2.3-3) and saving it again, this circle will be treated as 15 mm and jPicEdt will produce 4 `\put` commands and 44 `\multiput` commands for it as listed in Appendix D, whereas its original command is just one line `\put(27.00,72.00){\circle{15.00}}`.

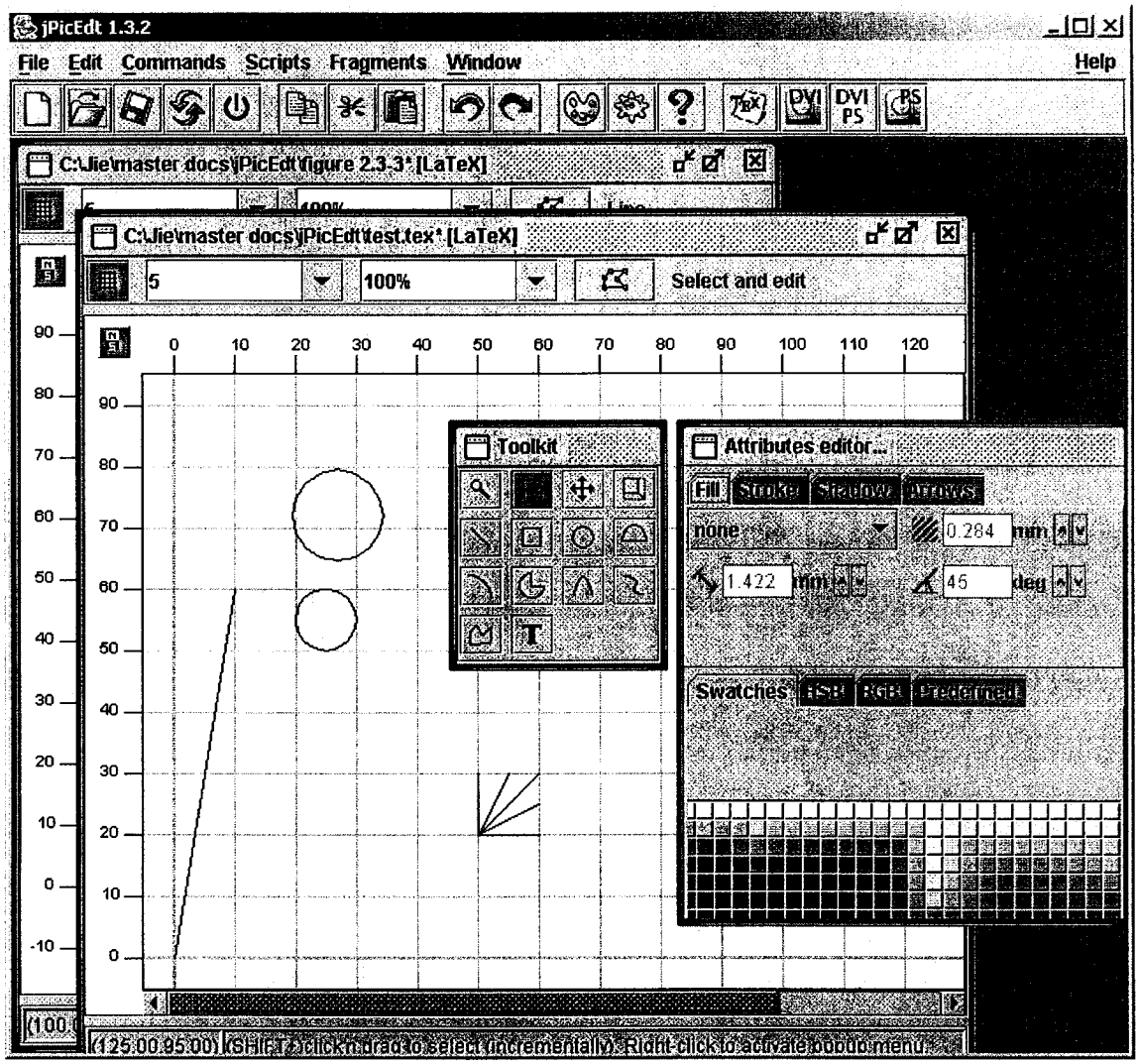


Figure 2.3-3 jPicEdt 1.3.2 User Interface and Sample Drawing

Using the mouse, all the elements can be started and ended at full or half grid axes as shown in Figure 2.3-3. For example, in one unit square (10x10), only 5 lines can be drawn, whereas there are 25 possible slope values allowed in LaTeX picture environment [3]. However, the precise control over the object location can be done through the properties panel where the user can change object shape using numerical entries.

jPicEdt seems a powerful graph editor for drawing LaTeX picture. Although the restrictions on mouse drawing help to some extent, the lack of supporting any layout algorithm makes it rely on the user to position the drawing element manually, which is tedious and time consuming.

2.4. Extendible Drawing Frameworks

The extensible drawing framework includes either drawing tools mentioned in previous sections which focus on defining a powerful, extendible graph editor, such as Graph Layout Toolkit, GraphEd, Graphlet, Graphviz, Ipe, Ivtools, jfig and Xfig, or frameworks which are reusable design for building graphical applications in some particular domains, like Drawlets [36] and GEF [37]. A good graph drawing framework would save a lot of work avoiding building the application from scratch and would make better research products. This section will discuss the selected frameworks with their usages, the algorithms supported to layout the graph if applicable, the extensibility and the way to customize them.

2.4.1. Drawlets

Drawlets is a Java version of HotDraw [38], thus a platform independent framework for structured drawing editors. It is an extensible framework that can be used to add drawing functionality to a host application or to build editors for specialized two-dimensional drawings such as schematic diagrams, blueprints, music, or program designs. The elements of these drawings can have constraints between them which make them connected, and they can react to commands by the user [36, 39].

Customization can be done in the following ways. Patterns for Drawlets [36] presents the detailed descriptions on how to implement a figure, add new tools, and make new handles.

- Add graphical aspects to your application with and without tool palettes
- Add new tools and palettes for the specific purpose
- Add new types of figures and associate them to the underlying domain objects

A case study [40] used HotDraw to create a planer simulator which divided the application development into the five tasks:

- Created a default empty application using the framework which can be customized to draw an EllipseFigure inside the application;
- Addressed a Figure's edit and selection behavior;
- Explored the capabilities of Tools in greater detail and a new Tool was created to add planets into the simulator, also appropriate Handles were developed to allow editing of a planets mass;
- Involved creating a customized Tool which focused on creating constraints between Figures;
- Addressed animation in the simulation (not applicable for Drawlets for now).

2.4.2. GEF

GEF [37], the Graph Editing Framework, is a free Java library that supports the construction of graph editing applications under different operating systems. GEF

supports a wide range of basic graphics primitives, such as rectangle, circle, line, polygon and text etc. for a representation of connected graphs, and user interactions for moving, resizing and reshaping the graph elements. It also supports several novel interactions such as the broom alignment tool and selection-action-buttons. GEF supports grid interface that helps people make nice looking diagrams, which in turn can be viewed in multiple coordinated windows. GEF can save and load the diagram in XML-based file formats following the Precision Graphics Markup Language (PGML) standard [41], which is a 2D scalable graphics language designed to meet both the simple vector graphics needs of casual users and the precision needs of graphics artists.

GEF is designed with only a few basic concepts that are implemented in clusters of classes. They are Editor, Fig, Layer, Mode, Cmd, and Selection which can be briefly described as follows.

- Editor is the most central class of the system; it is just a shell that dispatches control to various other objects that do the actual drawing and processing of input events.
- Fig is drawable object that can be shown and manipulated in the Editor.
- Layer serves both to group Figs into transparent overlays and to manage redraw order and find the objects under a given mouse point.
- Mode is mode of operation for the Editor.
- Cmd is an abstract class that define a doIt() method that performs some action in the Editor.
- Selection is object used by the Editor when the user selects a Fig.

GEF can be used to build a new application starting from scratch, or it can be used to add a graphical interface to an existing application. GEF is designed in the way that new features can be added without modifying the framework itself. The easiest way of using GEF is to add attributes to subclasses of GEF classes, but the classes already exist in the existing class hierarchy, then delegation must be used, instead of subclassing. These application specific classes have to be defined in the new project package to extend GEF and apply GEF to the project.

2.4.3. Graphviz

Except for the layout tools discussed in section 2.1.1.2, Graphviz supports also graphical tools to run these layout programs, such as Dotty, TclDot, Grappa, Montage and TclDG [42]. These tools run stand-alone, but can also be extended to create interfaces to external databases and systems. This usually involves writing Lefty, Tcl or Tcl/Tk scripts and Java code to customize the graph editor's behavior and to program it communicate with external files or programs.

Dotty [43, 44] is a customizable graph editor constructed as two co-operating processes. Its main components are a programmable graphics editor (Lefty) and the graph layout tools (*dot* and *neato*). The Lefty program that implements Dotty starts up *dot* as a separate process to compute layouts. When a new layout is required, Lefty sends the graph to *dot*, the layout tool computes the layout and outputs the graph (in the graph language notation) with coordinate and size information as graph attributes. Lefty then

redraws the picture using the new layout. Dotty can be customized to handle graphs for specific applications, or programmed to communicate with other processes, i.e., as a front end for applications that use graphs. Dotty can be controlled either through a WYSIWYG view, or through a program view. Thus, Customizations can be done online, by typing Lefty expressions at the program view, or by creating a Lefty script in a file and loading it in. Examples and some general guidelines for customizing Dotty are given in [44].

TclDot is a customizable graphical interface written in Tcl. It is essentially Dotty in Tcl. TclDot binds Libgraph functions to Tcl commands, and has internal functions for rendering graphs in Tk canvases. Although *dot* is linked in as a library, communication is still handled through graph string attributes. The base graph editor is customized by loading scripts and extensions.

TclDG [45] is a practical user interface toolkit that incorporates incremental graph layouts. It was written specially for prototyping user interfaces to distributed network management systems. TclDG is a direct descendent of TclDot and built on the next-generation of the graph libraries. The conceptual framework of TclDG combines graphs, views, and canvasses. It provides a Tcl script-level interface to instances of graphs, supported by Libgraph, and to instances of views, supported by the various Incremental Layout Engines (ILEs). The engine interface is not tied specifically to TclDG, so the set of layout engines can be reused in other environments or is extendible. The current selection includes:

- DynaDag - Hierarchical directed graphs, splined edges [46];

- OrthoGrid - Incremental manhattan layout [47];
- GeoGraph - User-defined node placement and straight edges. (For graphs in which nodes are manually placed, or in which nodes have intrinsic location properties that dictate placement).

TclDG can be customized through the following ways:

- Scriptable and customizable in Tcl;
- Extendible via other Tcl/Tk packages;
- Extendible via other programs using Libgraph's file language.

Grappa [46] is a graph package written in Java. It was created to provide graphical interaction with graphs through web browsers, integrated with other Java packages. It has a good number of useful features built into it, but is also extensible. Grappa has classes for graph representation and communication with layout services (via a Dot language parser). Grappa is extendible in the sense that application dependent classes can be defined naturally as sub-classes of Grappa object classes. Although Grappa does not share any code with the rest of Graphviz system, it adheres to Libgraph's model and file language, including Dot's conventions for graphical symbols and attributes, and so fits in well with the other graph processing tools.

2.4.4. jfig

jfig [9] is a 2D-graphics or diagram editor written in pure Java based upon xfig, a popular graphics editor for the X11 window system. The user interface of jfig aims to be

compatible with xfig. jfig runs well on Unix and Linux, but the most common reason to use jfig is to have an xfig-compatible graphics editor on a Windows system. jfig offers all standard drawing primitives including bezier splines, each with attributes like colors, line and fill styles, etc. jfig uses the Java2D graphics library for high-quality rendering.

From a software point of view, jfig is not just the jfig editor but a class library for the construction of graphics editors, which consists of four Java packages. The first package, `jfig.objects`, implements the graphical objects defined by the FIG format (e.g. rectangle, polyline, text) and also some utility classes like the FIG file parser and writer. The second package, `jfig.canvas`, provides the drawing canvas including rubberbanding, buffering, and efficient redraw. The third package, `jfig.gui` includes custom user interface classes. A fourth package, `jfig.commands`, collects the command objects that implement the individual drawing operations in the editor. jfig includes three main applications built from the jfig class libraries:

- jfig graphics editor is the main application built from the jfig class library. It is a 2D graphics editor based upon xfig. It tries to be as compatible with xfig as possible with Java. It reads and writes FIG 2.1, 3.1, and 3.2 formats, supports all object types and attributes from xfig 3.2.3, and provides all common edit operations.
- FIG viewer applet allows embedding FIG files directly into HTML pages.
- The presentation viewer and applet allow building simple PowerPoint-like presentations (slide-shows) from multiple FIG files. It uses a simple text-format index file which specifies the names and ordering of the external FIG files.

By design, the jfig editor is modular and can be extended easily. This allows creating user-defined graphics editors based on jfig by providing additional editor commands and even new object types.

2.5. Applications of Drawing Frameworks

There are many applications developed from different drawing frameworks. Some related to the frameworks described in section 2.4 are discussed in this section.

ERDraw [48] is an XML-based ER-diagram drawing and translation tool developed with Drawlets framework. It supports drawing ER-diagrams visually and translating them to relational database schema with integrity constraints automatically. ERDraw retrieves and outputs ER-diagram in ERML file, a language based on XML to convert an ER-diagram into ERML format and vice versa. It supports also the verification of the validity of an ER-diagram and ensures that only a well-formed one is exported in ERML file.

There are several products that use GEF, such as ArgoUML, Bandara and Disy Cadenza [37]. ArgoUML [49], as an example, is a modeling tool that helps on design using UML. ArgoUML uses GEF to edit UML diagrams. The following diagram types are supported: Class diagram, Statechart diagram, Activity diagram, Use Case diagram, Collaboration diagram, Deployment diagram and Sequence diagram.

Several Graphviz Dotty applications are described in [43]. They all fall into the class of customization that Dotty is programmed to act as a front-end for another process. ciao is

a graphical interface to the cia and cia++ program databases which are source code analysis tools for large programs written in C or C++ respectively. The graphical representations help to clarify relationships between program entities. The main customization to Dotty of ciao was the specification of a table that provides a mapping between node types (which correspond to cia entities) and operations appropriate for each type. The user interface functions were then modified so that when the user tries to bring up a menu over a graph node, the menu that appears contains exactly those operations that are appropriate. To generate complete graphs, such as the full function-to-function reference graph or the file-to-file inclusion graph, Dotty composes a UNIX command pipeline. The first part of the pipeline is the appropriate cia query. The second part of the pipeline is the cia tool that generates graphs from query output. Dotty then runs this command and collects the output graph. ciao consists of approximately 500 lines of Lefty code. No changes to cia had to be made in implementing ciao.

2.6. LaTeX Tool Implementation

The existing graphical tools generating LaTeX commands discussed in section 2.3 show the limitations either on the standard LaTeX picture environment or on the lack of supporting layout algorithms. The current work aims to develop a LaTeX tool solving these limitations within the extensible drawing framework.

As described in section 2.1.4, there exist two types of extensible drawing frameworks. One of each type will be selected for the current project to implement a graphical tool

that can generate LaTeX code directly from the drawn graph. The two implementations will be compared in chapter 5.

Graphviz shows its efficient layout algorithms for making very readable drawings of graphs and convenient graph drawing systems as described in the previous sections, especially Dotty which can be customized for many applications. One class of its customization is to handle graphs for specific applications. The LaTeX tool needs only to save the graph drawn in Dotty as LaTeX picture commands, whereas Dotty actually can write the graphs in attributed DOT format [18] whose objects have associated layout coordinates. Thus, customizing Dotty to a LaTeX tool seems feasible.

Drawlets was selected to perform case studies as a software evolution benchmark [50] and on test suite maintenance in system evolution [51, 52]. As discussed in section 2.4.1, the old version of Drawlets, HotDraw, was used in a case study to identify the problem of large scale reuse [40]. Although Drawlets doesn't support any specific layout algorithm, it provides constraints to make the drawing elements connected. Also, Drawlets's design pattern allows extending the layout algorithms feasible by adding an algorithm package as discussed in section 2.2.2.

Chapter 3. Graphviz LaTeX Tool

Graphviz LaTeX Tool is a customized graph editor extended from Graphviz's Dotty to generate LaTeX codes directly from the drawing graphs. Section 3.1 will briefly review Graphviz focused on Dotty and the remaining sections discuss our work in terms of the design, implementation, and sample drawings of Graphviz LaTeX Tool.

3.1. Graphviz Overview

Graphviz is a set of graph drawing tools for creating and editing technical pictures. The Graphviz toolkit contains base libraries for handling attributed graphs; a collection of graph layout algorithms; platform-dependent front ends; and a complement of file-stream graph processors. The main components are given in Table 3.1-1 [19, 42].

Sections 2.1.1.2 and 2.1.4.4 discussed layout tools and graphical tools respectively. Libraries and Graph Filters are out of the scope of the project and will not be discussed in the thesis, readers are referred to [42] for the detailed information. This section will focus on Dotty's system components and its design.

3.1.1. Dotty (*lefty*) System Components

Dotty is constructed as two co-operating processes, *dot* and *lefty*. *lefty* [31, 53] is a general-purpose programmable editor for technical pictures. This editor has no hardwired knowledge about specific picture layouts or editing operations. Each picture is described by a program that contains functions to draw the picture and functions to perform editing operations that are appropriate for the specific picture. Primitive user actions, like mouse

and keyboard events, are also bound to functions in this program. Besides the graphical view of the picture itself (WYSIWYG interface), the editor presents a textual view of the program that describes the picture. *lefty* can be used as a standalone editor, to prepare pictures for printing, but it can also communicate with other processes. Dotty is built on top of *lefty* by adding the user interface and graph editing operations as Lefty functions. It provides menu and mouse-driven graph operations. Graph layouts are made by *dot*, which runs as a separate process that communicates with *lefty* through pipes. The following sub-sections describe briefly each of Dotty (*lefty*) components.

Table 3.1-1 Graphviz Toolkit Components

Libraries	Libgraph	The base library for graph tools with attributed graph data structures and I/O
	Dynagraph	Incremental layout library
Layout Tools	dot	A tool for hierarchical layout of directed graphs
	neato	A tool for "spring" model layouts of undirected graphs
	fdp	A tool for "spring" layouts, similar to neato
	twopi	A tool for radial layout
	circo	A tool for circular layout
Graphical Tools	Dotty	A customizable graph editor written in Lefty
	Tcldot	A customizable graphical interface written in Tcl
	WebDot	A TclDot scripted WWW service for graphs in HTML documents.
	Grappa	A Java package for graphs with full Java graph data structures
	Montage	Generic ActiveX diagram container
	Tcldg/Dged	Tcl/Tk graph editor for incremental layout
Graph Filters	Gpr	Generic graph filter
	Sccmap	Decomposes graphs into strongly-connected components
	Colorize	Computes node colors from initial seeds
	Unflatten	Adjusts edges to improve aspect ratio of hierarchical layouts

3.1.1.1 Programming Language

lefty programs are written in a scripting language (also called Lefty) whose semantic level is about the same as conventional scripting languages (Visual Basic, Unix shell, Tcl/Tk). Lefty has string variables with automatic runtime conversion for arithmetic, associative arrays, hierarchical namespaces for organizing code and data, and functions with arguments and local variables. Its standard environment includes function libraries for working with files and processes, and a collection of common graphical widgets such as canvases with scroll bars, dialog boxes and file selection widgets. Lefty runs on a variety of popular window systems because it hides the host graphics and operating system. Lefty has a small library of C code specifically for supporting graph editors. A disadvantage of Lefty is that the language is non-standard, but most of its features, such as functions, scalars and associative arrays, hierarchical name spaces, windows, menus, and text-stream I/O are familiar to experienced programmers [43]. The specification of Lefty language can be found in [53].

3.1.1.2 Two Views

A picture in Dotty (*lefty*) is shown in two ways. One view is the usual "what you see is what you get view" (WYSIWYG). The other view is a textual view of the program that controls the picture. Users can perform operations on either view. The WYSIWYG view is more intuitive, while the program view is more functional.

The WYSIWYG view is the graphical representation of the picture which can consist of one or more widgets, such as drawing areas, buttons, lists, text areas, and scrollable widgets. Widgets can be manipulated using built-in functions. A global table called *widgets* lists all the widget entries. Each of these entries is a table that can be used to customize the behavior of the widget.

Dotty supports two types of WYSIWYG views: a normal view and a bird's-eye view. Each type of view can have its own way of displaying a graph and its own set of mappings from user actions to graph operations. Figure 3.1-1 is a finite state machine diagram loaded in Dotty's normal graphical view.

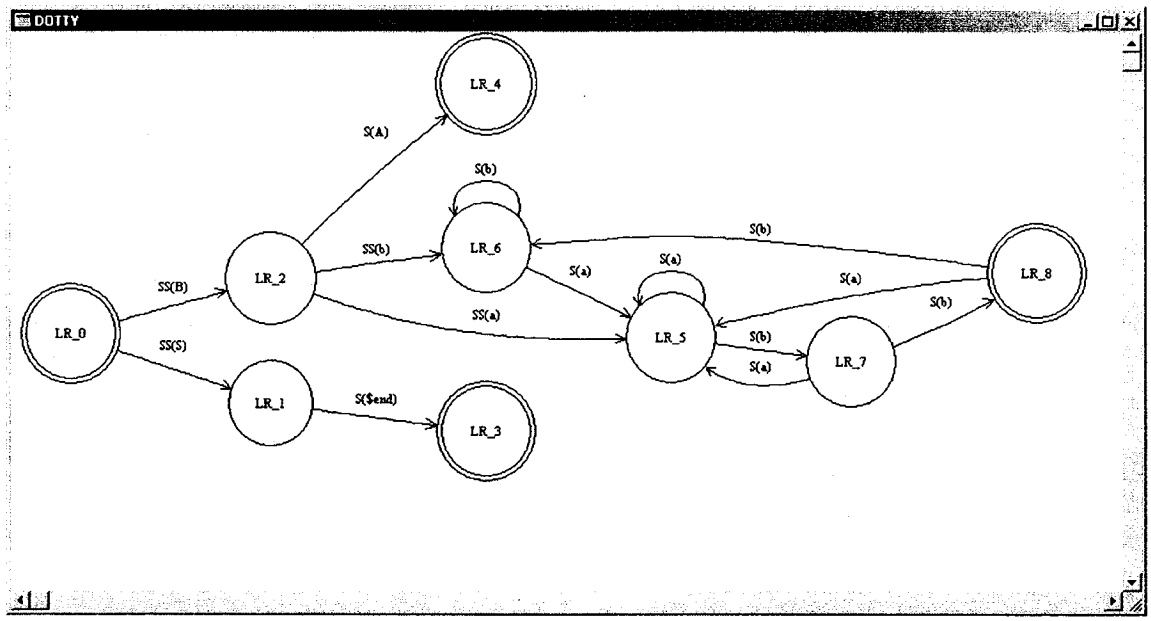


Figure 3.1-1 Sample Finite State Machine in Dotty WYSIWYG View

The program view is a textual representation of the picture state. It displays the name and value of each global object. The textual representation can be long, so the editor presents an abbreviated view by default: each name, value pair is displayed on a line. Clicking on

a line will expand the object. Figure 3.1-2 shows the program view with 'widgets 5' expanded.

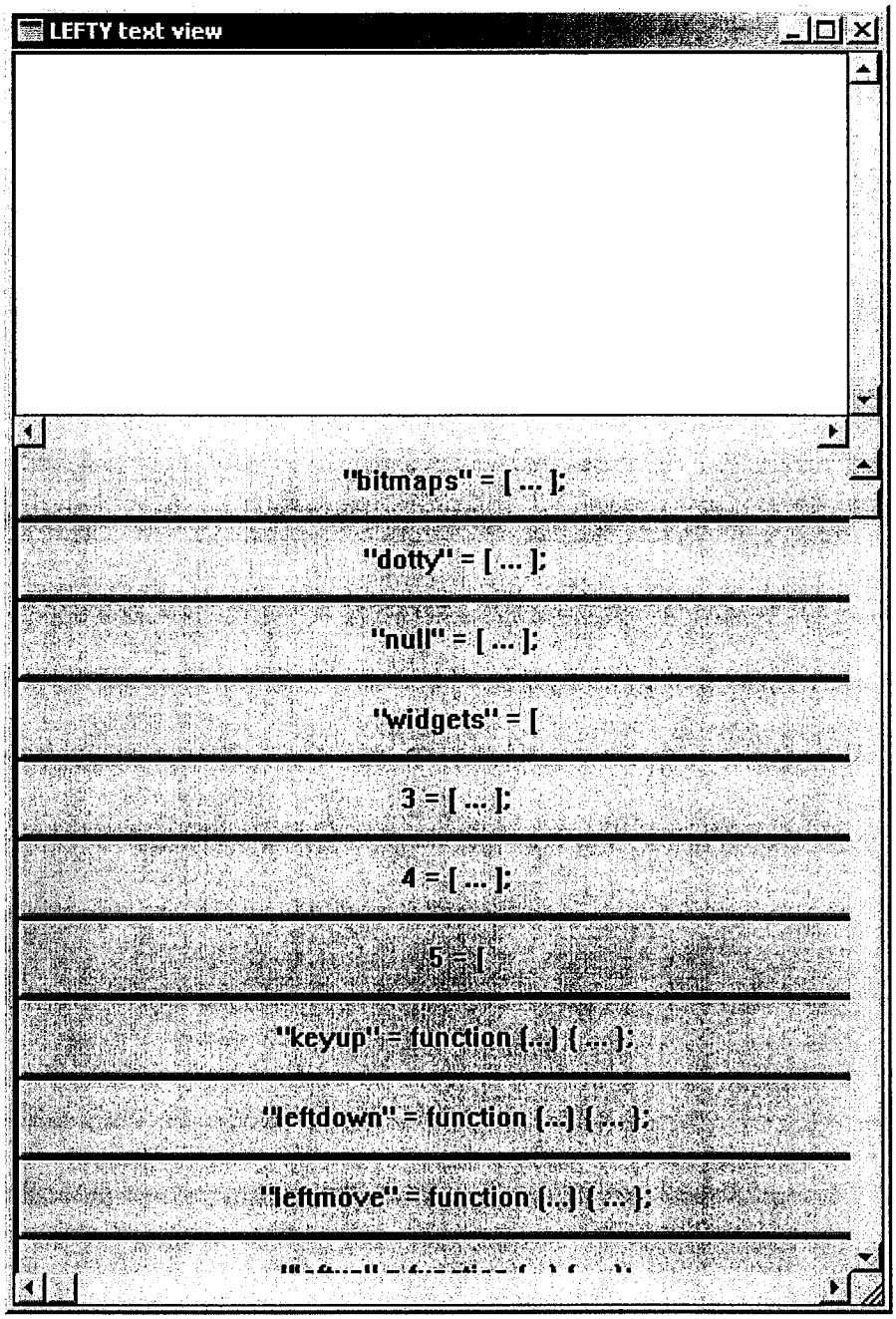


Figure 3.1-2 Expanded Dotty Program View

3.1.1.3 Inter-process Communication

lefty provides built-ins for communicating with other processes by exchanging ASCII strings. This allows *lefty* to communicate with many existing tools, without having to modify these tools at all. This capability can be used in several ways [53]:

- Purely for output
- For both input and output
- As an extension to the editor itself

Dotty implements a function that computes the layout in the second way. This function sends the graph to a *dot* process running in the background. The *dot* process computes the layout and sends the graph (with layout information inserted as graph attributes) back to the *lefty* process. *lefty* then updates the node and edge coordinates and redraws the graph on the screen.

3.1.1.4 Built-in Functions

Lefty built-ins can be used to perform window system and graphics operations and to access various system resources such as files. They can be classified into following categories, refer to [53] for the complete list:

- Widget Functions (`createwidget`, `setwidgetattr` ...)
- Graphics Functions (`clear`, `setpick`, `ask` ...)
- Bitmap Functions (`createbitmap`, `destroybitmap` ...)
- Input and Output Functions (`openio`, `writeline`, `writegraph` ...)
- Math Functions (`sqrt`, `random`, `toint` ...)

- Miscellaneous Functions (`echo`, `strlen`, `split` ...)

3.1.1.5 Dotty Functions

Dotty customizes *lefty* in Lefty language to handle graphs and their components. The program includes functions to insert and delete nodes and edges, as well as to draw these objects according to attributes such as color, shape, and style. Overall, the Dotty's Lefty program implements the following operations [43]:

- Create or destroy graphs.
- Create or destroy views of graphs (a graph may have several views).
- Load or save graphs to files (or sockets and pipes)
- Insert or delete nodes, edges, and subgraphs
- Pan or zoom within a view
- Search for a node by name
- Geometrically move a node (and have all its edges follow)
- Edit attributes of an object

3.1.2. Dotty Design

The user interface and graph editing operations of Dotty are written as Lefty programs that are contained in several Lefty files: *dotty.lefty*, *dotty_draw.lefty*, *dotty_edit.lefty*, *dotty_layout.lefty* and *dotty_ui.lefty*. These scripts define a set of functions and a set of data structures. All functions and variables are stored under a table called *dotty* as shown in Figure 3.1-2. The Dotty design interface is specified in [44] and briefly summarized as follows.

3.1.2.1 Design Interface

dotty defines the data structure used by Dotty as two main classes of objects, graphs and views. `protogt` is a table and contains the default prototype graph. One of its fields is `graph` which contains default values for graph, node, and edge attributes. `protogt` also contains all the functions that operate on the graph. `protovt` contains the default prototype view. Like `protogt`, this table contains both the data and the functions that manage a view. Two prototype views, `normal` and `birdseye` are defined.

dotty as a main entry loads the rest Lefty files. It initializes Dotty by `init` function, it contains functions operating on the graph such as `creategraph`, `layoutgraph`, `findnode` and `setattr` etc., and also functions managing views, such as `createview` and `destroyview`. It provides utilities functions like `openio`, and `printorsave` function to print out a graph or save it in a file.

dotty_draw defines the functions to draw the graph, such as `drawgraph`, `drawnode` and `setviewsize`. It contains shape functions, like `shapefunc.ellipse`. Adding new shape functions need to arrange for *dot* to recognize these shapes.

dotty_edit defines the functions to edit the graph by manipulating the graph structure, such as `mergegraph`, `removenode`, `cut` and `paste`.

dotty_layout defines functions to set communications with *dot* process and layout the graph like `grablserver` and `startlayout`.

dotty_ui defines user interface functions and the related data structures for both normal and birds-eye views. By default the left mouse button is bound to inserting or moving nodes (`leftdown`, `leftmove` and `leftup`), the middle button is bound to inserting edges between existing nodes (`middledown` and `middleup`), and the right button brings up a menu for selecting the operations to change node and edge attributes (`rightdown`). There are also node and edge specific menus defined.

3.1.2.2 Output in DOT Format

Dotty is stream-oriented program; it reads graphs, computes layouts, and writes the graphs either as layouts in a graphics language (PostScript, GIF, etc.) or as attributed graphs whose objects have associated layout coordinates in DOT format. Attributed DOT Format is given in Appendix C in [18] and is summarized as follow:

- Lower-left corner is set as origins
- Positions are represented as integers (X, Y) specified in points (1/72 of an inch) referring to the center of the object
- Lengths are given in inches
- `bb` specifies bounding box of drawing in integer points
- `lp` is graph or edge label position in integer points

- Edge is assigned a `pos` attribute as B-spline control points. Edge points are listed from top to bottom if there is no arrow; otherwise they are listed from arrow point with prefix '`e`' for forward arrow and '`s`' for backward arrow

3.2. Design of Graphviz LaTeX Tool

As discussed, Dotty is a graph editor and can write the drawing into attributed graph format. The purpose of our tool is to save the drawing graph into LaTeX picture format, thus the tool needs to keep all functions implemented in Dotty scripts (*dotty*.lefty* files) and to add some functions to transform the attributed format into LaTeX commands.

3.2.1. Customizing Dotty

Following the general guidelines given in [44], we customized the tool from Dotty by creating a new lefty script *latex.lefty*. This script loads *dotty.lefty*, defines new functions like *savelatexfile* to write the graph into LaTeX format, and override predefined functions. Unlike Dotty, it supports both *dot* and *neato* layout processes based on the user's choice. A batch file (*latex.bat*) is used to start up the program from *main* function.

3.2.1.1 Saving LaTeX File

The major function of the tool is *savelatexfile*. After the layout is done, it processes the attributed graph and saves it into LaTeX format by doing the followings:

- Open a file as *.*tex*
- Write out LaTeX Prefix by using default unitlength (1/72 of an inch)
- Process graph to get and write out `bb` (bounding box) value

- Iterate through graph nodes to get the positions and write out nodes of different shapes by different functions supporting Box, Circle, Ellipse, Triangle, Double Circle, Diamond, Parallelogram, Trapezium, and Msquare, and write out node label
- Iterate through edges, identify the arrow style (backward, forward and none arrow) and write out edge and edge label based on different rules
- Write out graph label
- Write out LaTeX Suffix
- Close the file

3.2.1.2 Customizing *do layout* Function

neato is compatible with *dot* to the extent of accepting the same input files and command line options and writing the drawing into the same attributed graph format. Thus, Dotty can switch between either of the two by changing the path name of the layout server. In *dotty.lefty* file, by default the layout tool is set to *dot* as `'lserver' = 'dot'`. The tool is designed to support both layout processes in the way that before performing *do layout*, it prompts a dialogue box for user to select the desired layout tool, *dot* or *neato*.

3.2.1.3 Overriding Predefined Functions

Some functions and default values defined by Dotty need to be overridden, such as the default table *protogt* and *init* function defined in *dotty.lefty*. The normal view name of the graphical interface should be changed to a more meaningful name like '`Latex`

`Generator`' in the *protovt* table. The new item `save latex file` should be added into `menus` table and `actions.general` function.

3.2.2. Common Functions

As discussed in section 1.1.1, the LaTeX picture environment is a special environment for inserting simple drawing commands into a LaTeX file. It only provides lines with slope (m, n) for small integer values of m and n (maximum 6 for lines and 4 for vectors, which are lines with arrows). So if a user draws a line of general slope, the program should move it to a 'latex line'. In Dotty, lines are presented by edges and sides of nodes with shape like triangle, diamond, parallelogram and trapezium. Vectors are the edges with arrow. The following functions are needed to meet this requirement:

- Change number to integer: the built-in *toint* function truncates the decimal part of a number; for more accuracy, this function will increment the number if the decimal part is greater than 0.5. It will be used to set m and n in line or vector slope (m, n) to integer.
- Remove common divisors: this function removes the common divisors of m and n in line or vector slope (m, n) and replaces it with the same slope (a, b).
- Adjust slope and its length: after removing the common divisors of the slope (m, n), if a and b in slope (a, b) are still greater than the maximum integers allowed, this function will iterate through the allowed integer pairs to find the closest match, slope (c, d) by comparing b/a and d/c. It also will adjust the length of the line or vector.

- Find minimum number: this function returns the minimum value of two numbers. It is used to set the maximum divisor to remove the common divisors of m and n in slope (m, n), and to set the diameters of circles and double circles.

3.3. Implementation of Graphviz LaTeX Tool

Figure 3.3-1 outlines the major structure of *latex.lefty* script with the comments given by leading #. Pseudo codes for `savelatexfile` function, the functions for saving different node shapes, and the two major common functions `removeCommonDivisor` and `adjustSlope` are listed in Appendix E-1.

```
# load the script
load dotty.lefty;

# initialize latex table
latex [];

# override protogt table
latex.protogt [];

# override protovt table and set the window name
latex.protovt [name = Latex Generator];

# override init function called in maiin
latex.init ();

# main function to start up the program
latex.main ();

# add action 'save latex file' to general actions table
latex.protogt.actions.general [save latex file] (gt, vt, data);

# customize 'do layout' action in general actions table
# to allow choosing different layout tools
latex.protogt.actions.general [do layout] (gt, vt, data);

# function to save latex commands to a Tex file
latex.protogt.savelatexfile (gt, name, type);

# function to remove common divisors for line slope
# called when processing nodes and edges in savelatexfile function
latex.removecommondivisor(x, y);

# global function to return minimum number
# called by removecommondivisor to choose the common divisor
min (a, b);
```

```
# global function to change number to integer without truncating
# used by removecommondivisor for its arguments or line slope
integer (a);

# function to adjust slope and its length
# called when processing nodes and edges in savelatexfile function
latex.adjustslope (sx, sy, lx, ly);

# functions to save nodes in different shapes
latex.savenode.box (fd, gt, node);
latex.savenode.circle (fd, gt, node) ;
latex.savenode.ellipse (fd, gt, node) ;
latex.savenode.triangle (fd, gt, node) ;
latex.savenode.doublecircle (fd, gt, node) ;
latex.savenode.diamond (fd, gt, node);
latex.savenode.parallelogram (fd, gt, node);
latex.savenode.trapezium (fd, gt, node);
latex.savenode.Msquare (fd, gt, node);

# override and add 'save latex file' to general menus table
latex.protovt.menus [[general [save latex file]]];

# entry point of the program
latex.main ();
```

Figure 3.3-1 Main Structure of latex.lefty

3.4. Sample drawing of Graphviz LaTeX Tool

The sample lattice diagram as in reference [54] is drawn in Graphviz LaTeX tool as shown in Figure 3.4-1. Figure 3.4-2 is the graph visualized in DVI viewer based on the TeX output. The corresponding LaTeX commands generated are listed in Appendix D.

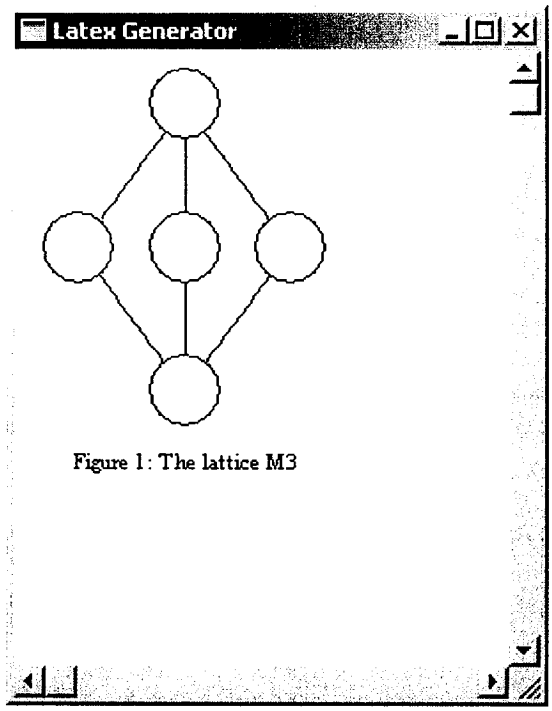


Figure 3.4-1 Sample Lattice Diagram in Graphviz LaTeX Tool

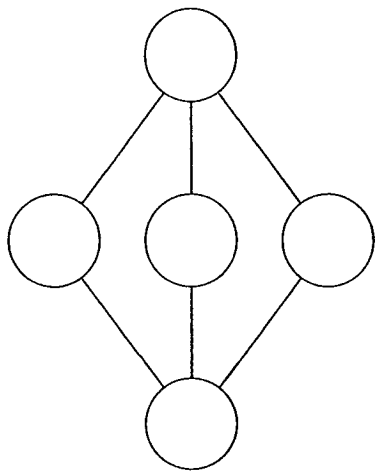


Figure 1: The lattice M3

Figure 3.4-2 Sample Lattice Diagram Generated from TeX File

Chapter 4. Drawlets LaTeX Tool

Drawlets LaTeX Tool is a customized graph editor based on Drawlets framework which generates LaTeX commands directly from drawings. Section 4.1 will briefly review the design of Drawlets framework by presenting its fundamental roles, package and class diagrams; and the remaining sections discuss our work on the design, implementation and sample drawings of our LaTeX tool. The common functions `removeCommonDivisor` and `adjustSlope` created for the Graphviz LaTeX tool are reused and just implemented in a different language, Java, thus will not be discussed again.

4.1. Drawlets Design Overview

Interfaces are the most powerful feature of Java which allows individual object to inherit from different useful hierarchies due to the lack of multiple inheritances in Java [55]. The design of the Drawlets framework is captured in interfaces to describe the fundamental roles, and various kits of abstract and concrete classes are built on top of it. This enables separating design inheritance from implementation inheritance. See More Background on Drawlets in [36].

4.1.1. Fundamental Roles

Drawlets has the following fundamental roles: `Drawing`, `DrawingCanvas`, `DrawingStyle`, `Figure`, `Handle`, `Locator` and `Tools`. Figure 4.1-1 from the Drawlets User's Guide [36] presents the relationships between these roles as discussed below. (Note: some comments are from source codes)

Drawing: Defines a generic `Drawing` which holds a `SequenceOfFigures` (or a sequence of Figures) that can be painted and potentially manipulated. It is expected that this will be the fundamental unit to store and retrieve diagrams, pictures, etc.

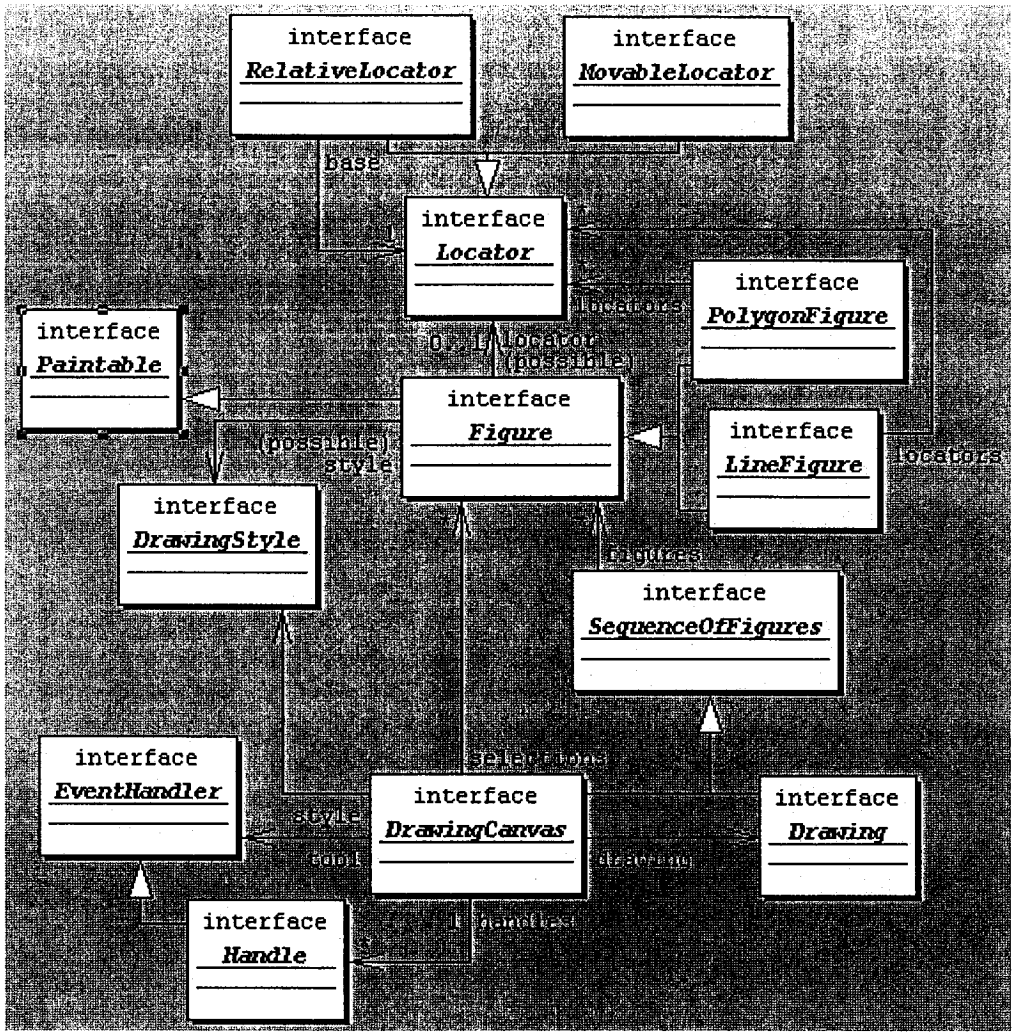


Figure 4.1-1 Class Diagram for Drawlets Fundamental Roles

DrawingCanvas: Allows a `SequenceOfFigures` to be manipulated. It has an active `Tool` (implementer of `InputEventHandler`) which can be used to create or manipulate various

kinds of Figures. When it receives an Event, it will pass it on to either a prominent `Handle` associated with Figures or its active `Tool`.

DrawingStyle: Defines a set of getters and setters for drawing styles. `DrawingStyles` are used by `Figure` and `DrawingCanvas` to define various attributes such as color and font.

Figure: Defines a generic `Figure` that can be drawn and potentially manipulated on a `DrawingCanvas`. Figures are essentially treated as rectangles by the framework but can visually represent any shape and they draw themselves using one or more properties of a `DrawingStyle`.

Handle: Defines protocols for Handles which are event handlers that may visibly appear on a `DrawingCanvas`. Handles provide direct editing behavior for Figures through Locators. When a `Figure` is selected by the selection `Tool`, it presents a set of Handles. Manipulating a `Handle` changes some properties of its `Figure` or performs some actions [56].

Locator: Defines protocols for objects that provide 2-D coordinates. It's used by most Figures and Handles to locate them.

Tools: Unlike other roles, `Tools` are the implementers of the `InputEventHandler` interface. They perform operations on the figures of a drawing. Tools are responsible for

interpreting a user action, such as a mouse click or a keyboard press, by performing some action on the drawing, such as moving a `Figure`, creating a new `Figure` etc.

4.1.2. Class Diagrams

Visual Paradigm for UML [57] supports two kinds of code reverse engineering: to reverse from codes that are generated in the code generation process, or to reverse from existing Java source files. To better understand the architecture of Drawlets framework, Visual Paradigm was used as reverse engineering tool to generate class models based on the existing Drawlets Java source files, and the desired class diagrams are then created.

Among the fundamental roles discussed in the previous section, `Figure`, `Handle`, `Locator` and `Tools` are the most important ones with many subclasses implemented. Patterns for Drawlets [36] is based on the Appendix of a paper written by Ralph Johnson [39], it documents Drawlets framework as a set of patterns where these roles are described.

The most important interface in Drawlets is `Figure`, which is the interface of drawing elements. Figures are responsible for rendering, hittesting, and notifying dependents when their appearance changes. Figure 4.1-2 is the class hierarchical diagram showing the implementers of `Figure` supported by Drawlets framework. It is obvious that each kind of drawing element is an implementer of `Figure`. By default, Drawlets supports the simple geometric objects like `Ellipse`, `RectangleShape`, `PolygonShape`, `Line`, and `TextLabel`.

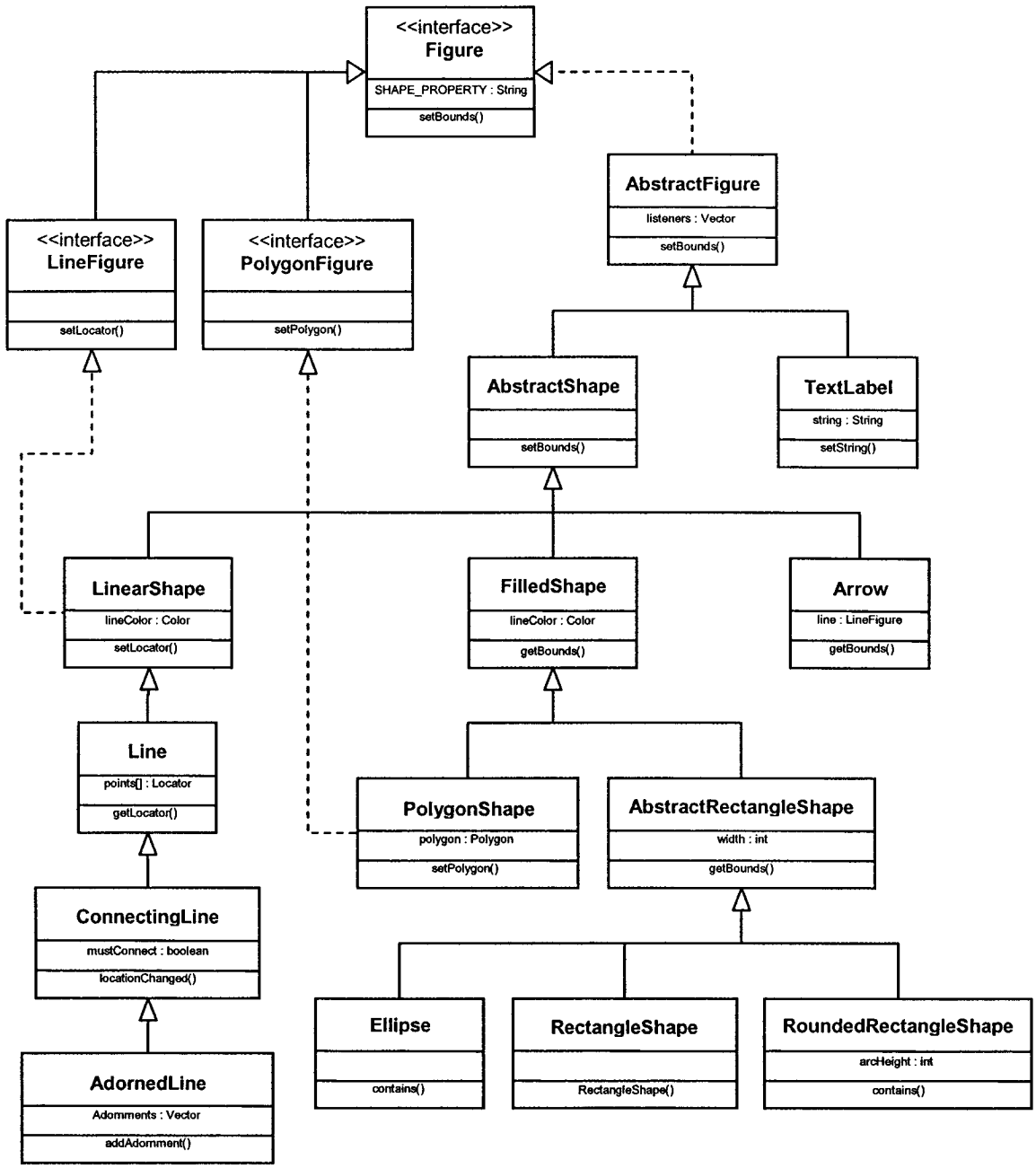


Figure 4.1-2 Class Hierarchical Diagram for `Figure` Implementers

Figures 4.1-3 shows the class hierarchical diagram for `Handle` implementers. Handles are `InputEventHandlers`. Several types of general purpose handles are already included with Drawlets. They provide the ability to do things ranging from resizing Figures to

creating Lines through the corresponding handles like subclasses of `BoundsHandle` and `ConnectingLinePointHandle`. Handles can be attached to any part of a `Figure`, and they move when the `Figure` moves.

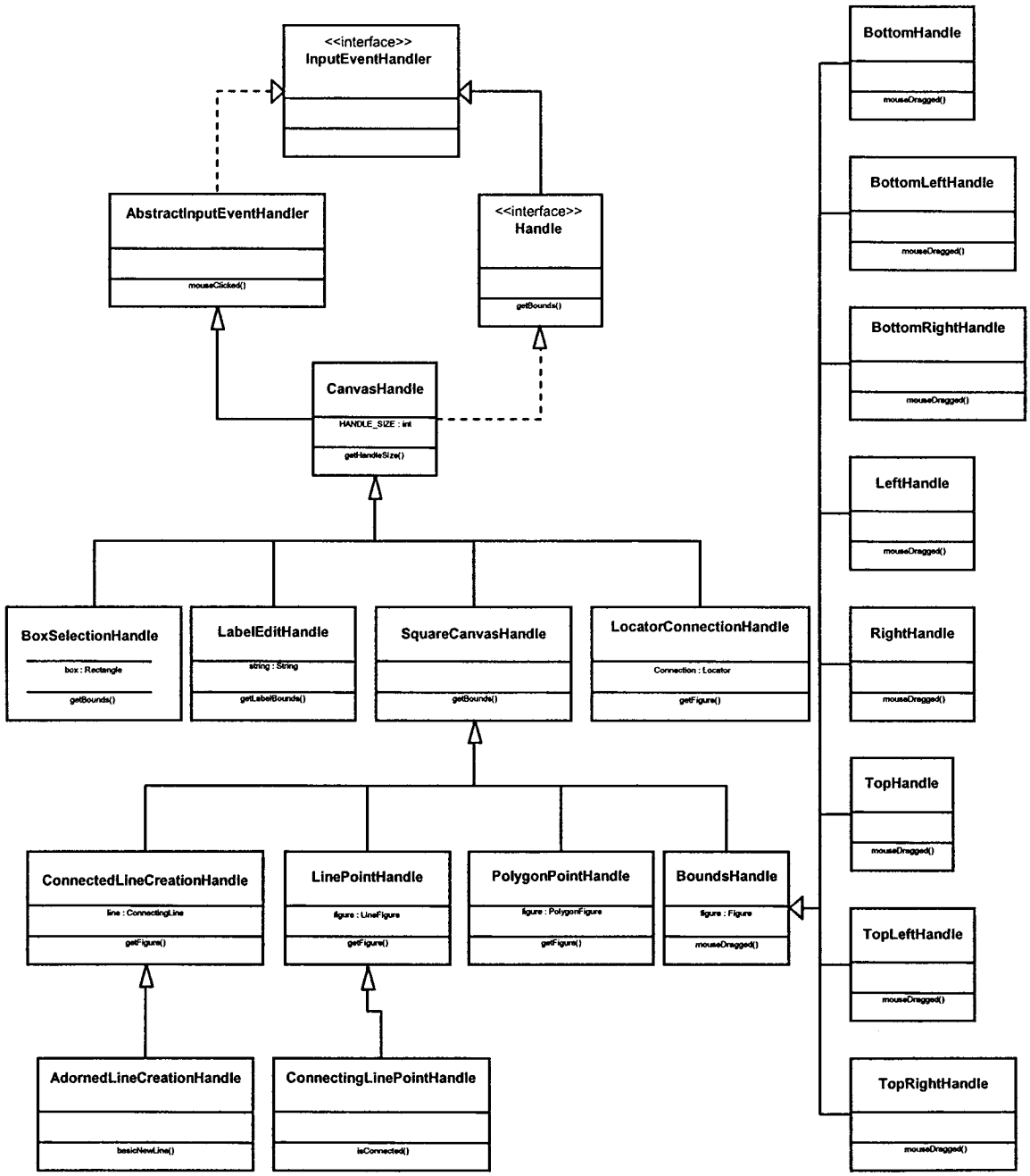


Figure 4.1-3 Class Hierarchical Diagram for `Handle` Implementers

Three interfaces define the functionality which Locators provide as shown in Figure 4.1-4. First, `Locator` itself is able to provide its x, y, r and theta; a `MovableLocator` can additionally manipulate these values; and a `RelativeLocator` is a `Locator` whose position is based on some other position. Locators are the preferred means of specifying position in Drawlets, when combined with LocationListeners (discussed in section 5.2.2) they allow constraints to be applied to Figures based on location.

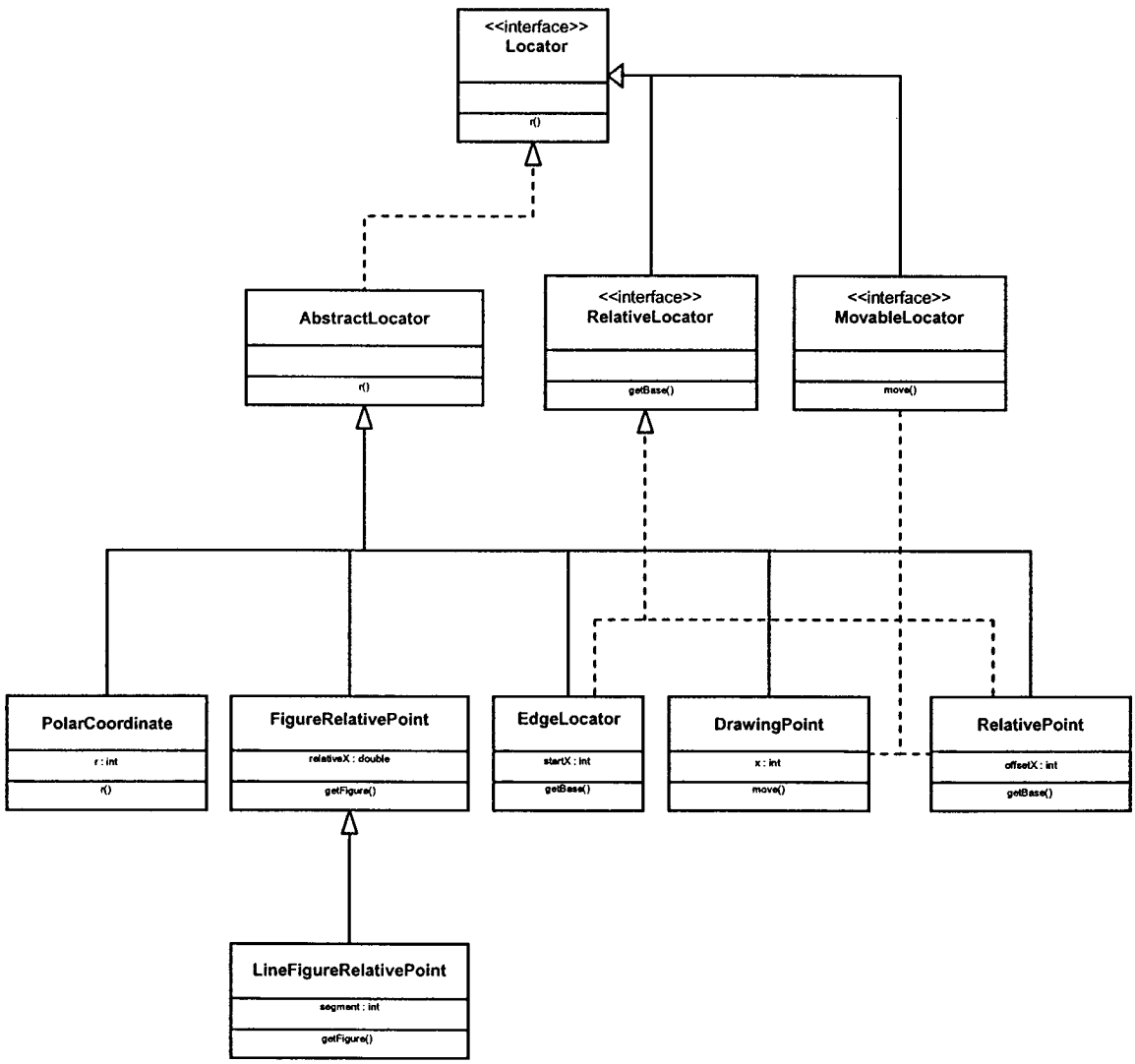


Figure 4.1-4 Class Hierarchical Diagram for `Locator` Implementers

Tools are simply InputEventHandlers that know about their canvas and act on it depending on the events generated. They allow users to manipulate Figures, create new Figures, or perform operations upon a `Figure` or the entire `DrawingCanvas`. For example, a `SelectionTool` can be used to manipulate handles and move Figures around. As shown in Figures 4.1-5, there is a corresponding tool for various different types of Figures.

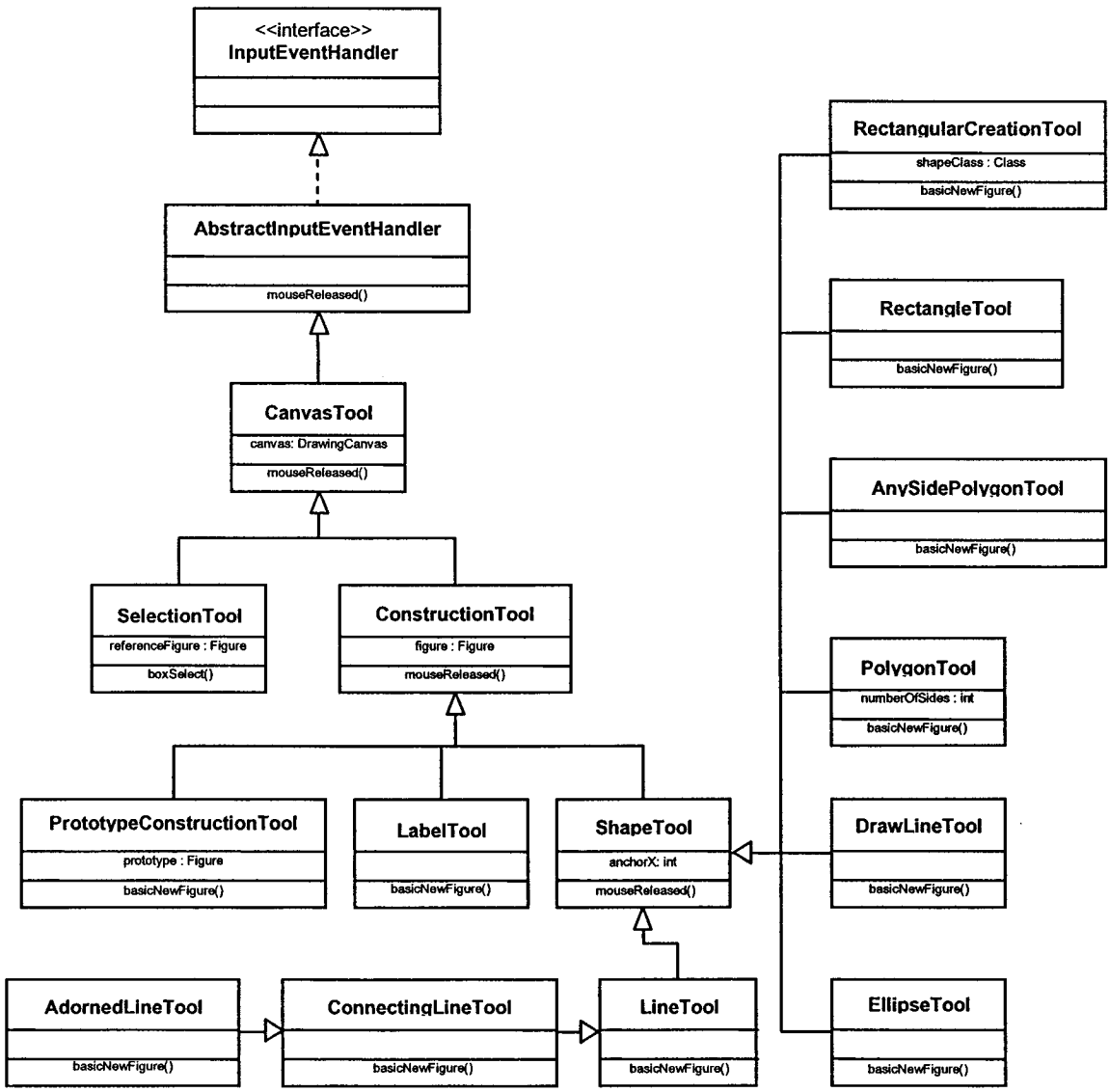


Figure 4.1-5 Class Hierarchical Diagram for Tools

4.1.3. Package Design

Drawlets is not just a HotDraw rewritten in Java. It was engineered using Patterns for Building an Unusually Adaptable Java Framework [58]. It takes advantage of many of the features of the Java Language and the more adaptable parts of the available libraries [36]. One general principle in building the adaptable java framework is to use Packages as Building Blocks. The packages categorized should build cleanly on one another so that they can be easily replaced by alternate versions without impacting other packages. Figure 4.1-6 shows the different categories of packages defined and their inter-relationships.

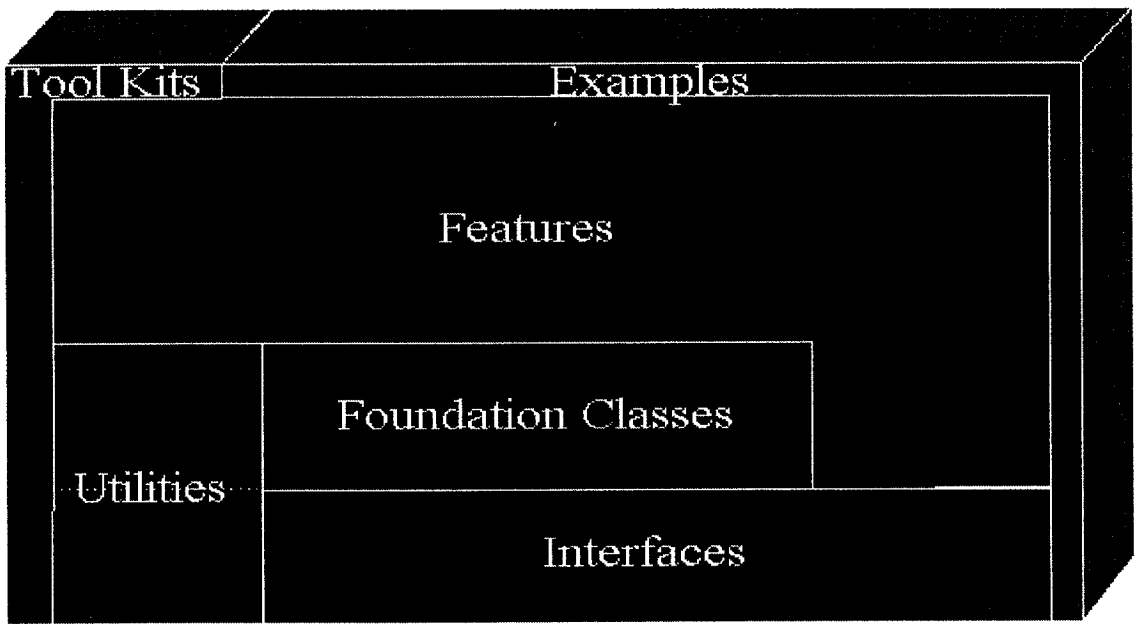


Figure 4.1-6 Package as Building Blocks

Following the principle above, the Drawlets framework was designed to group its interfaces and classes into different packages as listed in Table 4.1-1, where the package

names are simplified by suppressing the leading `com.rolemodelsoft`. The same rule is applied in Figure 4.1-7, which presents the relationships between these packages.

Table 4.1-1 Drawlets Framework Packages

Package Type	Package Name	Description
Interface Package	drawlet	All interfaces in the framework
Foundation Class Package	drawlet.basics	Abstract classes and simple classes implements interfaces
Utilities Package(s)	drawlet.util	Utilities interfaces and classes which do not rely on anything
Feature Kit Packages	drawlet.shapes drawlet.shapes.ellipse drawlet.shapes.lines drawlet.shapes.polygons drawlet.shapes.rectangles drawlet.text	Features for general shapes Features for ellipse shapes Features for various types of lines Features for polygon shapes Features for rectangular figures Features for text related figures
Tool Kit Package(s)	drawlet.awt drawlet.jfc	AWT windowing toolkit Swing/JFC components
Example Packages	drawlet.examples drawlet.examples.awt drawlet.examples.jfc drawlet.examples.graphnode	Example class implements Observable Examples using the AWT window toolkit Examples using the Swing/JFC components Example to use Drawlets without a tool palette

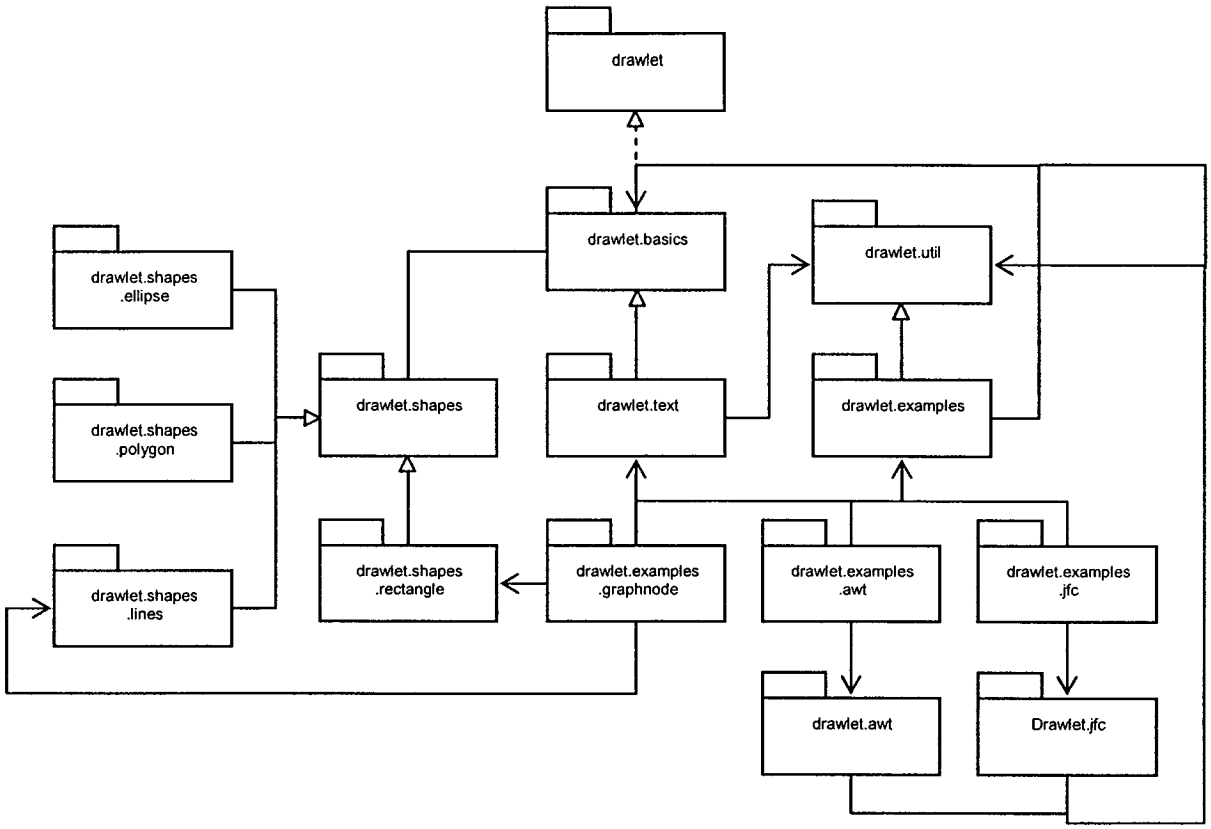


Figure 4.1-7 Drawlets Framework Package Diagram

4.2. Design of Drawlets LaTeX Tool

As with the Graphviz LaTeX Tool, the purpose of our Drawlets LaTeX Tool is to save the drawings as LaTeX commands directly. This requires the tool to have a graphical user interface (GUI) which allows user drawing Figures with different shapes, such as rectangle, circle etc., lines and texts. It should also provide ways to edit figures and save drawings in LaTeX picture commands format to a TeX file.

4.2.1. Customizing Drawlets Example

By examining and running the simple examples provided by the Drawlets framework, it was found that the `SimpleModelPanel` class is sufficient to get started. Becasue JFC is a

superset of AWT, with many more components and services [59], the one under the `examples.jfc` package was chosen as the basis class. Since no updates for the framework are required, the Drawlets framework was built into `Drawlets.jar` and used as library in the current project's LatexGen created for the LaTeX tool.

Our project creates only one package latex.main containing one class `LatexGenerator` based on `SimpleModelPanel`. All the images under Drawlets bin folder are stored under images folder and `Drawlets.jar` is saved under lib folder. The class `LatexGenerator` uses all the functions in `SimpleModelPanel` with the following things created and updated:

- Create `SaveLatexAction` inner class to generate LaTeX commands for all kinds of Figures supported by Drawlets (except `RoundeRectangleShape`), which include `RectangleShape`, `PolygonShape`, `Ellipse`, `AdornedLine`, `Line` and `TextLabel`. When the action is invoked, it iterates through Figures contained in the drawing to get their positions and transform them to LaTeX format.
- Override two functions `getToolBar()` and `getToolPalette()` defined in `SimplePanel` class with correct image paths and adding tooltips.
- Add **TEX** image icon to ToolBar allowing perform SaveLatexAction.
- Create function `saveLatex()` to store the LaTeX commands generated from drawings with LaTeX Prefix and Suffix to a TeX file.
- Reuse the two common functions `removeCommonDivisor()` and `adjustSlope()` created for Graphviz LaTeX tool to support the limited LaTeX lines.

4.2.2. Class Diagram of Drawlets LaTeX Tool

A simplified class diagram (Figure 4.2-1) is drawn to present the different roles (high level abstract classes) related to `LatexGenerator` and the relationships between them. `LatexGenerator` extends `SimplePanel` having two `JToolBar`, one for holding the list of actions to be performed on drawing and another for storing different tools like `SelectionTool` and `Figure` tools to be manipulated on Figures. It implements a `SimpleDrawingCanvas` which resides in a GUI specific placeholder `JDrawingCanvasComponent`. It obtains the drawing through `SingleDrawingModel`, which is then manipulated by `CanvasTool` and `CanvasHandle` through Locators on the `SimpleDrawingCanvas`. The major classes are discussed further as below.

Except for those used by Drawlets example `SimpleModelPanel`, some other Java packages are also used which are not included in the diagram. They are `java.util` (use `Vector` and `Enumeration` etc. to iterate through Figures in a drawing model), `java.lang` (Math functions) and `java.io` (create and output LaTeX file).

AbstractLocator: Defines protocols for objects that provide 2-D coordinates with default implementation. It's used by most Figures (`AbstractFigure` and `AbstractLocator`) and Handles (`CanvasHandle`) to locate them.

CanvasTool: Offers a simple base for Tools attached to a `DrawingCanvas` (`SimpleDrawingCanvas`). Subclasses would typically provide a constructor that takes a `DrawingCanvas` as an argument.

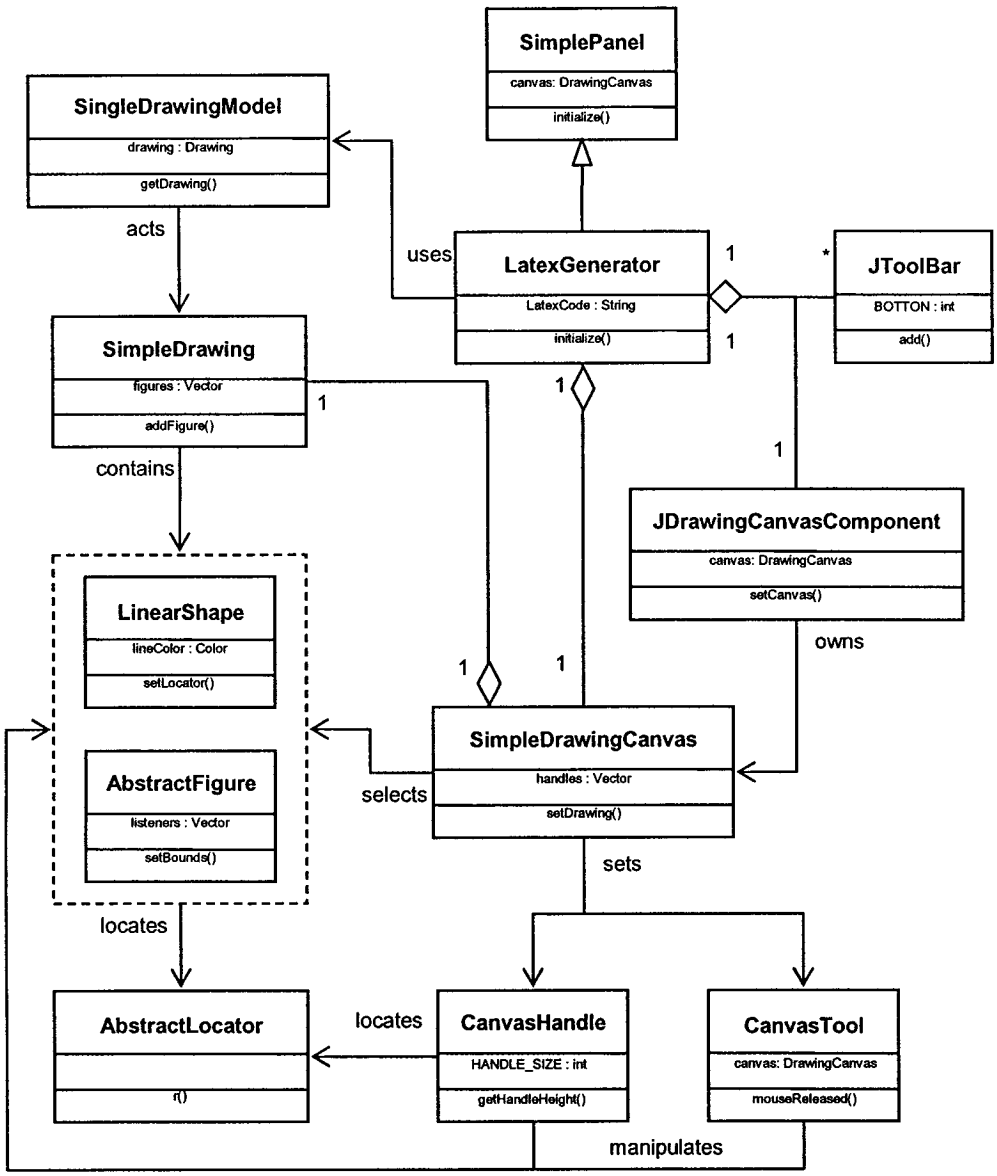


Figure 4.2-1 Simplified Class Diagram of Drawlets LaTeX Tool

CanvasHandle: Offers a simple base for handles to be used on a canvas (`SimpleDrawingCanvas`). Subclasses need to provide, at a minimum, implementations for `bounds()` and `paint(Graphics)`. Typically, these types of handles wait for some outside object to ask them to `takeControl()` (e.g. mouse down on top of it), then they

do something in response to mouse events, and finally they `releaseControl()` (e.g. mouse up).

JDrawingCanvasComponent: A Swing component which owns and acts on a `DrawingCanvas` (`SimpleDrawingCanvas`). `JDrawingCanvasComponent` is GUI specific placeholder which allows `SimpleDrawingCanvas` to reside in JFC applications.

SimpleDrawing: A sequence of Figures to be manipulated on `SimpleDrawingCanvas`.

SimpleDrawingCanvas: Implements both `DrawingCanvas` and `InputEventHandler`. It supports basic functionality necessary to provide a meaningful working version of a `DrawingCanvas` which is responsible for displaying and allowing the manipulation of Drawings.

SimplePanel: Provides a basic `DrawingTool` consisting of `JToolBar` (toolBar and toolPalette) and a `JDrawingCanvasComponent` which holds a `DrawingCanvas`.

SingleDrawingModel: Defines the customized operations to be manipulated on the drawing, such as `clearDrawing()` and `saveDrawing()`.

4.3. Implementation of Drawlets LaTeX Tool

As mentioned in section 4.2.1, `LatexGenerator` is based on the example class `SimpleModelPanel`. This section will list the pseudo codes to describe the main structure

of the `LatexGenerator` class. The pseudo codes are in simplified Java format and the comments are added inside the codes as in Figure 4.3-1. The detailed pseudo codes for the class are listed in Appendix E-2.

```
// the package that LatexGenerator created in
latex.min;

// packages used by SimpleModelPanel
import …

class LatexGenerator extends SimplePanel {
    // define parameters used

    // constructors are in the same formats as for SimpleModelPanel
    LatexGenerator() {…}

    // the component holding the canvas
    getCanvasComponent() {…}

    // the model manipulated
    getModel() {…}

    // Initialize class
    initialize() {
        // add toolbar for clear, save and restore ations
        toolBar.add (…)

        // Class for Latex code generation
        class SaveLatexAction extends AbstractAction {
            SaveLatexAction (name, icon, model, fileName) {};

            actionPerformed(action event) {
                // process each figure of the drawing, such as box, line etc.
                for (first figure; next figure;) {
                    // set each figure's boundary and origin;
                    // save in LatexCode;
                }
                // save LatexCode to TEX file
                saveLatex(LatexFileName, LatexCode);
            }
        }
        // add icon for SaveLatexActioin
        toolBar.add (new SaveLatexAction(with tooltip);
    }

    // override the getToolBar() with tooltips and correct pathes
    JToolBar getToolBar() {…}

    // override the getToolPalette()  with tooltips and correct pathes
    JToolBar getToolPalette() {…}

    // function to store the Latex code of drawings into TEx file
    saveLatex(filename, latexcode) {}

    removeCommonDivisor(width, height) {…}
```

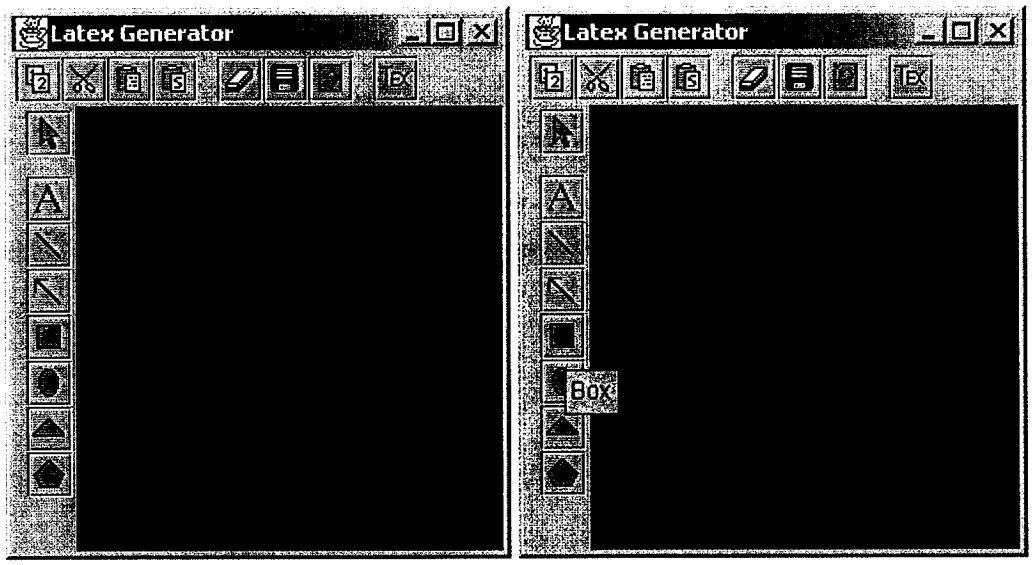
```
    adjustSlope(sx, sy, width, height) {…}

    main() {
        initiates a LatexGenerator();
    }
}
```

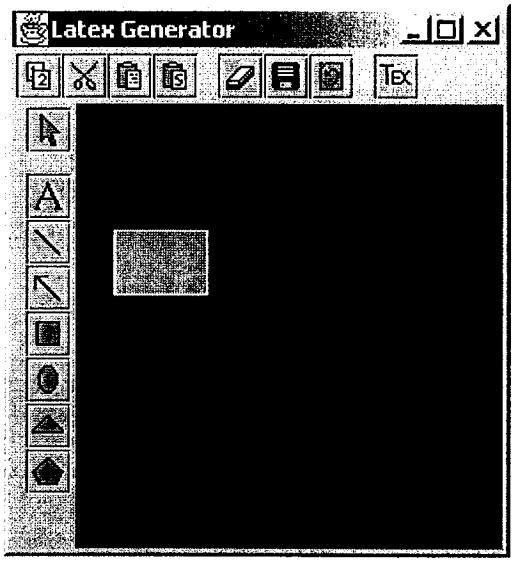
Figure 4.3-1 Main Structure of LatexGenerator Class

4.4. Sample Drawing of Drawlets LaTeX Tool

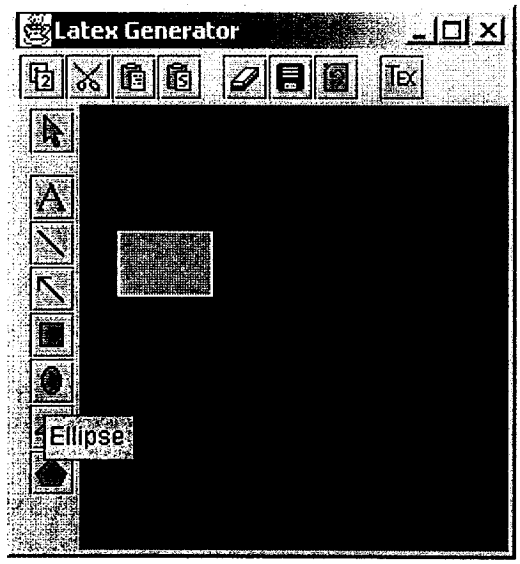
A sequence of drawings is listed in Figure 4.4-1 to show the usage of Drawlets LaTeX tool.



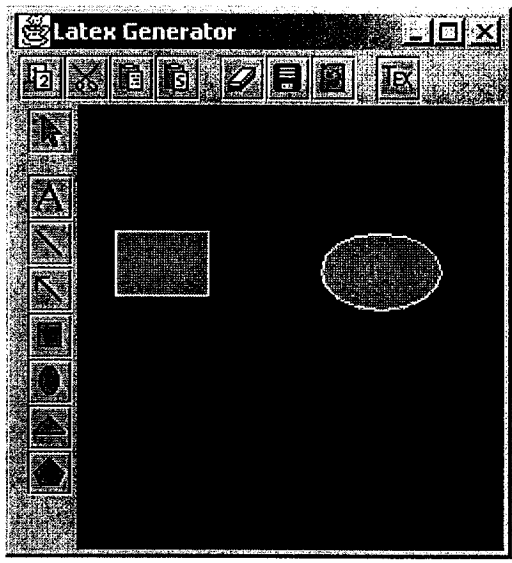
(a) Start the Drawlets tool (b) Left click on Box icon



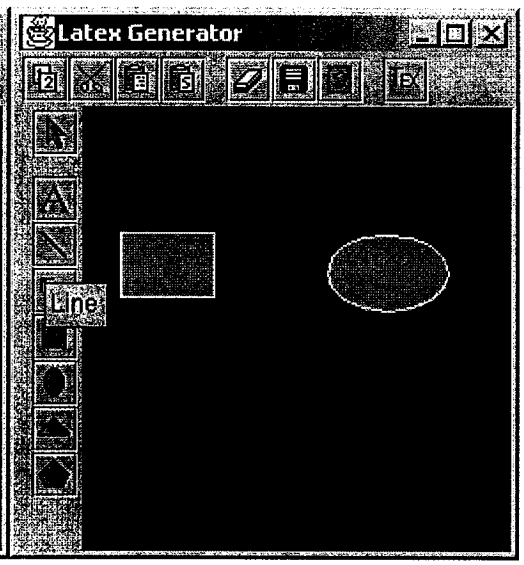
(c) Left click and drag mouse on drawing area, release mouse button



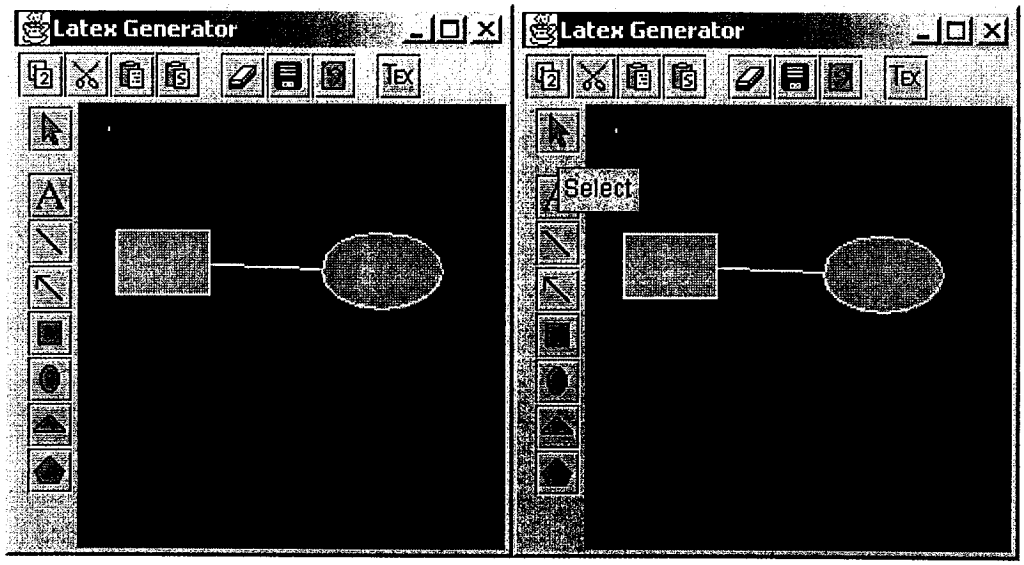
(d) Left click on Ellipse icon



(e) Left click and drag mouse on drawing area, release mouse button

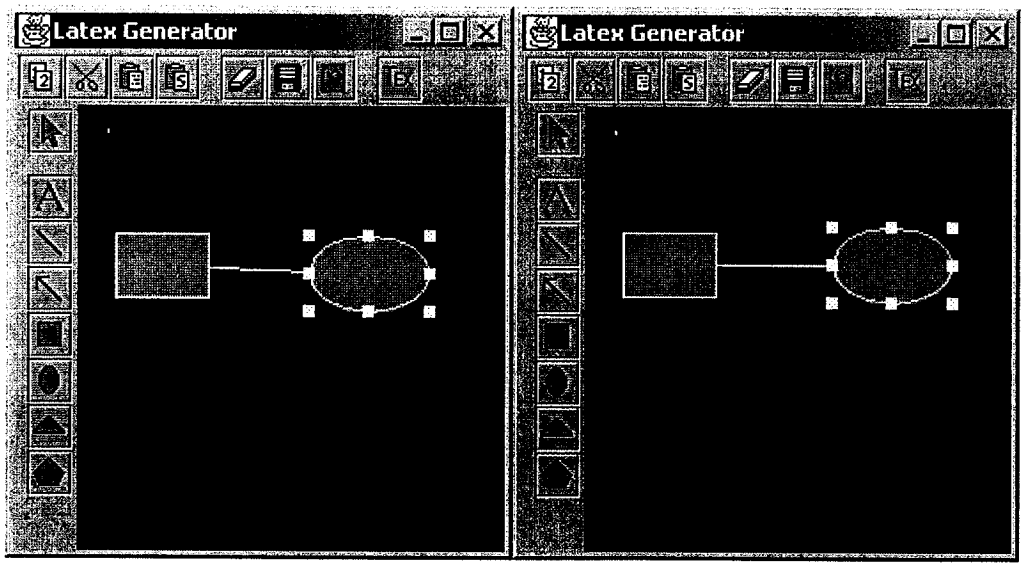


(f) Left click on Lin icon



(g) Left click over Box and drag mouse onto Ellipse, release mouse button

(h) Left click on Select icon



(i) Left click over Ellipse

(j) Move Ellipse up utill the line is straight, release mouse button

Figure 4.4-1 Sequence of Drawings in Drawlets LaTeX Tool

The sample lattice diagram as in reference [54] is drawn in Drawlets LaTeX Tool as shown in Figure 4.4-2. Figure 4.4-3 is the graph visualized in DVI viewer based on the

TeX output generated. The corresponding LaTeX commands generated are listed in Appendix D.

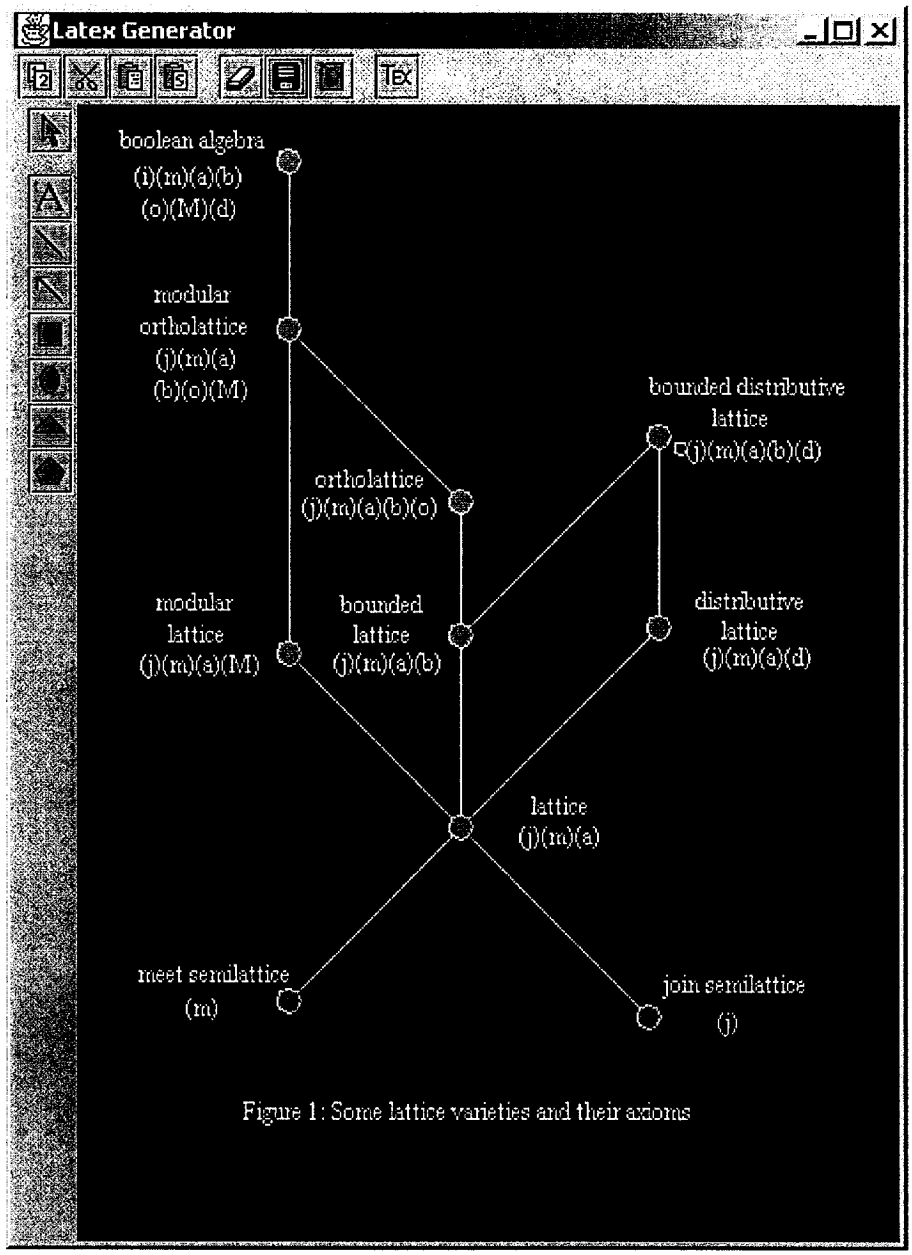


Figure 4.4-2 Sample Lattice Diagram in Drawlets LaTeX Tool

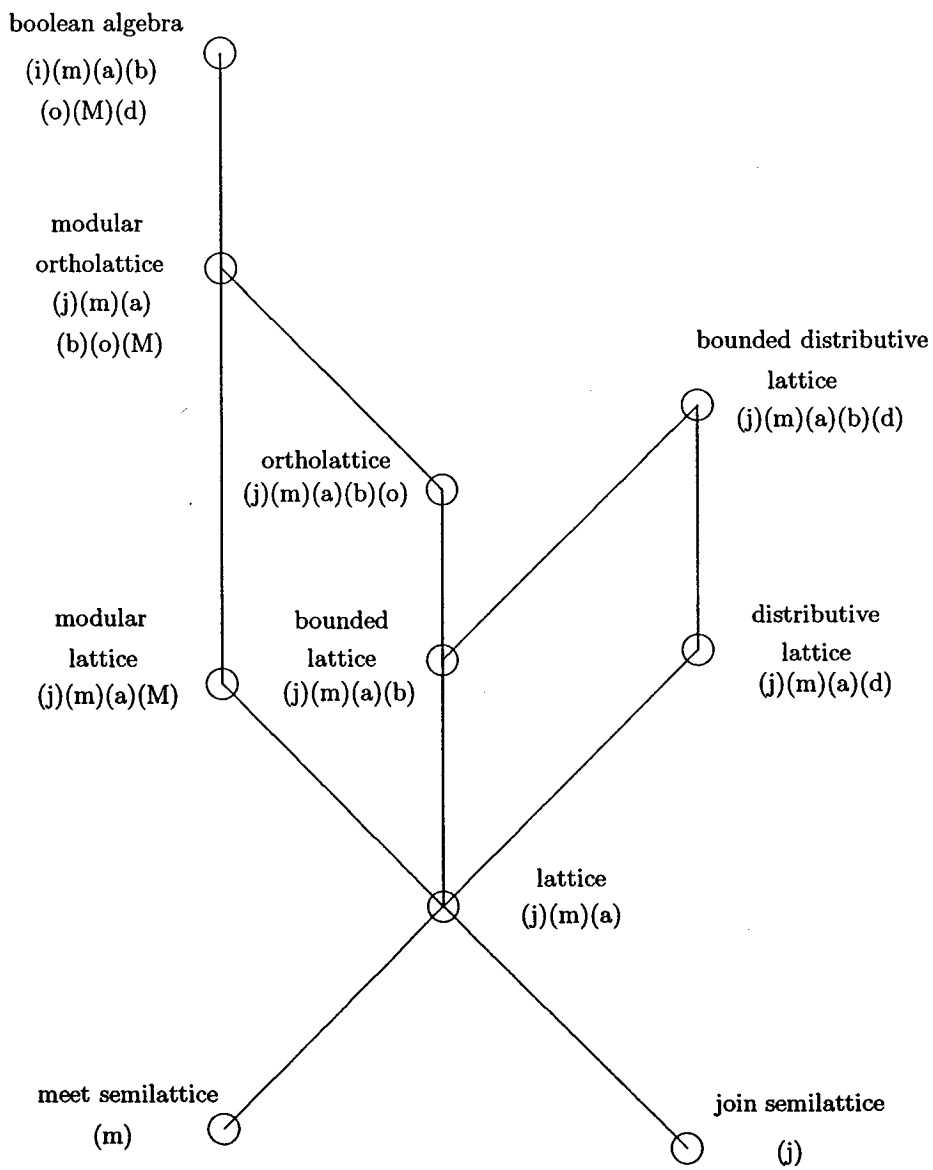


Figure 1: Some lattice varieties and their axioms

Figure 4.4-3 Sample Lattice Diagram Generated from TeX File

Chapter 5. Comparison of the Two LaTeX Tools

In the previous two chapters we have covered the design and implementation of Graphviz and Drawlets LaTeX tools. This chapter discusses the two tools by focusing on their supported features, applied layout algorithms or constraints, graphical user interface and language used to implement the tool.

5.1. Features Supported

Both Graphviz and Drawlets LaTeX Tools support the basic LaTeX picture commands such as `\put`, `\oval`, `\line`, `\vector`, and `\makebox`, which allow saving the drawings with simple connected lines and vectors, ellipses, rectangles, triangles and the specified texts to a TeX file. They both supports the basic functions to create and manipulate on the graphs, such as *select*, *cut*, *copy*, *paste*, *clear* (*new graph*) and resize the graphs. They can save the drawings to a file and retrieve them later. Besides these common features, they support some different functions and have certain limitations.

5.1.1. Graphviz LaTeX Tool

As shown in Figures 5.1-1 and 5.1-2, Graphviz LaTeX Tool supports also the following shapes: circle, filled circle, double circle, ellipse, triangle, diamond, parallelogram, trapezium and msquare, and functions like *undo*, *save graph as* to save drawing to a different file, *zoom in*, *zoom out* and *print graph*. It allows deleting all the nodes with same attributes or grouping them to a single node. It provides multiple views through *copy view* and *clone view*, especially a *birdseye view* which always displays the entire graph.

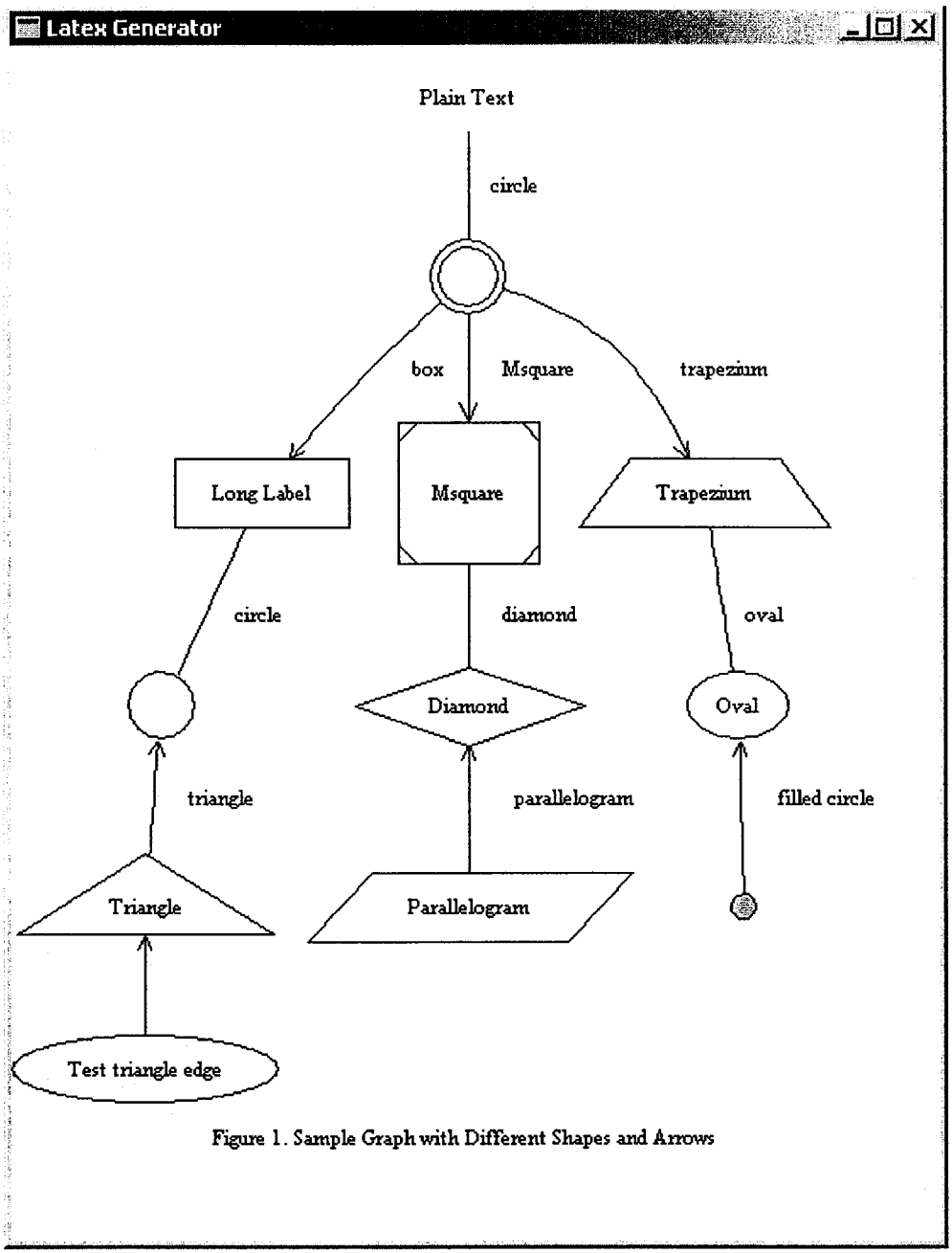


Figure 5.1-1 Different Shapes and Lines Supported by Graphviz LaTeX Tool

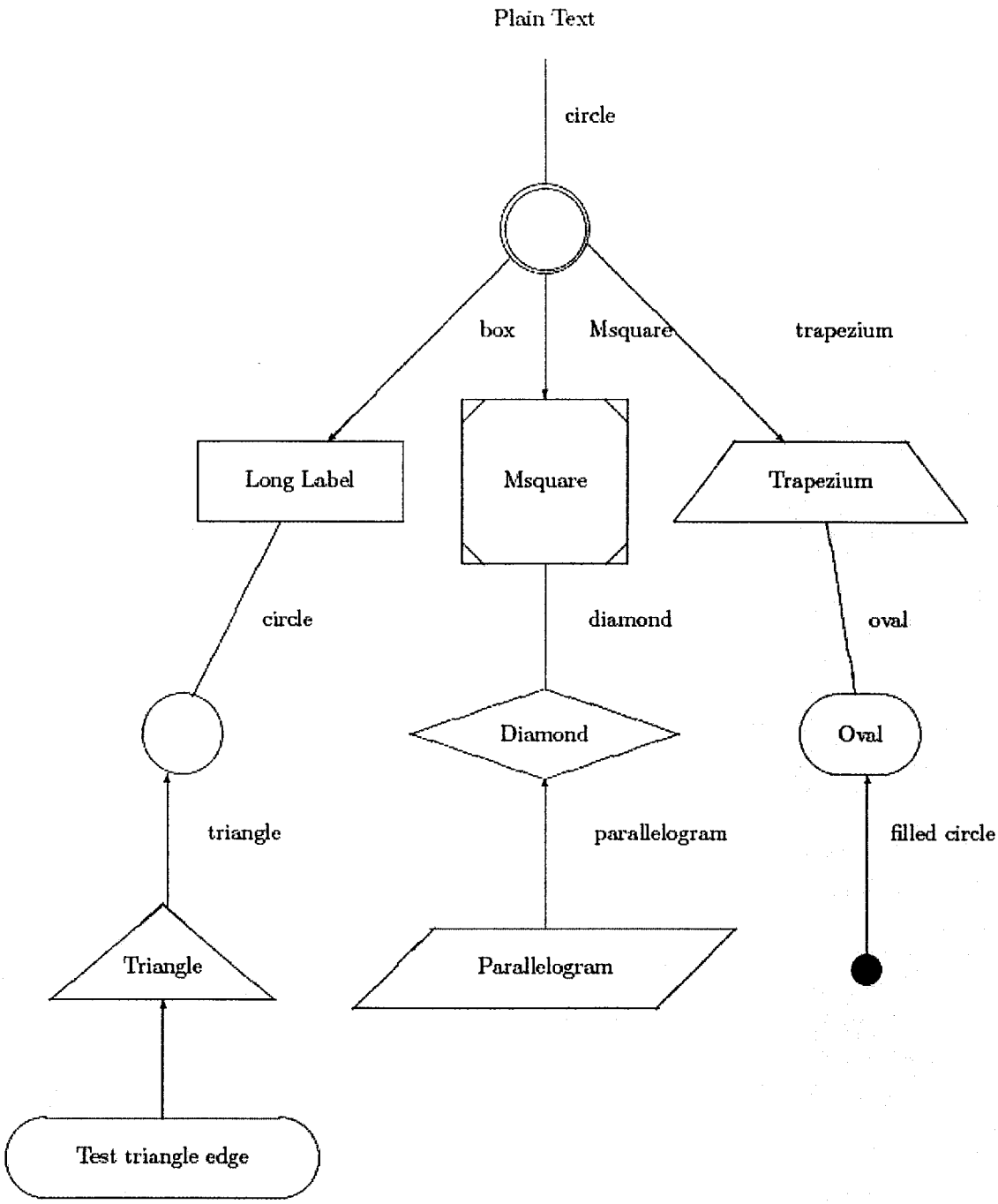


Figure 1. Sample Graph with Different Shapes and Arrows

Figure 5.1-2 Different Shapes and Lines in Graphviz Tool Generated from TeX file

Graphviz LaTeX Tool has the following limitations which are inherited from Dotty:

- Triangles are in the predefined form and can be changed only by its height and width.
- Lines are used to connect nodes, i.e. edges. No individual lines can be drawn.
- The mouse's middle button is used to create edges between nodes. If there is no middle button, no edges can be drawn.
- For all circles, there are limitations from LaTeX picture commands as in Appendix B. The maximum diameters for hollow and filled circles are 40 and 15 points, respectively. This may result in the smaller circles with edge not connected in DVI viewer.

5.1.2. Drawlets LaTeX Tool

There are also extra shapes supported by Drawlets LaTeX tool: pentagon and circle. Actually, circle can be drawn with ellipse icon by dragging it to a circle visually, and several circles can be overlapped to produce double circle or even triple circle. Lines and vectors are used to connect different figures, but they also can be drawn alone in any directions. Triangles and pentagons can be draw into any form that you what by clicking and moving left mouse button. The special function supported by the tool is to copy text from other application and paste it to the current drawing. Figures 5.1-3 and 5.1.4 shows these as an example.

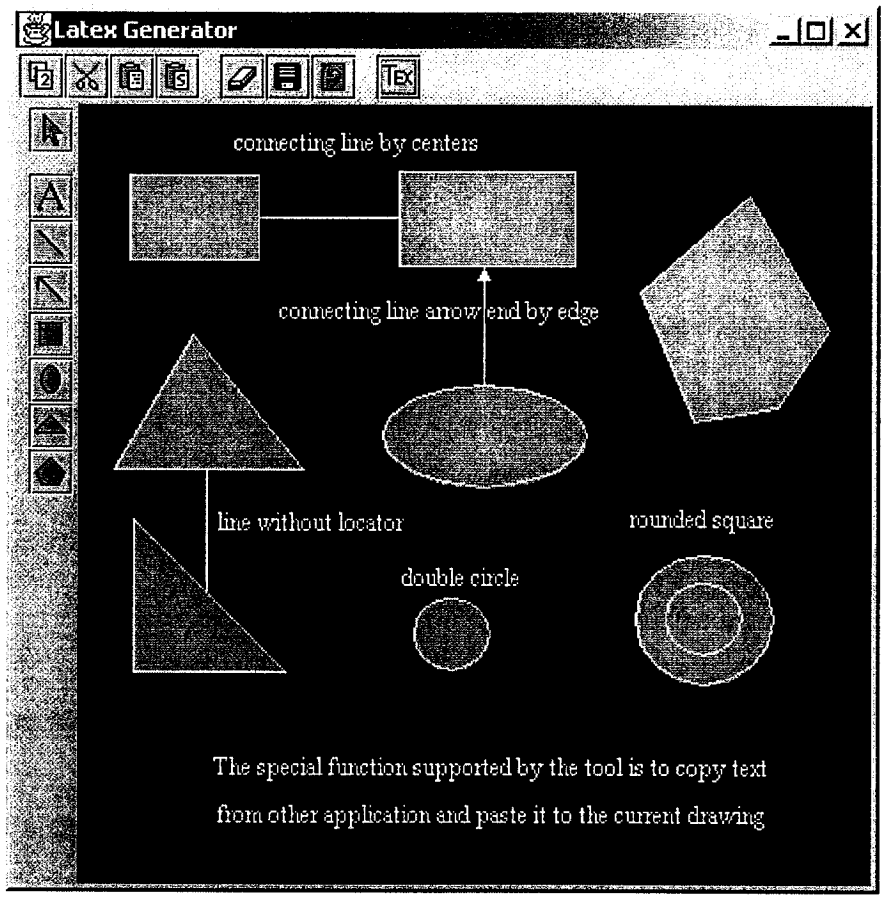
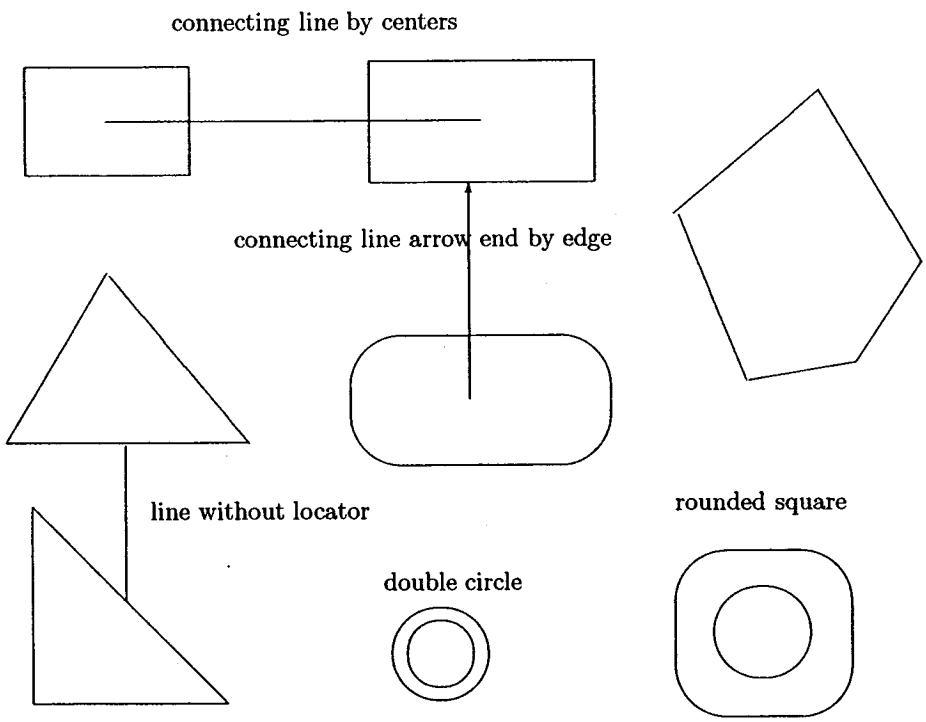


Figure 5.1-3 Different Shapes and Lines Supported by Drawlets LaTeX Tool



The special function supported by the tool is to copy text

from other application and paste it to the current drawing

Figure 5.1-4 Different Shapes and Lines in Drawlets Tool Generated from TeX file

The followings are the limitations of Drawlets LaTeX Tool:

- The most important one is that lines connecting figures start and end in the figure center as shown in Figure 4.4-2. This is the way that Drawlets framework is designed to impose the constraint on Lines. Whereas for vectors, AdornedLine is used to connect figures by edge at the arrow end (see Section 5.2.2 for more discussions).
- The vectors can't be saved and restored to and from temp.rmd file.
- The flexibility for drawing lines, triangles and pentagons allows extremely irregular connections and Figures drawn, which may result in an unpleasant graph viewed in DVI viewer: this is due to the limited slope number allowed by LaTeX; and even the closest matched slope does not match.
- The limitation from the LaTeX picture command for circle size also applies for Drawlets tool, but in a different way. When the circle size is bigger than allowed, it's displayed as rounded square in DVI viewer as shown in Figures 5.1-3 and 5.1-4.

5.2. Graph Layout Algorithms and Constraints

Both tools provide a way to adjust the positions and connections between nodes automatically. Graphviz LaTeX Tool supports the static layout algorithms tools, *dot* and *neato*, thus changes the positions in one shot, whereas the Drawlets LaTeX Tool updates the connections between Figures dynamically through the applied constraints. Both have advantages and disadvantages as discussed below.

5.2.1. Graphviz LaTeX Tool

As mentioned in section 3.2.1.2, the Drawlets LaTeX Tool is customized to support both *dot* and *neato* layout algorithms tools through the '`do layout`' function. These layout algorithms rely on a specific set of fields to record position and drawing information. For example, each node has `ND_coord` attribute for the position of its center, `ND_width` and `ND_height` attributes for the size of the bounding box of the node, in inches. They will assign node positions, represent edges as splines, handle the special case of an unconnected graph, plus deal with various technical features such as preventing node overlaps [19]. With the position and size information set, the tool will then draw the nodes and edges of the graph by representing edges as line segments (even though some of them are drawn by *dot* as spline curves as in Figures 5.1-1 and 5.1-2).

In both algorithms, the first step is to call a layout-specific `init_graph` function. For example, dot calls the `dot_init_graph` function and neato calls the `neato_init_graph` function to initialize the graph for the specific algorithm. Initialization will then establish the data structures specific to the given algorithm. Both algorithms also end with the `dotneato` postprocess. The role of this function is to do some final tinkering with the layout, still in layout coordinates, such as attaching the root graph's label, and normalizing the drawing so that the lower left corner of its bounding box is at the origin. In addition to the graph, the function takes an algorithm-specific function used for setting node sizes [19].

dot is a four-pass algorithm for drawing directed graphs. The first pass `rank()` places the nodes in discrete ranks, an efficient way of ranking the nodes using a network simplex algorithm. The second `ordering()` sets the order of nodes within ranks to avoid edge crossings. The third `position()` sets the actual layout coordinates of nodes. The final pass, `make_splines()`, finds the spline control points for edges [17].

neato makes a 'spring model' layout of undirected graphs, in which nodes are treated as physical objects influenced by forces, some of which arise from the edges in the graph. The layout is then derived by finding positions of the nodes which minimize the forces or total energy within the system. The forces need not correspond to true physical forces, and typically the solution represents some local minimum. Such layouts are sometimes referred to as symmetric, as they tend to be the visualization of geometric symmetries within the graph. By default, *neato* draws edges as line segments to further enhance the display of symmetries [19].

The limitation with these layout algorithms is that after applying both layout algorithms, the fields that record position and drawing information are fixed by the layout library and cannot be altered by the application. Thus, user has no control over it at all. They may do more than what is desired. For example, node size is dependent on node label after doing layout and the user can't change it.

5.2.2. Drawlets LaTeX Tool

As mentioned in Patterns for Drawlets [36], Drawlets allows several ways of imposing constraints on Lines. The first way of adding functionality to Lines is used in `ConnectingLine` which can be forced always to be connected at both ends. The second is by making Lines use customized locators. These locators act as constraints on the Lines' position, and by registering a `Line` to listen to a `Figure` and then using a `RelativeLocator` provided by that `Figure`, changes in the Figure's position are propagated to the `Line`, the line then requests the updated position from the `RelativeLocator`.

By default, Drawlets uses the second way to impose the constraints on Lines. It uses Locators to specify the positions of Figures and the RelatedLocaitonListeners to provide the dynamic updating through the constraints applied to Lines based on the Locators. These constraints keep the Figures and Lines connected while you move them, which allow the drawing easy manipulated and better organized.

Locators like `FigureRelativePoint` and `LineFigureRelativePoint` associated in some way with Figures are actually FigureHolders as shown in Figure 5.2-1. If a Figure such as `AdornedLine` has a `FigureHolder`, it becomes a `RelatedLocationListener` on the held `Figure`. When the location of the held `Figure` changes, the figure (`AdornedLine`) will be notified and updates itself accordingly through asking a `LineFigureRelativePoint` for its new coordinates. This is how the connecting line between two Figures (e.g., `Figure` and

AdornedLine) works. Moving the Figure allows the connected line to follow the Figure around.

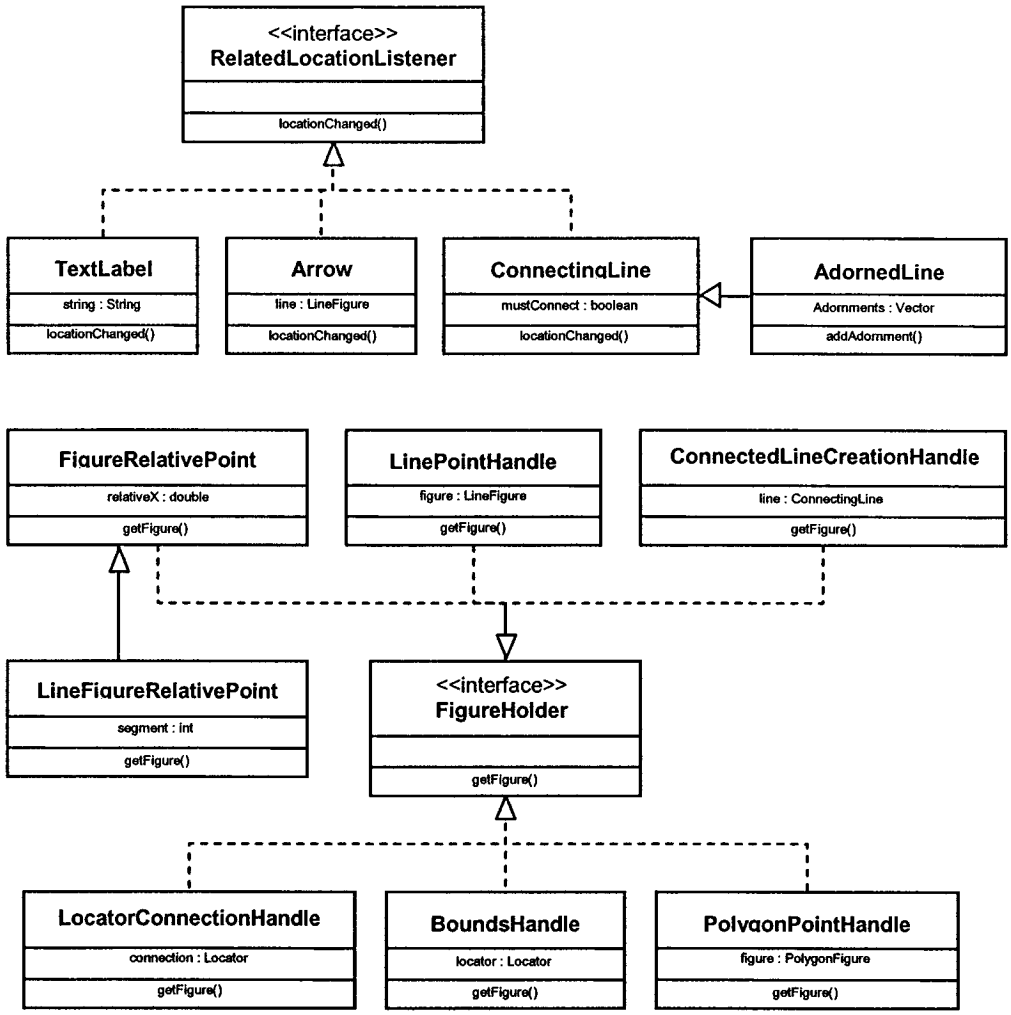


Figure 5.2-1 Class Diagrams for Elements Constraint

The only disadvantage of the constraint imposed on Lines is that the Figures are connected through their centers, not their edges, which are sometimes not desirable, especially when the Figure is not filled (hollow Figure). Since the tool supports drawing

individual lines, there is a workaround on the way to connect Figures on edges by not applying the constraint (see Figures 5.1-3 and 5.1-4):

- First draw the Figures;
- Draw lines away from Figures, thus no FigureHolder set for it;
- Then move either Figures or Lines to make them connected.

5.3. Graphical User Interface

The two LaTeX tools provide a GUI for the user to create and manipulate drawings on it. The Graphviz LaTeX tool is based on Dotty which is developed using a script language Lefty. This language supports limited widgets and graphics functions [53]. The Drawlets tool is built on Java which is more general than Lefty, and its powerful toolkit packages, such as the AWT class library and Swing components in JFC, allows building a user friendly graphical interface [59].

5.3.1. Graphviz LaTeX Tool

The tool supports no toolbars and all the operations have to be done through two menus and dialog boxes as shown in Figures 5.3-1 and 5.3-2.

Clicking the right mouse button on the blank area displays the general purpose menu (Left in Figure 5.3-1), which provides the functions related to the whole drawing. For example, clicking '`save latex file`' saves the graph in a TeX file; clicking '`do layout`' pops up a dialog box (Right in Figure 5.3-2) for user to select the desired layout

tool. Figure 5.3-3 shows the example snapshots before and after applying the '`do layout`' function for the two algorithms, *dot* and *neato*.

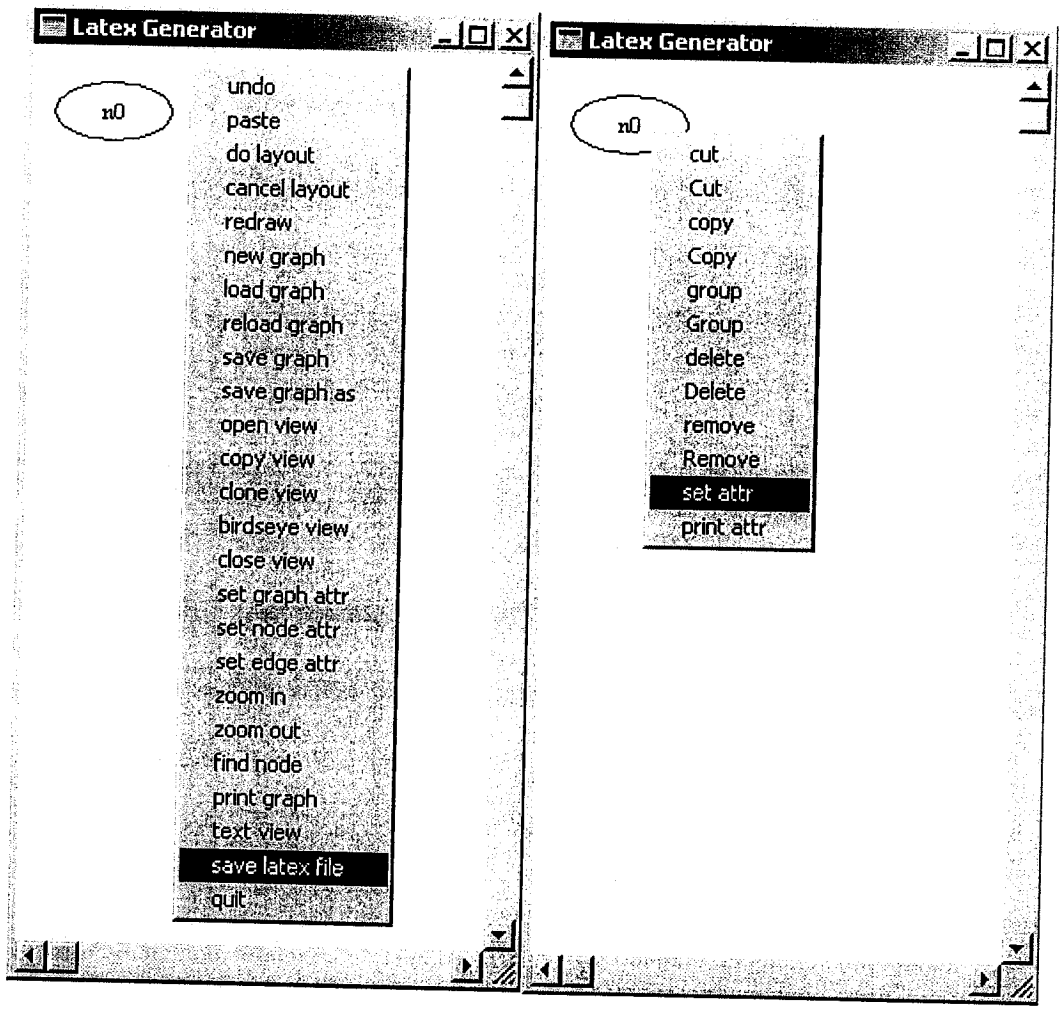


Figure 5.3-1 GUI of Graphviz LaTeX Tool and its Supported Menus

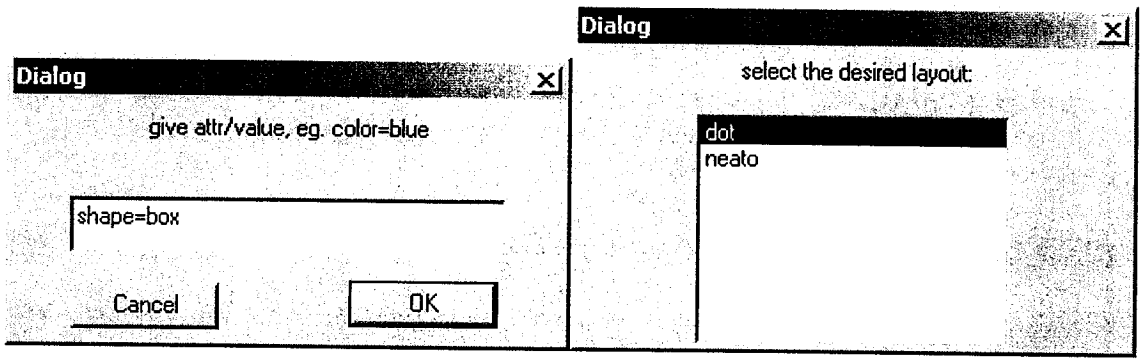


Figure 5.3-2 Dialog Boxes Supported by Graphviz LaTeX Tool

A node is drawn by clicking the left mouse button on the drawing area. Ellipse is the default node shape and it can be changed by '`set node attr`' item on the general purpose menu. Clicking on it pops up a dialog box (Left in Figure 5.3-2); typing `shape=box` in the text field, then OK, allows drawing box directly.

Clicking the right mouse button over a node brings up a menu for nodes modification. For example '`set attr`' allow changing node size and its other attributes. All the functions are described in detail in [44] and will not be discussed here.

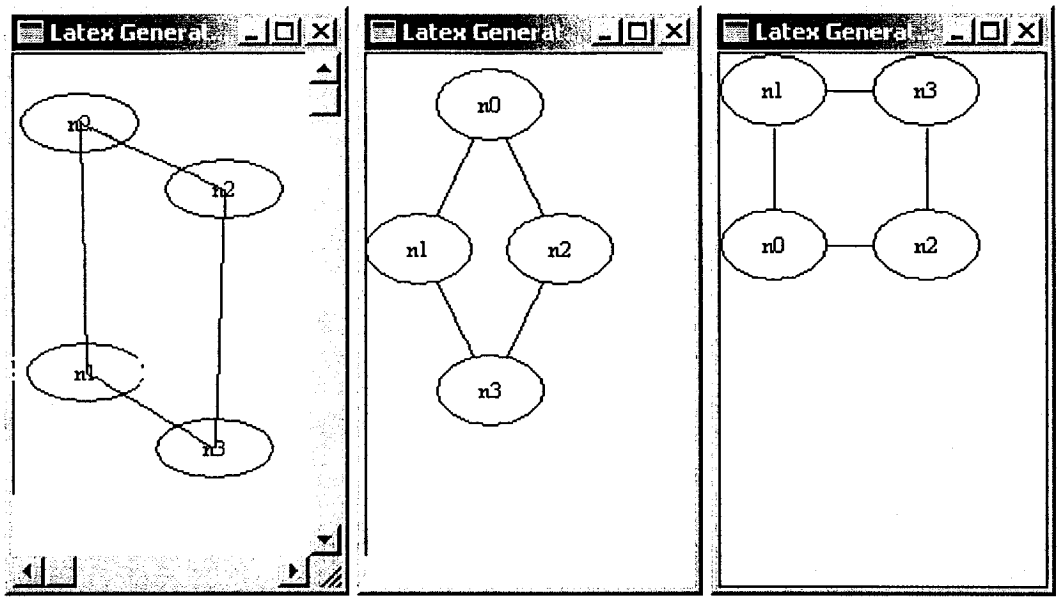


Figure 5.3-3 Layout Effects on Sample Graph (Middle *dot* and Right *neato*)

5.3.2. Drawlets LaTeX Tool

Figure 5.3-4 is a snapshot of the GUI of Drawlets LaTeX Tool with some explanations on the usage of icons. Those without examples are self-explanatory.

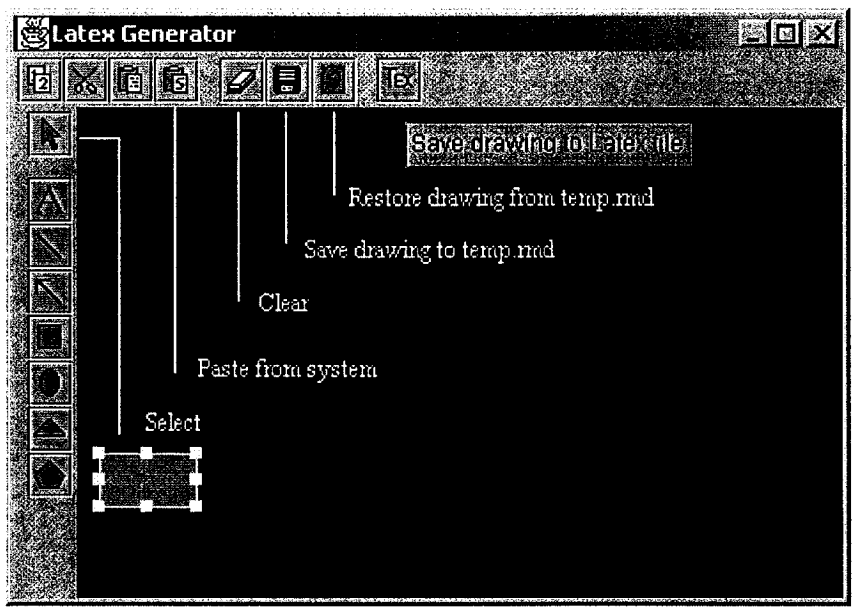


Figure 5.3-4 GUI of Drawlets LaTeX Tool

It is obvious that there is a corresponding icon on the GUI for each function and Figure supported by the tool. There is also a tooltip associated with each icon which can be displayed by moving the mouse over it, as shown for the **TEX** icon in Figure 5.3-4.

Clicking on different Figure icon, then left-click on the drawing area by holding and dragging mouse directly draws different Figure. For triangle and pentagon, they can be drawn by clicking the icon, then keeping left-clicking and moving the mouse on the drawing area. The size of a Figure can be changed by selecting it and dragging its editing box.

Chapter 6. Concluding Remarks

Two drawing frameworks were used and customized to be the graphical drawing tools generating LaTeX picture environment commands from the drawing directly. The implementations of the two LaTeX tools accomplished the desired objectives and will facilitate the usage of drawing figures in the LaTeX picture environment. The work highlights the advantages of using the LaTeX picture environment, that is, the TeX file produced containing the figures is self-contained, light-weight, standard, and thus portable. The following two sections summarize the work done and present suggestions for the future work.

6.1 Conclusions

The Graphviz LaTeX Tool is based on Graphviz's Dotty and keeps all its functions. The tool implemented adds some functions to transform the drawing in attributed format into LaTeX picture environment commands and saves them in a TeX file. Unlike Dotty, it supports both *dot* and *neato* layout algorithms tools based on the user's choice.

The Drawlets LaTeX Tool is built on top of the Drawlets framework starting with its example class `SimpleModelPanel` under the `examples.jfc` package. The tool uses the Drawlets framework as a library and no updates for the framework were done. The current project implements some classes and functions to save the drawings in a TeX file in LaTeX picture environment commands format.

Both LaTeX Tools support the basic LaTeX picture commands such as `\put`, `\oval`, `\line`, `\vector`, and `\makebox`. They both support the basic operations to create and manipulate drawings (like *select*, *cut*, *copy*, *paste*) and to resize graphs. They can save the drawings to a file temporarily and retrieve them later for update. Besides these common features, they all support some other functions or features. For example, the Graphviz tool provides multiple views, and the Drawlets tool allows *copy* and *paste* between different applications. There are certain limitations for the tools, also: for example, they can not create large circles as wanted due to the constraint on the LaTeX picture environment.

Both tools provide a way to adjust automatically the positions and connections of drawing elements. The Graphviz LaTeX Tool supports two static layout algorithms, *dot* and *neato*, which update drawings and produce directed and undirected graphs, respectively, in a well organized format. But this also implies that the drawing styles are fixed by the layout algorithms and cannot be altered as user wishes. The Drawlets LaTeX Tool provides dynamic updating through the constraints imposed on the connecting lines, which allow the drawing to be manipulated easily. The user has the flexibility to draw the graphs desired.

The two LaTeX tools provide a GUI for a user to create and update drawings with the supported functions. The Graphviz LaTeX tool supports no toolbars and all the operations have to be done through menus and dialog boxes, whereas the GUI of the

Drawlets tool is more user friendly; it has tool bars containing each function and Figure supported by the tool, together with the mouse drag event.

6.2 Suggestions for Future Work

The purpose of this project is to investigate ways to build a WYSIWYG tool for the LaTeX picture environment within the existing drawing frameworks, but not to focus on a fully functional tool. Apparently, there are more things that could be done to improve the tools developed.

1. Both tools were developed and tested under Windows system, since the two frameworks are platform independent, the tools are supposed to be work with some updates under Unix system etc.
2. Both frameworks are evolving, especially for Graphviz; its version is now 1.16 comparing 1.10 when the project started. The tools need to be updated accordingly for the new features added. For example, it seems possible to add *ftp* layout process to the Graphviz tool, as both *neato* and *ftp* are the tools for 'spring' layout algorithms.
3. For Drawlets tool, try to connect node by edges and handle arrow lines in Save and Restore functions, these may need changes on the framework.
4. Drawlets's design pattern allows extending the layout algorithms feasible by adding an algorithm package, it is up to user to use it or not.
5. It is desirable to add grid onto drawing interface, and this can be done through creating a new implementation of `DrawingCanvas` (see Patterns for Drawlets in [36]).

References

1. Leslie Lamport. LaTeX: A Document Preparation System, Addison-Wesley, 1986.

2. Hypertext Help with LaTeX
   http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/teTeX/latex/latex2e-html/ltx-2.html, March 26, 2005.

3. Urs Oswald. Graphics in LATEX2ε, March 10, 2003.
   http://www.ursoswald.ch/LaTeXGraphics/overview/latexgraphics.pdf, March 26, 2005.

4. Keith Reckdahl. Using Imported Graphics in LATEX2ε, Version 2.0, December 15, 1997. http://www.ctan.org/tex-archive/info/epslatex.pdf, March 26, 2005.

5. aiSee – Graph Visualization, http://www.absint.com/aisee/, March 26, 2005.

6. Graphviz - Graph Visualization Software, http://www.graphviz.org/, March 26, 2005.

7. Ipe Home Page, http://ipe.compgeom.org/, March 26, 2005.

8. Drawing Graphs with VGJ
   http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html, March 26, 2005.

9. jfig Home Page.
   http://tech-www.informatik.uni-hamburg.de/applets/javafig/index.html, March 26, 2005.

10. JasTex, http://www.liafa.jussieu.fr/~gastin/JasTeX/JasTeX.html, March 26, 2005.

11. jPicEdt for LaTeX, http://jpicedt.sourceforge.net/, March 26, 2005.

12. MiKTeX, http://www.miktex.org/, March 26, 2005.

13. CAD, Drawing & Painting Tools, http://gd.tuwien.ac.at:8050/E/2/, March 26, 2005.

14. Visualization of Compiler Graphs
http://rw4.cs.uni-sb.de/users/sander/html/gsvcg1.html, March 26, 2005.

15. G. Sander. Graph Layout through the VCG Tool. Technical Report A03-94, Universität des Saarlandes, FB 14 Informatik, 1994.
http://rw4.cs.uni-sb.de/~sander/html/gspapers.html#graphlayout, March 26, 2005.

16. aiSee Graph Visualization User Documentation – Windows Version 2.00. AbsInt Angewandte Informatik GmbH, September 1, 2000.
http://www.absint.com/aisee/manual/windows/, March 26, 2005.

17. Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A Technique for Drawing Directed Graphs. IEEE Trans. Software Engineering, 19(3):214–230, May 1993.

18. Emden R. Gansner, Eleftherios Koutsofios and Stephen C. North. Drawing graphs with dot. Technical Report, AT&T Bell Laboratories, Murray Hill NJ, February 2002.

19. E. R. Gansner. Drawing graphs with Graphviz, Graphviz Drawing Library Manual, November 22, 2004. http://www.graphviz.org/cvs/doc/libguide/libguide.pdf, March 26, 2005.

20. Stephen C. North. Drawing graphs with NEATO, Technical Report, AT&T Bell Laboratories, Murray Hill NJ, April 10, 2002.
http://www.graphviz.org/Documentation/neatoguide.pdf, March 26, 2005.

21. K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. IEEE Trans. Systems, Man and Cybernetics, SMC-11(2):109–125, February 1981.

22. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. Information Processing Letters, 31(1):7–15, April 1989.s with NEATO.

23. J. Kruskal and J. Seery. Designing network diagrams. In Proc. First General Conf. on Social Graphics, pages 22–50, 1980.

24. J. Cohen. Drawing graphs to convey proximity: an incremental arrangement method. ACM Transactions on Computer-Human Interaction, 4(11):197–229, 1987.

25. Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-directed Placement. Software – Practice and Experience, 21(11):1129–1164, November 1991.

26. G. Wills. Nicheworks - interactive visualization of very large graphs. In G. DiBattista, editor, Symposium on Graph Drawing GD'97, volume 1353 of Lecture Notes in Computer Science, pages 403–414, 1997.

27. Janet Six and Ioannis Tollis. Circular drawings of biconnected graphs. In Proc. ALENEX 99, pages 57–73, 1999.

28. Janet Six and Ioannis Tollis. A framework for circular drawings of networks. In Proc. Symp. Graph Drawing GD'99, volume 1731 of Lecture Notes in Computer Science, pages 107–116. Springer-Verlag, 2000.

29. M. Kaufmann and R. Wiese. Maintaining the mental map for circular drawings. In M. Goodrich, editor, Proc. Symp. Graph Drawing GD'02, volume 2528 of Lecture Notes in Computer Science, pages 12–22.

30. Stephen North. Incremental Layout in DynaDAG. *In Proc. Graph Drawing '95, volume 1027 of Lecture Notes in Computer Science*, pages 409-418. Springer- Verlag, 1996.

31. Stephen C. North and Gordon Woodhull. On-line Hierarchical Graph Drawing. AT&T Labs – Research, Florham Park, New Jersey.

http://www.graphviz.org/Documentation/NW01.pdf, March 26, 2005.

32. Otfried Cheong. The Ipe manual. November 22, 2004. http://ipe.compgeom.org/manual.pdf, March 26, 2005.

33. Graphlet, The GML File Format, http://infosun.fmi.uni-passau.de/Graphlet/GML/, March 26, 2005.

34. Visualizing Graphs with Java (VGJ) User Manual http://www.eng.auburn.edu/department/cse/research/graph_drawing/manual/vgj_manual.html, March 26, 2005.

35. GasTex: Graphs and Automata Simplified in TeX http://www.liafa.jussieu.fr/~gastin/gastex/gastex.html, March 26, 2005.

36. RoleModel Software. Drawlets Home Page. http://www.rolemodelsoftware.com/drawlets, March 26, 2005.

37. GEF Project Home Page, http://gef.tigris.org/, March 26, 2005.

38. HotDraw Home Page http://st-www.cs.uiuc.edu/users/brant/HotDraw/HotDraw.html, March 26, 2005.

39. Ralph E. Johnson. Documenting Frameworks Using Patterns. ACM SIGPLAN Notices, 27(10):63–76, 1992.

40. Douglas Kirk. Identifying the problems of large scale reuse: A personal case study. Department of Computer Science, University of Strath, GLASGOW, UK. http://www.cis.strath.ac.uk/~efocs/home/Research-Reports/EFoCS-41-2001.pdf, March 26, 2005.

41. Precision Graphics Markup Language (PGML). World Wide Web Consortium Note, April 10, 1998. http://www.w3.org/TR/1998/NOTE-PGML-19980410, March 26, 2005.

42. E.R. Gansner and S.C. North. An open graph visualization system and its applications to software engineering. Software – Practice and Experience, 30:1203–1233, 2000.

43. Eleftherios Koutsofios and Steve North. Applications of Graph Visualization. In Proceedings of Graphics Interface, pages 235–245, May 1994.

44. Eleftherios Koutsofios, Stephen C. North. Editing graphs with dotty. Techical Report, 96b (06-24-96).
http://www.graphviz.org/Documentation/dottyguide.pdf, March 26, 2005.

45. J. Ellson and S. North. TclDG - a Tcl extension for dynamic graphs. In Proc. 4th USENIX Tcl/Tk Workshop, pages 37–48, 1996.

46. W. Lee, N. Barghouti, and J. Mocenigo. Grappa: A graph package in Java. *Proc. Symp. Graph Drawing*, GD '97, September 1997.

47. Kanth Miriyala, Scot W. Hornick, and Roberto Tamassia. An Incremental Approach to Aesthetic Graph Layout. In Proc. Sixth International Workshop on Computer-Aided Software Engineering, pages 297{308. IEEE Computer Society, July 1993.

48. Shuyun Xu, Yu Li and Shiyong Lu, "ERDraw: An XML-based ER-diagram Drawing and Translation Tool", in Proc. of the 18th International Conference on Computers and Their Applications (CATA'2003), pp. 143--146, Honolulu, Hawaii, USA, March, 2003.

49. ArgoUML Project Home Page, http://argouml.tigris.org/, March 26, 2005.

50. Serge Demeyer, Tom Mens and Michel Wermelinger. Towards a Software Evolution Benchmark. *Proc. Int. Workshop on Principles of Software Evolution*, Vienna, Austria, September 2001.

51. M. Skoglund and P. Runeson. A Case Study on Regression Test Suite Maintenance in System Evolution. ICSM'04 - The 20th IEEE International Conference on Software Maintenance, pp. 438-442, 2004.

52. M. Skoglund and P. Runeson. A Case Study on Testware Maintenance and Change Strategies in System Evolution. Technical Report 04-007, Department of Computer and Systems Sciences (DSV), 2004.

53. Eleftherios Koutsofios. Editing Pictures with lefty. Technical Report, 96b (06-24-96). http://www.graphviz.org/Documentation/leftyguide.pdf, March 26, 2005.

54. How to draw Hasse diagrams (lattice diagrams) in LaTeX http://osl.iu.edu/~tveldhui/misc/hasse.html, March 26, 2005.

55. James W. Cooper. The Design Patterns Java Companion. *Design Patterns Series*. Addison-Wesley, October1998.

56. John M. Brant. HotDraw. Master Thesis, University of Illinois at Urbana-Champaign, 1995.

57. Visual Paradigm for UML, http://www.visual-paradigm.com, March 26, 2005.

58. Ken Auer. Patterns for Building an Unusually Adaptable Java Framework. *RoleModel Software, Inc.* http://www.rolemodelsoftware.com/moreAboutUs/publications/articles/javaextend.php, March 26, 2005.

59. Cay S. Horstmann, Gary Cornell. Core Java 2 Volume I-Fundamentals. The Sun Microsystems Press, Java Series, 2001.

Appendix A. Installation and Setup

MiKTeX

1. Install the complete package of MiKTeX by running *setup-2.4.1445* under Disk:\MiKTeX or download the latest version from web:

   http://www.miktex.org/setup.html
2. Add Drive:\texmf\miktex\bin to system variable PATH.
3. Two other pieces of shareware that are needed for a complete system are GhostScript, which is a program for PostScript documents, and GSView, which uses GhostScript to view PS documents on your computer. GSview is a graphical interface for GhostScript under MS-Windows, OS/2 and Unix. GhostScript is an interpreter for the PostScript page description language used by laser printers. Install the two by running *gsv45w32* and *gs813w32* under Disk:\gsview or download them from:

   http://www.cs.wisc.edu/~ghost/
4. From Windows Explorer -> Tools -> Folder Options.. -> File Types -> Change..., set DVI file extension to allow opening DVI file by DVI viewer by default.
5. How does MiKTeX work? Put all of the text, including the LaTeX formatting commands into a TeX file. The program "*latex*" reads this file and produces a device independent file, a DVI file, which can be viewed or printed by a DVI viewer. MiKTeX includes a program "*dvips*" to turn the DVI file into a PostScript file. All of these programs are run from the MSDOS prompt.

Graphviz

1. Install Graphviz by running *graphviz-1.10* under Disk:\Graphviz or download the latest version 1.16 from web:

   http://www.graphviz.org/Download_windows.php

2. Remove circle on the edge by opening file under Drive:\Program Files\ATT\ Graphviz\bin\ *dotty.lefty*, and set `'edgehandles' = 0.`

3. Install Graphviz LaTeX tool by unzipping *Latex Generator.zip* under Disk:\Graphviz, which will store the tool under Drive:\Latex Generator.

4. Start Graphviz tool using batch file "*latex*" under Drive:\Latex Generator (all the batch files are stored under it), draw the graph and save it as *test.tex* under the same directory; run batch file "*test*" and then open *test.dvi* file generated to check the result; run "dvips" will produce PS file.

Drawlets

1. Install JDK1.3 or JBuilder4 and add Drive:\jdk1.3\bin or Drive:\jbuilder4\jdk1.3 \bin to system variable PATH.

2. Install Drawlets LaTeX tool by unzipping *LatexGen.zip* under Disk:\Drawlets, which will store the tool under Drive:\LatexGen.

3. Start the tool in two ways: i) run batch file "*java_env*" under Drive:\LatexGen (all the batch files are stored under it) to check and set JDK path, then run "*run_latex*" to start the application; ii) open *LatexGen.jpx* project under Drive:\LatexGen in JBuilder and run it.

4. Draw the graph and save it as *test.tex* under the same directory; run batch file "*test*" and then open test.dvi generated to check the result; run "dvips" will produce PS file.
5. Optionally, install Drawlets framework source code by unzipping *drawlet2.0.zip* under Disk:\Drawlets or download the latest version from: http://www.rolemodelsoftware.com/drawlets/index.php

Appendix B. LaTeX Picture Environment Commands

Command	Syntax	Description
\circle	\circle[*]{diameter}	Produces a circle of the specified diameter. If the *-form of the command is used, LaTeX draws a solid circle. The maximum size of a filled circle is 15 points, and 40 for a hollow circle.
\frame	\frame{ ... }	Puts a rectangular frame around the object specified in the argument.
\framebox	\framebox(width,height)[position]{...}	Puts a frame around the outside of the box that it creates.
\dashbox	\dashbox{dash length}(width,height)[pos]{ ... }	Like \framebox but has an extra argument which specifies the width of each dash.
\line	\line(x-slope,y-slope){length}	Draws a line of the specified length and slope. x-slope and y-slope must both be integers from -6 to +6 inclusive without common divisor larger than one.
\linethickness	\linethickness{dimension}	Declares the thickness of horizontal and vertical lines in a picture environment to be dimension, which must be a positive length.
\makebox	\makebox(width,height)[position]{ ... }	Creates a box to contain the specified text.
\mbox	\mbox{text}	Creates a box just wide enough to hold the text in its argument.
\multiput	\multiput(x coord,y coord)(delta x,delta y){number of copies}{object}	Used when you are putting the same object in a regular pattern across a picture.
\oval	\oval(width,height)[portion]	Produces a rectangle with rounded corners. The optional argument, portion, allows you to select part of the oval.
\put	\put(x-coord,y-coord){object}	places the object specified by the mandatory argument at the given coordinates.
\setlength	\setlength{len-cmd}{len}	Used to set the value of a length command
\shortstack	\shortstack[position]{ ... \\ ... \\ ... }	Produces a stack of objects. The valid positions are: r, l and c.
\unitlength	\unitlength{len}	Defines the units used in the Picture Environment. The default value is 1 point (approximately 1/72.27 inch).
\vector	\vector(x-slope,y-slope){length}	Draws an arrow of the specified length and slope with the arrow head at the opposite end of the line from the reference point. x-slope and y-slope must both be integers from -4 to +4 inclusive.

Appendix C-1. Development of Shells Specified in GDL

```
graph: {
       title: "shells"
       splines: yes
       layoutalgorithm: minbackward
       layout_nearfactor: 0
       layout_downfactor: 100
       layout_upfactor: 100

       // First the time scale

       node.height: 26
       node.width: 60
       node.borderwidth: 0
       edge.linestyle: dashed

       node: { title: "1972" vertical_order: 1 horizontal_order: 1}
       node: { title: "1976" vertical_order: 2 horizontal_order: 1 }
       node: { title: "1978" vertical_order: 3 }
       node: { title: "1980" vertical_order: 4 }
       node: { title: "1982" vertical_order: 5 horizontal_order: 1 }
       node: { title: "1984" vertical_order: 6 }
       node: { title: "1986" vertical_order: 7 }
       node: { title: "1988" vertical_order: 8 }
       node: { title: "1990" vertical_order: 9 }
       node: { title: "future" vertical_order: 10 horizontal_order: 1 }

       edge: { source: "1972" target: "1976" }
       edge: { source: "1976" target: "1978" }
       edge: { source: "1978" target: "1980" }
       edge: { source: "1980" target: "1982" }
       edge: { source: "1982" target: "1984" }
       edge: { source: "1984" target: "1986" }
       edge: { source: "1986" target: "1988" }
       edge: { source: "1988" target: "1990" }
       edge: { source: "1990" target: "future" }

       // We need some invisible edge to make the graph fully connected.
       // Otherwise, the horizontal_order attribute would not work.

       edge: { source: "ksh-i" target: "Perl" linestyle: invisible priority: 0 }
       edge: { source: "tcsh" target: "tcl" linestyle: invisible priority: 0 }
       nearedge: { source: "1988" target: "rc" linestyle: invisible }
       nearedge: { source: "rc" target: "Perl" linestyle: invisible }

       // Now the shells themselves
       // Note: the default value -1 means: no default

       node.height: -1
       node.width: -1
       node.borderwidth: 2
       edge.linestyle: solid

       node: { title: "Thompson" vertical_order: 1 horizontal_order: 2 }
       node: { title: "Mashey" vertical_order: 2 horizontal_order: 3 }
       node: { title: "Bourne" vertical_order: 2 horizontal_order: 2 }
       node: { title: "Formshell" vertical_order: 3 }
       node: { title: "csh" vertical_order: 3 shape: triangle }
       node: { title: "esh" vertical_order: 4 horizontal_order: 2 }
```

```
        node: { title: "vsh" vertical_order: 4 }
        node: { title: "ksh" vertical_order: 5 horizontal_order: 3 shape: ellipse
        }
        node: { title: "System-V" vertical_order: 5 horizontal_order: 5 }
        node: { title: "v9sh" vertical_order: 6 }
        node: { title: "tcsh" vertical_order: 6 shape: triangle }
        node: { title: "ksh-i" vertical_order: 7 shape: ellipse }
        node: { title: "KornShell" vertical_order: 8 shape: ellipse }
        node: { title: "Perl" vertical_order: 8 }
        node: { title: "rc" vertical_order: 8 }
        node: { title: "tcl" vertical_order: 9 shape: rhomb }
        node: { title: "Bash" vertical_order: 9 }
        node: { title: "POSIX" vertical_order: 10 horizontal_order: 3 }
        node: { title: "ksh-POSIX" vertical_order: 10 horizontal_order: 2 shape:
        ellipse }

        edge: { source: "Thompson" target: "Mashey" }
        edge: { source: "Thompson" target: "Bourne" }
        edge: { source: "Thompson" target: "csh" horizontal_order: 4 }
        edge: { source: "Bourne" target: "ksh" }
        edge: { source: "Bourne" target: "esh" }
        edge: { source: "Bourne" target: "vsh" }
        edge: { source: "Bourne" target: "System-V" }
        edge: { source: "Bourne" target: "v9sh" }
        edge: { source: "Bourne" target: "Formshell" }
        edge: { source: "Bourne" target: "Bash" }
        edge: { source: "csh" target: "tcsh" }
        edge: { source: "csh" target: "ksh" }
        edge: { source: "Formshell" target: "ksh" horizontal_order: 4 }
        edge: { source: "esh" target: "ksh" }
        edge: { source: "vsh" target: "ksh" }
        edge: { source: "ksh" target: "ksh-i" }
        edge: { source: "System-V" target: "POSIX" }
        edge: { source: "v9sh" target: "rc" }
        edge: { source: "ksh-i" target: "KornShell" }
        edge: { source: "ksh-i" target: "Bash" }
        edge: { source: "KornShell" target: "Bash" }
        edge: { source: "KornShell" target: "POSIX" }
        edge: { source: "KornShell" target: "ksh-POSIX" }
}
```

Appendix C-2. Development of Shells Specified in DOT

```
digraph shells {
        size="7,8";
        node [fontsize=24, shape = plaintext];

        1972 -> 1976;
        1976 -> 1978;
        1978 -> 1980;
        1980 -> 1982;
        1982 -> 1984;
        1984 -> 1986;
        1986 -> 1988;
        1988 -> 1990;
        1990 -> future;

        node [fontsize=20, shape = box];
        { rank=same;  1976 Mashey Bourne; }
        { rank=same;  1978 Formshell csh; }
        { rank=same;  1980 esh vsh; }
        { rank=same;  1982 ksh "System-V"; }
        { rank=same;  1984 v9sh tcsh; }
        { rank=same;  1986 "ksh-i"; }
        { rank=same;  1988 KornShell Perl rc; }
        { rank=same;  1990 tcl Bash; }
        { rank=same;  "future" POSIX "ksh-POSIX"; }

        Thompson -> Mashey;
        Thompson -> Bourne;
        Thompson -> csh;
        csh -> tcsh;
        Bourne -> ksh;
        Bourne -> esh;
        Bourne -> vsh;
        Bourne -> "System-V";
        Bourne -> v9sh;
        v9sh -> rc;
        Bourne -> Bash;
        "ksh-i" -> Bash;
        KornShell -> Bash;
        esh -> ksh;
        vsh -> ksh;
        Formshell -> ksh;
        csh -> ksh;
        KornShell -> POSIX;
        "System-V" -> POSIX;
        ksh -> "ksh-i";
        "ksh-i" -> KornShell;
        KornShell -> "ksh-POSIX";
        Bourne -> Formshell;

        edge [style=invis];
        1984 -> v9sh -> tcsh ;
        1988 -> rc -> KornShell;
        Formshell -> csh;
        KornShell -> Perl;
}
```

Appendix C-3. Sample Drawing of VGJ Specified in GML

```
graph [
   directed 1
   node [
      id 0
      label "Box"
      graphics [
         Image [
            Type ""
            Location ""
         ]
         center [
            x -133.0
            y 64.0
            z 0.0
         ]
         width 70.0
         height 42.0
         depth 23.0
      ]
      vgj [
         labelPosition "below"
         shape "Rectangle"
      ]
   ]
   node [
      id 1
      label "Oval"
      graphics [
         Image [
            Type ""
            Location ""
         ]
         center [
            x 33.0
            y 65.0
            z 0.0
         ]
         width 63.0
         height 37.0
         depth 20.0
      ]
      vgj [
         labelPosition "below"
         shape "Oval"
      ]
   ]
   edge [
      linestyle "solid"
      label ""
      source 0
      target 1
   ]
]
```

Appendix D. LaTeX Tools Outputs in TeX Format

TeX output for Figure 2.3-3

```
%Created by jPicEdt 1.x
%Standard LaTeX format (emulated lines)
%Tue Feb 01 11:19:48 EST 2005
\unitlength 1mm
\begin{picture}(35.00,80.00)(0,0)

\linethickness{0.15mm}
%Polygon 0 0(0.00,0.00)(10.00,60.00)
\multiput(0.00,0.00)(0.12,0.72){83}{\line(0,1){0.72}}
%End Polygon

\linethickness{0.15mm}
%Ellipse 0 0(27.50,72.50)(15.00)(15.00)  arcStart=0.0 arcExtent=0.0
\put(34.98,72.01){\line(0,1){0.98}}
\multiput(34.86,73.96)(0.13,-0.97){1}{\line(0,-1){0.97}}
\multiput(34.60,74.91)(0.13,-0.47){2}{\line(0,-1){0.47}}
\multiput(34.23,75.82)(0.13,-0.30){3}{\line(0,-1){0.30}}
\multiput(33.74,76.67)(0.12,-0.21){4}{\line(0,-1){0.21}}
\multiput(33.14,77.45)(0.12,-0.16){5}{\line(0,-1){0.16}}
\multiput(32.45,78.14)(0.12,-0.12){6}{\line(1,0){0.12}}
\multiput(31.67,78.74)(0.16,-0.12){5}{\line(1,0){0.16}}
\multiput(30.82,79.23)(0.21,-0.12){4}{\line(1,0){0.21}}
\multiput(29.91,79.60)(0.30,-0.13){3}{\line(1,0){0.30}}
\multiput(28.96,79.86)(0.47,-0.13){2}{\line(1,0){0.47}}
\multiput(27.99,79.98)(0.97,-0.13){1}{\line(1,0){0.97}}
\put(27.01,79.98){\line(1,0){0.98}}
\multiput(26.04,79.86)(0.97,0.13){1}{\line(1,0){0.97}}
\multiput(25.09,79.60)(0.47,0.13){2}{\line(1,0){0.47}}
\multiput(24.18,79.23)(0.30,0.13){3}{\line(1,0){0.30}}
\multiput(23.33,78.74)(0.21,0.12){4}{\line(1,0){0.21}}
\multiput(22.55,78.14)(0.16,0.12){5}{\line(1,0){0.16}}
\multiput(21.86,77.45)(0.12,0.12){6}{\line(1,0){0.12}}
\multiput(21.26,76.67)(0.12,0.16){5}{\line(0,1){0.16}}
\multiput(20.77,75.82)(0.12,0.21){4}{\line(0,1){0.21}}
\multiput(20.40,74.91)(0.13,0.30){3}{\line(0,1){0.30}}
\multiput(20.14,73.96)(0.13,0.47){2}{\line(0,1){0.47}}
\multiput(20.02,72.99)(0.13,0.97){1}{\line(0,1){0.97}}
\put(20.02,72.01){\line(0,1){0.98}}
\multiput(20.02,72.01)(0.13,-0.97){1}{\line(0,-1){0.97}}
\multiput(20.14,71.04)(0.13,-0.47){2}{\line(0,-1){0.47}}
\multiput(20.40,70.09)(0.13,-0.30){3}{\line(0,-1){0.30}}
\multiput(20.77,69.18)(0.12,-0.21){4}{\line(0,-1){0.21}}
\multiput(21.26,68.33)(0.12,-0.16){5}{\line(0,-1){0.16}}
\multiput(21.86,67.55)(0.12,-0.12){6}{\line(1,0){0.12}}
\multiput(22.55,66.86)(0.16,-0.12){5}{\line(1,0){0.16}}
\multiput(23.33,66.26)(0.21,-0.12){4}{\line(1,0){0.21}}
\multiput(24.18,65.77)(0.30,-0.13){3}{\line(1,0){0.30}}
\multiput(25.09,65.40)(0.47,-0.13){2}{\line(1,0){0.47}}
\multiput(26.04,65.14)(0.97,-0.13){1}{\line(1,0){0.97}}
\put(27.01,65.02){\line(1,0){0.98}}
\multiput(27.99,65.02)(0.97,0.13){1}{\line(1,0){0.97}}
\multiput(28.96,65.14)(0.47,0.13){2}{\line(1,0){0.47}}
\multiput(29.91,65.40)(0.30,0.13){3}{\line(1,0){0.30}}
\multiput(30.82,65.77)(0.21,0.12){4}{\line(1,0){0.21}}
\multiput(31.67,66.26)(0.16,0.12){5}{\line(1,0){0.16}}
```

```
\multiput(32.45,66.86)(0.12,0.12){6}{\line(0,1){0.12}}
\multiput(33.14,67.55)(0.12,0.16){5}{\line(0,1){0.16}}
\multiput(33.74,68.33)(0.12,0.21){4}{\line(0,1){0.21}}
\multiput(34.23,69.18)(0.13,0.30){3}{\line(0,1){0.30}}
\multiput(34.60,70.09)(0.13,0.47){2}{\line(0,1){0.47}}
\multiput(34.86,71.04)(0.13,0.97){1}{\line(0,1){0.97}}
%End Ellipse

\linethickness{0.15mm}
%Ellipse 0 0(25.00,55.00)(10.00)(10.00)  arcStart=0.0 arcExtent=0.0
\put(25.00,55.00){\circle{10.00}}
%End Ellipse

\end{picture}
```

TeX output for Figure 3.4-1

```
\documentclass{article}
\begin{document}

\begin{picture}(172,216)
% draw node 0
\put(86,190){\circle{36}}
\put(86,190){\makebox(0,0){}}
% draw node 1
\put(32,118){\circle{36}}
\put(32,118){\makebox(0,0){}}
% draw node 2
\put(86,118){\circle{36}}
\put(86,118){\makebox(0,0){}}
% draw node 3
\put(140,118){\circle{36}}
\put(140,118){\makebox(0,0){}}
% draw node 4
\put(86,46){\circle{36}}
\put(86,46){\makebox(0,0){}}
% draw edges
\put(86,172){\line(0,-1){36}}
\put(75,175){\line(-3,-4){32.250000}}
\put(97,175){\line(3,-4){32.250000}}
\put(86,100){\line(0,-1){36}}
\put(129,103){\line(-3,-4){32.250000}}
\put(43,103){\line(3,-4){32.250000}}
\put(26,10){Figure 1: The lattice M3}
\end{picture}
\end{document}
```

TeX output for Figure 4.4-1

```
\documentclass{article}
\begin{document}

\begin{picture}(392,513)
% draw figure 0
\put(107,485){\line(0,-1){84}}
% draw figure 1
\put(107,485){\oval(12,12)}
```

```
% draw figure 2
\put(107,401){\line(0,-1){163}}
% draw figure 3
\put(107,401){\line(1,-1){87}}
% draw figure 4
\put(107,401){\oval(12,12)}
% draw figure 5
\put(107,238){\line(1,-1){87}}
% draw figure 6
\put(106,241){\line(1,-3){1}}
% draw figure 7
\put(107,238){\oval(12,12)}
% draw figure 8
\put(294,347){\line(0,-1){96}}
% draw figure 9
\put(194,151){\line(1,1){100}}
% draw figure 10
\put(294,251){\oval(12,12)}
% draw figure 11
\put(107,64){\line(1,1){87}}
% draw figure 12
\put(107,64){\oval(12,12)}
% draw figure 13
\put(194,151){\line(1,-1){95}}
% draw figure 14
\put(289,56){\oval(12,12)}
% draw figure 15
\put(194,247){\line(0,-1){96}}
% draw figure 16
\put(194,151){\oval(12,12)}
% draw figure 17
\put(194,314){\line(0,-1){67}}
% draw figure 18
\put(194,247){\line(1,1){100}}
% draw figure 19
\put(194,247){\oval(12,12)}
% draw figure 20
\put(194,314){\oval(12,12)}
% draw figure 21
\put(294,347){\oval(12,12)}
% draw figure 22
\put(58,496){\makebox(0,0){boolean algebra}}
% draw figure 23
\put(56,478){\makebox(0,0){(i)(m)(a)(b)}}
% draw figure 24
\put(57,462){\makebox(0,0){(o)(M)(d)}}
% draw figure 25
\put(58,419){\makebox(0,0){modular}}
% draw figure 26
\put(59,403){\makebox(0,0){ortholattice}}
% draw figure 27
\put(60,387){\makebox(0,0){(j)(m)(a)}}
% draw figure 28
\put(63,371){\makebox(0,0){(b)(o)(M)}}
% draw figure 29
\put(148,326){\makebox(0,0){ortholattice}}
% draw figure 30
\put(148,312){\makebox(0,0){(j)(m)(a)(b)(o)}}
% draw figure 31
\put(60,248){\makebox(0,0){lattice}}
% draw figure 32
\put(62,233){\makebox(0,0){(j)(m)(a)(M)}}
% draw figure 33
```

```
\put(59,264){\makebox(0,0){modular}}
% draw figure 34
\put(154,248){\makebox(0,0){lattice}}
% draw figure 35
\put(154,263){\makebox(0,0){bounded}}
% draw figure 36
\put(157,234){\makebox(0,0){(j)(m)(a)(b)}}
% draw figure 37
\put(197,9){\makebox(0,0){Figure 1: Some lattice varieties and their axioms}}
% draw figure 38
\put(339,373){\makebox(0,0){bounded distributive}}
% draw figure 39
\put(342,341){\makebox(0,0){(j)(m)(a)(b)(d)}}
% draw figure 40
\put(340,265){\makebox(0,0){distributive}}
% draw figure 41
\put(344,237){\makebox(0,0){(j)(m)(a)(d)}}
% draw figure 42
\put(244,147){\makebox(0,0){(j)(m)(a)}}
% draw figure 43
\put(333,73){\makebox(0,0){join semilattice}}
% draw figure 44
\put(329,54){\makebox(0,0){(j)}}
% draw figure 45
\put(69,78){\makebox(0,0){meet semilattice}}
% draw figure 46
\put(63,61){\makebox(0,0){(m)}}
% draw figure 47
\put(335,357){\makebox(0,0){lattice}}
% draw figure 48
\put(340,250){\makebox(0,0){lattice}}
% draw figure 49
\put(244,162){\makebox(0,0){lattice}}

\end{picture}
\end{document}
```

TeX output for Figure 5.1-1

```
\documentclass{article}
\begin{document}

\begin{picture}(478,586)
% draw node 10
\put(388,131){\circle*{36}}
\put(388,131){\makebox(0,0){}}
% draw node 1
\put(243,464){\circle{39.920000}}
\put(243,464){\circle{43.920000}}
\put(243,464){\makebox(0,0){}}
% draw node 11
\put(182.160000,236){\line(3,1){59.842623}}
\put(182.160000,236){\line(3,-1){59.842623}}
\put(303.840000,236){\line(-3,1){59.842623}}
\put(303.840000,236){\line(-3,-1){59.842623}}
\put(243,236){\makebox(0,0){Diamond}}
% draw node 2
\put(60.920000,331){\line(1,0){92.160000}}
\put(60.920000,367){\line(1,0){92.160000}}
\put(60.920000,331){\line(0,1){36}}
```

```
\put(153.080000,331){\line(0,1){36}}
\put(107,349){\makebox(0,0){Long Label}}
% draw node 3
\put(205.920000,311.920000){\line(1,0){74.160000}}
\put(205.920000,386.080000){\line(1,0){74.160000}}
\put(205.920000,311.920000){\line(0,1){74.160000}}
\put(280.080000,311.920000){\line(0,1){74.160000}}
\put(215.920000,311.920000){\line(-1,1){10}}
\put(270.080000,311.920000){\line(1,1){10}}
\put(205.920000,376.080000){\line(1,1){10}}
\put(280.080000,376.080000){\line(-1,1){10}}
\put(243,349){\makebox(0,0){Msquare}}
% draw node 4
\put(310.120000,331){\line(1,0){131.760000}}
\put(415.528000,367){\line(-1,0){79.056000}}
\put(310.120000,331){\line(3,4){27.365538}}
\put(441.880000,331){\line(-3,4){27.365538}}
\put(376,349){\makebox(0,0){Trapezium}}
% draw node 5
\put(385,236){\oval(54,36)}
\put(385,236){\makebox(0,0){Oval}}
% draw node 15
\put(243,560){\makebox(0,0){Plain Text}}
% draw node 6
\put(19.970000,116.780000){\line(1,0){102.060000}}
\put(19.970000,116.780000){\line(6,5){51.330176}}
\put(122.030000,116.780000){\line(-6,5){51.330176}}
\put(71,131){\makebox(0,0){Triangle}}
% draw node 7
\put(156.960000,113){\line(1,0){137.664000}}
\put(329.040000,149){\line(-1,0){137.664000}}
\put(156.960000,113){\line(1,1){36.440471}}
\put(329.040000,149){\line(-1,-1){36.440471}}
\put(243,131){\makebox(0,0){Parallelogram}}
% draw node 8
\put(71,46){\oval(141.840000,36)}
\put(71,46){\makebox(0,0){Test triangle edge}}
% draw node 9
\put(84,236){\circle{36}}
\put(84,236){\makebox(0,0){}}
% draw edges
\put(243,312){\line(0,-1){56}}
\put(262.500000,284){diamond}
\put(226,450){\vector(-4,-3){110.666667}}
\put(210.500000,414){box}
\put(243,442){\vector(0,-1){56}}
\put(262.500000,414){Msquare}
\put(243,542){\line(0,-1){56}}
\put(260.500000,514){double circle}
\put(264,456){\vector(4,-3){118.666667}}
\put(357.500000,414){trapezium}
\put(377,331){\line(1,-6){12.833333}}
\put(393,284){oval}
\put(71,64){\vector(0,1){53}}
\put(103,331){\line(-1,-5){15.400000}}
\put(105,284){circle}
\put(74,157){\vector(1,4){15.250000}}
\put(93,188){triangle}
\put(387,149){\vector(0,1){69}}
\put(398.500000,188){filled circle}
\put(243,149){\vector(0,1){67}}
\put(265.500000,188){parallelogram}
\put(101.500000,10){Figure 1. Sample Graph with Different Shapes and Arrows}
```

```
\end{picture}
\end{document}
```

TeX output for Figure 5.1-3

```
\documentclass{article}
\begin{document}

\begin{picture}(379,364)
% draw figure 0
\put(58,308){\line(1,0){148}}
% draw figure 1
\put(26,329){\line(1,0){65}}
\put(26,329){\line(0,-1){42}}
\put(26,287){\line(1,0){65}}
\put(91,329){\line(0,-1){42}}
% draw figure 2
\put(201,199){\vector(0,1){85}}
% draw figure 3
\put(162,331){\line(1,0){89}}
\put(162,331){\line(0,-1){47}}
\put(162,284){\line(1,0){89}}
\put(251,331){\line(0,-1){47}}
% draw figure 4
\put(316.5,106.5){\oval(69,65)}
% draw figure 5
\put(28,157){\line(0,-1){77}}
\put(28,80){\line(1,0){77}}
\put(105,80){\line(-1,1){77}}
% draw figure 6
\put(205.5,198.5){\oval(103,51)}
% draw figure 7
\put(65,181){\line(0,-1){61}}
% draw figure 8
\put(189.0,99.0){\oval(26,26)}
% draw figure 9
\put(189.0,99.0){\oval(38,36)}
% draw figure 10
\put(316.0,107.0){\oval(38,36)}
% draw figure 11
\put(339,319){\line(-6,-5){57}}
\put(284,271){\line(2,-5){26}}
\put(311,206){\line(6,1){42}}
\put(353,213){\line(2,3){26}}
\put(379,252){\line(-3,5){40}}
% draw figure 12
\put(209.0,32.5){\makebox(0,0){The special function supported by the tool is to
copy text}}
% draw figure 13
\put(209.0,8.5){\makebox(0,0){from other application and paste it to the
current drawing}}
% draw figure 14
\put(58,249){\line(-3,-5){40}}
\put(18,182){\line(1,0){96}}
\put(114,182){\line(-5,6){55}}
% draw figure 15
\put(141.0,345.5){\makebox(0,0){connecting line by centers}}
% draw figure 16
\put(183.0,261.5){\makebox(0,0){connecting line arrow end by edge}}
% draw figure 17
```

```
\put(315.5,157.5){\makebox(0,0){rounded square}}
% draw figure 18
\put(194.0,127.5){\makebox(0,0){double circle}}
% draw figure 19
\put(118.0,155.5){\makebox(0,0){line without locator}}

\end{picture}
\end{document}
```

Appendix E-1. Pseudo Code for Graphviz Tool

Pseudo Codes for `savelatexfile` Function

```
procedure savelatexfile(gt, name, type)
    # define local parameters for graph, edge, node, slope etc.
    graph, edge, node, bb, slope, …;

    # Tex file to save latex commands
    create and open a file;

    # add LatexCode Prefix to tex file
    output (file, \documentclass{article}) ;
    output (file, begin{document}) ;

    # process graph by iterating through nodes to obtain its boundary box bb
    for i=0 to last_node do
        graph.x = max(node.x);
        graph.y = max(node.y);
    end
    graph.bb = (0, 0, graph.x, graph.y);

    # add bb to tex file
    output(file, bb);

    # iterate through nodes and call different savenode functions
    # based on the node shape
    for i=0 to last_node do
        # save node to tex file
        savenode[node[i].shape](file, gt, node);

        # save node label with label or name
        if node[i].label == true then
            output(file, node[i].label);
        else
            output(file, node[i].name);
        end if
    end

    # iterate through edges and save them to tex file
    for i=0 to last_edge do
        # if edge has points, then loop through the points
        if edge[i].points == true then
            for j=0 to last_point do
                # loop through edge points by doing nothing
            end

            # decide the two ending points for different line and vector
            if edge[i].sp == true then
                set the two ending points for backward vector;
            else if edge[i].ep == true then
                set the two ending points for forward vector;
            # process none arrow edge
            else
                set the two ending points for line;
            end if
        end if

        # decide max number for latex line and vector
        # if none arrow line, else vector
        if edge[i].dir == false then
```

```
            maxmum = 6;
            line = line;
        else
            maxnum = 4;
            line = vector;
        end if

        # define delta x and delta y, slope and adjusted newslope for edge[i]
        dx, dy, slope [], newslope [];

        if dx > 0 and dy > 0 then
            // remove common divisors and set new line slope
            // same for the rest slope line, won't repeat.
            slope = removeCommonDivisor(dx,dy);
            newslope = adjustSlope(slope[0],slope[1],dx, dy);
            # line that x increases while y increases
            output(file, slope line);
        else if dx > 0 and dy < 0 then
            process slope line and get new slope;
            # line that x increases while y decreases
            output(file, slope line);
        else if dx > 0 and dy = 0 then
            output(file, upward vertical line);
        else if dx < 0 and dy > 0 then
            process slope line and get new slope;
            # line that x decreases while y increases
            output(file, slope line);
        else if dx < 0 and dy < 0 then
            process slope line and get new slope;
            # line that x decreases while y decreases
            output(file, slope line);
        else if dx < 0 and dy = 0 then
            output(file, downward vertical line);
        else if dx = 0 and dy > 0 then
            output(file, right horizontal line);
        else if dx = 0 and dy < 0 then
            output(file, left horizontal line);
        else
            # do nothing
        end if

        # write out edge label
        output(file, edge[i].label);
    end

    # save graph label
    if graph.lp == true then
        set graph lp;
        output(file, graph label);
    end if

    // save latex Suffix
    output (file, \end{picture}) ;
    output (file, \end{document})
    close(file);
end
```

Pseudo Codes for Saving Nodes in Different Shapes

```
# function to process and save box node
```

```
procedure savenode.box(file, gt, node)
    # define the two x, y coordinates of the line
    output(file, first horizontal line);
    output(file, second horizontal line);
    output(file, first vertical line);
    output(file, second vertical line);
end

# function to process and save circle node
procedure savenode.circle(file, gt, node)
    output(file, circle line);
end

# function to process and save ellipse node
procedure savenode.ellipse(file, gt, node)
    output(file, ellipse line);
end

# function to process and save triangle node
procedure savenode.triangle(file, gt, node)
    # define the two x, y coordinates of line, slope and newslop
    maxnum = 6;

    # support triangle with two orientation
    if node.orientation != - 90 then
        set the two x, y coordinates;
        process slope line and get new slope;
        # save three lines
        output(file, horizontal line);
        output(file, slope line);
        output(file, slope line);
    else
        set the two x, y coordinates;
        process slope line and get new slope;
        # save three lines
        output(file, vertical line);
        output(file, slope line);
        output(file, slope line);
    end if
end

# function to process and save doublecircle node
procedure savenode.doublecircle(file, gt, node)
    output(file, first circle line);
    # second circle with diameter - 4
    output(file, second circle line);
end

# function to process and save diamond node
procedure savenode.diamond(file, gt, node)
    # define the two x, y coordinates of line, slope and newslop
    maxnum = 6;

    set the two x, y coordinates;
    process slope line and get new slope;
    save four lines;
end

# function to process and save parallelogram node
procedure savenode.parallelogram (file, gt, node)
    # same as for diamond pseudo codes
end
```

```
# function to process and save trapezium node
procedure savenode.trapezium (file, gt, node)
    # same as for diamond pseudo codes
end

# function to process and save Msquare node
procedure savenode.Msquare (file, gt, node)
    # same as for diamond pseudo codes
    # output the four lines with slope(1, 1) at different directions
end
```

Pseudo Codes for `removecommondivisor` Function

```
procedure removecommondivisor (x, y)
    # define temp to hold the new slop returned
    temp[];

    # choose the small number as the maxdivisor
    maxdivisor = min(x, y);

    for i=0 to maxdivisor do
        while temp[0] and temp[1] can be divided by i do
            temp[0] = temp[0]/i;
            temp[1] = temp[1]/i;
            maxdivisor = maxdivisor/i;
    end

    return temp;
end
```

Pseudo Codes for `adjustslope` Function

```
procedure adjustslope (x, y, dx, dy)
    # define temp to hold the new slop returned
    temp[];

    # define tempslope to hold the current slope processed
    tempslope[];

    # define sd to hold the current smallest difference between original
    # and new slope, initiate it as maxmum
    sd = maxmum;

    # define a tempsd for slope difference comparison

    # difference between x and y is bigger than 2 times of maxnum
    if y/x > 2*maxnum then
        temp[0] = 0;
        temp[1] = 1;
        temp[2] = dy;
        return temp;

    if y/x < i/(2*maxnum) then
        temp[0] = 1;
        temp[1] = 0;
        temp[2] = dx;
```

```
        return temp;

    end if

    # iterate through allowed x coordinate
    for i=0 to maxnum do
        # iterate through allowed y coordinate
        for j=0 to maxnum do
            if i=j and i>1 then
                # do nothing, done when i = j = 1
            end if

            # remove common divisors for the slope (i, j)
            tempslope = removeCommonDivisor(i, j);

            # calculate the slope difference between original and current one
            tempsd = y/x - tempslope[1]/tempslope[0];

            # if tempsd less than the previous sd
            if |tempsd| < sd then
                # set tempsd as new sd
                sd = tempsd;
                temp[0] = tempslope[0];
                temp[1] = tempslope[1];
                # adjust slope length
                # new length = old length * old slope / new slope
                temp[2] = dx * y/x * tempslope[0]/tempslope[1];
            end if
    end

end
```

Appendix E-2. Pseudo Code for Drawlets Tool

Pseudo Codes for LatexGenerator Class

```
// the package that LatexGenerator created in
latex.min;

// packages used by SimpleModelPanel
import ...
// more java packages required to handle file output
// support Math function and iterate through Figures
import java.io.*;
import java.lang.*;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Arrays;

class LatexGenerator extends SimplePanel {
   // the model for this application
   model;

   // the temporary file name that will be saved to/restore from
   TempFileName = "temp.rmd";

   // file used to save latex commands
   LatexFileName = "test.tex";

   // latex code generated from drawing
   LatexCode;

   // maxmium number used to define line slope in latex
   // 6 for line and 4 for vector
   maxnum;

   // constructors are in the same formats as for SimpleModelPanel
   LatexGenerator() {…}

   // the component holding the canvas
   getCanvasComponent() {…}

   // the model manipulated
   getModel() {…}

   // Initialize class
   initialize() {
      class ClearAction extends AbstractAction {…}
      class SaveAction extends AbstractAction {…}
      class RestoreAction extends AbstractAction {…}

      // add toolbar for clear, save and restore ations
      toolBar.add (…)

      // Class for Latex code generation
      class SaveLatexAction extends AbstractAction {
         SaveLatexAction (name, icon, model, fileName) {
            super(name, icon);
            set model;
         }

         actionPerformed(action event) {
```

```
// get drawing boundary from model first
// then save in LatexCode
drawWidth, drawHeight;
LatexCode = \begin{picture}(width, height);

// define each figure's boundary, origin, and line slope
figureWidth, figureHeight, originX, originY, slope, newslope;

// process each figure of the drawing, such as box, line etc.
for (first figure; next figure;) {
   // set each figure's boundary and origin
   // from top-left to bottom-left
   figurebound = figure boundary;
   figureWidth = with of figurebound;
   figureHeight = height of figurebound;
   originX = x coordinate of figurebound;
   originY = drawHeight - y coordinate of figurebound;

   if (figure is RectangleShape) {
      // append four lines in Latex command format to LatexCode
      LatexCode += first horizontal line;
      LatexCode += second horizontal line;
      LatexCode += first vertical line;
      LatexCode += second vertical line;
   }
   else if (figure is PolygonShape) {
      maxnum = 6;

      // define the two ending points of a polygon side
      x0, x1, y0, y1;

      // draw lines between consecutive points
      for(i = 0; i < number of points-1; i++){
         // set the two ending points of a polygon side
         x0 = x coordinate of points[i];
         x1 = x coordinate of points[i+1];
         y0 = drawHeight - y coordinate of points[i];
         y1 = drawHeight - y coordinate of points[i+1];

         //vertical line
         if(x0 = x1) {
            if (y0 > y1)
               LatexCode += vertical line;
            else
               LatexCode += vertical line;
         }
         //horizontal line
         else if (y0 == y1) {
            if (x0 > x1)
               LatexCode += vertical line;
            else
               LatexCode += vertical line;
         }
         else if (x0 < x1) {
            if (y0 <y1) {
               // remove common divisors and set new line slope
               // same for the rest slope line, won't repeat.
               slope = removeCommonDivisor(dx,dy);
               newslope = adjustSlope(slope[0],slope[1],dx, dy);
               LatexCode += slope line;
            }
            else {
               process and output slope line with new slope;
```

```
            }
        }
        else if (x0 > x1) {
            if (y0 <y1) {
                process and output slope line with new slope;
            }
            else {
                process and output slope line with new slope;
            }
        }
    }
}
else if (figure is Ellipse) {
    LatexCode += oval line;
}
else if (figure is AdornedLine) {
    maxnum = 4;

    // set arrow coordinates
    x, y;

    // vertical line ends at origin, default width is 8
    if (figureWidth = 8) {
        if(y = y coorcinate of figurebound)
            LatexCode += upward vertical vector;
        else
            LatexCode += downward vertical vector;
    }
    //horizontal line ends at origin, default height is 8
    else if (figureHeight = 8) {
        if(x = originX)
            LatexCode += left horizontal vector;
        else
            LatexCode += right horizontal vector;
    }
    // slope vectors
    else if (figure contains (originX, y coordinate of
    figurebound)) {
        process slope vector;
        // vector ends at origin
        if(x = originX)
            LatexCode += vector y increases while x decreases;
        else
            LatexCode += vector x increases while y decreases;
    }
    else {
        process slope vector;
        // vector ends at origin
        if(x = originX)
            LatexCode += vector x decreases while y decreases;
        else
            LatexCode += vector x increases while y increases;
    }
}
else if (figure is Line) {
    maxnum = 6;

    // vertical line
    if (figureWidth is 0) {
        LatexCode += vertical line;
    }
    // horizontal line
    else if (figureHeight is 0) {
```

```
                        LatexCode += horizontal line;
                    }
                    // slope line that x coordinate decreases while y decreases
                    else if (figure contains originX and y coordinate of
                    figurebound)) {
                        process and output slope line with new slope;
                    }
                    // line that x coordinate increases while y increases
                    else {
                        process and output slope line with new slope;
                    }
                }
                else if (figure is TextLabel) {
                    LatexCode += text label;
                }
                next figure;
            }
            // save LatexCode to TEX file
            saveLatex(LatexFileName, LatexCode);
        }
    }
    // add icon for SaveLatexActioin
    toolBar.add (new SaveLatexAction(with tooltip);
}

// override the getToolBar() with tooltips and correct pathes
JToolBar getToolBar() {…}

// override the getToolPalette()  with tooltips and correct pathes
JToolBar getToolPalette() {…}

// function to store the Latex code of drawings into TEx file
saveLatex(filename, latexcode) {
    create a file;

    // generate LatexCode Prefix
    output (file, \documentclass{article});
    output (file, begin{document});
    // add latex code
    output (file, latexcode);
    // generate LatexCode Suffix
    output (file, \end{picture});
    output (file, \end{document});

    close file;
}

removeCommonDivisor(width, height) {…}

adjustSlope(sx, sy, width, height) {…}

main() {
    initiates a LatexGenerator();
}
}
```

Appendix F-1. Code for Graphviz Tool (Lefty)

```
##########################################################################
#
#            Latex Generator - A customized tool from Graphviz DOTTY
#
##########################################################################

#
# customize DOTTY GUI
#
load ('dotty.lefty');
latex = [];
latex.protogt = [
    'layoutmode' = 'sync';
    'actions' = copy (dotty.protogt.actions);
    # new actions are added later in the file
];
latex.protovt = [
    'name' = 'Latex Generator';
    'type' = 'normal';
];
#
# initialization functions
#
latex.init = function () {
    dotty.init ();
    monitorfile = dotty.monitorfile;
};
#
# main operations
#
latex.main = function () {
    local gnvt, gt, maxnum, slope, newslope;
    #echo ('starting main');

    latex.init ();
    gnvt = dotty.createviewandgraph (null, 'file', latex.protogt,
latex.protovt);
    gt = gnvt.gt;
    txtview ('off');
};
#
# add 'save latex file' action
#
latex.protogt.actions.general['save latex file'] = function (gt, vt, data) {
      gt.savelatexfile (gt, null, 'file');
};
#
# customize 'do layout' action
#
latex.protogt.actions.general['do layout'] = function (gt, vt, data) {
      # choose layout first
      gt.lserver = ask ('select the desired layout:', 'choice', 'dot|neato');
      gt.layoutgraph (gt);
};
#
# create latex file
#
latex.protogt.savelatexfile = function (gt, name, type) {
```

```
    local fd, graph, gid, sgraph, nid, node, eid, edge, pointi, attr, bb, x1,
x2, y1, y2, dx, dy, slope, newslope, line;

    if (~name)
        if (~(name = ask ('file name:', 'file', '')))
            return;

    fd = openio ('file', name, 'w+');

    # LaTeX format
    writeline(fd, '\documentclass{article}');
    writeline(fd, '\begin{document}');

    writeline(fd, '');
    #writeline(fd, '\setlength{\unitlength}{0.013889in} % selecting unit
length');

    # process graph
    graph = copy (gt.graph);
    for (gid in graph.graphs) {
        sgraph = graph.graphs[gid];
        if (sgraph.rect)
            sgraph.graphattr.bb = concat (sgraph.rect[0].x, ',',
                    sgraph.rect[0].y, ',', sgraph.rect[1].x, ',',
                    sgraph.rect[1].y);
        if (sgraph.lp & tablesize (sgraph.lp) > 0)
            sgraph.graphattr.lp = concat (sgraph.lp.x, ',', sgraph.lp.y);
        else if (sgraph.lp)
            sgraph.graphattr.lp = "";
    }

    if (~graph.rect) {
        graph.rect = [
            0 = ['x' = 0; 'y' = 0; ]; 1 = ['x' = 1; 'y' = 1; ];
        ];
        for (nid in graph.nodes) {
            node = graph.nodes[nid];
            if (graph.rect[1].x < node.pos.x + node.size.x / 2)
                graph.rect[1].x = node.pos.x + node.size.x / 2;
            if (graph.rect[1].y < node.pos.y + node.size.y / 2)
                graph.rect[1].y = node.pos.y + node.size.y / 2;
        }
    }
    graph.graphattr.bb = concat (graph.rect[0].x, ',',
            graph.rect[0].y, ',', graph.rect[1].x, ',',
            graph.rect[1].y);

    # write out bb
    bb = concat (graph.rect[1].x, ',',graph.rect[1].y);
    writeline(fd, concat('\begin{picture}(', bb, ')'));

    # write out nodes
    for (nid in graph.nodes) {
        node = graph.nodes[nid];
        writeline(fd, concat('% draw node ', nid));

        latex.savenode[node.attr.shape](fd, gt, node);

        # write out and center node label
        if (node.attr.label ~= '\N')
            writeline(fd, concat('\put(', node.pos.x, ',', node.pos.y,
'){\makebox(0,0){', node.attr.label, '}}'));
        else
```

```
            writeline(fd, concat('\put(', node.pos.x, ',', node.pos.y,
'){\makebox(0,0){', node.name, '}}'));
    }

    # write out edges
    writeline(fd, '% draw edges');
    for (eid in graph.edges) {
        edge = graph.edges[eid];
        if (edge.points) {
            attr = '';
             for (pointi = 0; edge.points[pointi]; pointi = pointi + 1) {
                 # loop through edge points
             }
             # backward arrow edge
            if (edge.sp) {
                #writeline(fd, concat('s,', edge.sp.x, ',', edge.sp.y, ' '));
                 x1 = edge.sp.x;
                 y1 = edge.sp.y;
                 x2 = edge.points[pointi-1].x;
                 y2 = edge.points[pointi-1].y;
             }
             # forward arrow edge
            else if (edge.ep) {
                 #writeline(fd, concat('e,', edge.ep.x, ',', edge.ep.y, ' '));
                x1 = edge.ep.x;
                 y1 = edge.ep.y;
                 x2 = edge.points[0].x;
                 y2 = edge.points[0].y;
             }
             # none arrow edge
             else {
                 x2 = edge.points[0].x;
                 y2 = edge.points[0].y;
                 x1 = edge.points[pointi-1].x;
                 y1 = edge.points[pointi-1].y;
             }
        }

         # decide max number for latex slope
         if (edge.attr.dir == 'none') {
             maxnum = 6;
             line = 'line';
         }
         else {
             maxnum = 4;
             line = 'vector';
         }

         dx = x1 - x2;
         dy = y1 - y2;
         slope = [];
         newslope = [];

         # minimize dx, dy with no common divisors larger than one
         if (dx > 0 & dy > 0) {
             slope = latex.removecommondivisor (integer(dx), integer(dy));
             newslope = latex.adjustslope (slope[0], slope[1], dx, dy);
             writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(',
newslope[0], ',', newslope[1], '){', newslope[2], '}}'));
         } else if (dx > 0 & dy < 0) {
            slope = latex.removecommondivisor (integer(dx), integer(-dy));
             newslope = latex.adjustslope (slope[0], slope[1], dx, -dy);
```

```
            writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(',
newslope[0], ',', -newslope[1], '){', newslope[2], '}}'));
        } else if (dx > 0 & dy == 0) {
            writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(', 1,
',', integer(dy), '){', dx, '}}'));
        } else if (dx < 0 & dy > 0) {
            slope = latex.removecommondivisor (integer(-dx), integer(dy));
            newslope = latex.adjustslope (slope[0], slope[1], -dx, dy);
            writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(', -
newslope[0], ',', newslope[1], '){', newslope[2], '}}'));
        } else if (dx < 0 & dy < 0) {
            slope = latex.removecommondivisor (integer(-dx), integer(-dy));
            newslope = latex.adjustslope (slope[0], slope[1], -dx, -dy);
            writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(', -
newslope[0], ',', -newslope[1], '){', newslope[2], '}}'));
        } else if (dx < 0 & dy == 0) {
            writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(', -1,
',', integer(dy), '){', (-dx), '}}'));
        } else if (dx == 0 & dy > 0) {
            writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(',
integer(dx), ',', 1, '){', dy, '}}'));
        } else if (dx == 0 & dy < 0) {
            writeline(fd, concat('\put(', x2, ',', y2, '){\', line, '(',
integer(dx), ',', -1, '){', (-dy), '}}'));
        } else {
            # don't exist, do nothing
        }

        # write out edge label
        if (edge.lp)
            writeline(fd, concat ('\put(', edge.lp.x - 2.5 * strlen
(edge.attr.label), ',', edge.lp.y, '){', edge.attr.label, '}'));
    }

    #write out graph label
    if (graph.lp & tablesize (graph.lp) > 0) {
        graph.graphattr.lp = concat (graph.lp.x, ',', graph.lp.y);
        writeline(fd, concat ('\put(', graph.lp.x - 2.5 * strlen
(graph.graphattr.label), ',', graph.lp.y, '){', graph.graphattr.label, '}'));
    }

    writeline(fd, '\end{picture}');

    # MiKTeX format
    #writeline(fd, '\bye');

    # PCTeX format
    writeline(fd, '\end{document}');

    # from save funciton, can write graph in DOTTY format to tex file
    #if (~((fd = dotty.openio (name, type, 'w')) >= 0) |
    #       ~writegraph (fd, graph, 0)) {
    #    dotty.message (0, 'cannot save graph');
    #    return;
    #}

    if (~(type == 'file' & name == '-'))
        closeio (fd);
};
#
# remove common divisors
#
latex.removecommondivisor = function (x, y) {
```

```
    local temp, divisor, maxdivisor;
    temp = [];
    maxdivisor = min(x, y);
    temp[0] = x;
    temp[1] = y;

    for(divisor = 2; divisor <= maxdivisor; divisor = divisor + 1) {
        while (((temp[0]/divisor - toint(temp[0]/divisor)) == 0) &
((temp[1]/divisor - toint(temp[1]/divisor)) == 0)) {
            temp[0] = temp[0]/divisor;
            temp[1] = temp[1]/divisor;
            maxdivisor = maxdivisor/divisor;
        }
    }
    return temp;
};
#
# find minimum number
#
min = function (a, b) {
    if (a <= b)
        return a;
    return b;
};
#
# change to integer
#
integer = function (a) {
    if (a < 0) {
        if ((toint(a) - a) > 0.5)
            return toint(a) - 1;
        else
            return toint(a);
    } else {
        if ((a - toint(a)) > 0.5)
            return toint(a) + 1;
        else
            return toint(a);
    }
};
#
# adjust slope and its length
#
latex.adjustslope = function (sx, sy, lx, ly) {
    local temp, slopex, slopey, sd, tempsd, tempslope;
    temp = [];
    tempslope = [];
    sd = maxnum;

    # if slope is bigger than 2 times of maxnum
    if (sy/sx > 2 * maxnum) {
        temp[0] = 0;
        temp[1] = 1;
        temp[2] = ly;
        return temp;
    }

    if (sy/sx < 1 /(2 * maxnum)) {
        temp[0] = 1;
        temp[1] = 0;
        temp[2] = lx;
        return temp;
    }
```

```
    for(slopex = 1; slopex <= maxnum; slopex = slopex + 1) {
        for (slopey = 1; slopey <= maxnum; slopey = slopey + 1) {
            if ((slopex == slopey) & (slopex >1))
                continue;

            tempslope = latex.removecommondivisor (slopex, slopey);
            tempsd = sy/sx - tempslope[1]/tempslope[0];
            if (tempsd < 0)
                tempsd = - tempsd;
            if (tempsd <= sd) {
                sd = tempsd;
                temp[0] = tempslope[0];
                temp[1] = tempslope[1];
                temp[2] = lx * sy/sx * tempslope[0]/tempslope[1];
                #temp[2] = lx;
            }
        }
    }
    return temp;
};
#
# write out box node
#
latex.savenode.box = function (fd, gt, node) {
    local x1, x2, y1, y2;

    x1 = node.pos.x - node.size.x / 2;
    x2 = node.pos.x + node.size.x / 2;
    y1 = node.pos.y - node.size.y / 2;
    y2 = node.pos.y + node.size.y / 2;

    writeline(fd, concat('\put(', x1, ',', y1, '){\line(1,0){', node.size.x,
'}}'));
    writeline(fd, concat('\put(', x1, ',', y2, '){\line(1,0){', node.size.x,
'}}'));
    writeline(fd, concat('\put(', x1, ',', y1, '){\line(0,1){', node.size.y,
'}}'));
    writeline(fd, concat('\put(', x2, ',', y1, '){\line(0,1){', node.size.y,
'}}'));
};
#
# write out circle node
#
latex.savenode.circle = function (fd, gt, node) {
    local diameter;

    diameter = min(node.size.x, node.size.y);

    if (node.attr.style == 'filled')
        writeline(fd, concat('\put(', node.pos.x, ',', node.pos.y,
'){\circle*{', diameter, '}}'));
    else
        writeline(fd, concat('\put(', node.pos.x, ',', node.pos.y,
'){\circle{', diameter, '}}'));
};
#
# write out ellipse node
#
latex.savenode.ellipse = function (fd, gt, node) {
    writeline(fd, concat('\put(', node.pos.x, ',', node.pos.y, '){\oval(',
node.size.x, ',', node.size.y, ')}'));
};
```

```
#
# write out triangle node
#
latex.savenode.triangle = function (fd, gt, node) {
    local x1, x2, y1, y2;
    maxnum = 6;
    slope = [];
    newslope = [];

    if (node.attr.orientation ~= -90) {
        x1 = node.pos.x - node.size.x/2 * 3/4;
        x2 = node.pos.x + node.size.x/2 * 3/4;
        y1 = node.pos.y - node.size.y / 4;
        y2 = node.pos.y + node.size.y / 2;

        slope = latex.removecommondivisor (integer(node.size.x/2),
integer(node.size.y));
        newslope = latex.adjustslope (slope[0], slope[1], node.size.x/2 * 3/4,
node.size.y * 3/4);

        writeline(fd, concat('\put(', x1, ',', y1, '){\line(1,0){', node.size.x
* 3/4, '}}'));
        writeline(fd, concat('\put(', x1, ',', y1, '){\line(', newslope[0],
',', newslope[1], '){', newslope[2], '}}'));
        writeline(fd, concat('\put(', x2, ',', y1, '){\line(', -newslope[0],
',', newslope[1], '){', newslope[2], '}}'));
    } else {
        x1 = node.pos.x - node.size.x/4;
        x2 = node.pos.x + node.size.x/2;
        y1 = node.pos.y - node.size.y/2 * 3/4;
        y2 = node.pos.y + node.size.y/2 * 3/4;

        slope = latex.removecommondivisor (integer(node.size.x),
integer(node.size.y/2));
        newslope = latex.adjustslope (slope[0], slope[1], node.size.x * 3/4,
node.size.y/2 * 3/4);

        writeline(fd, concat('\put(', x1, ',', y1, '){\line(0,1){', node.size.y
* 3/4, '}}'));
        writeline(fd, concat('\put(', x1, ',', y1, '){\line(', newslope[0],
',', newslope[1], '){', newslope[2], '}}'));
        writeline(fd, concat('\put(', x1, ',', y2, '){\line(', newslope[0],
',', -newslope[1], '){', newslope[2], '}}'));
    }
};
#
# write out doublecircle node
#
latex.savenode.doublecircle = function (fd, gt, node) {
    local diameter;
    diameter = min(node.size.x, node.size.y);

    writeline(fd, concat('\put(', node.pos.x, ',', node.pos.y, '){\circle{',
diameter-4, '}}'));
    writeline(fd, concat('\put(', node.pos.x, ',', node.pos.y, '){\circle{',
diameter, '}}'));
};
#
# write out diamond node
#
latex.savenode.diamond = function (fd, gt, node) {
    local x1, x2, y1, y2;
    maxnum = 6;
```

```
    x1 = node.pos.x - node.size.x / 2;
    x2 = node.pos.x + node.size.x / 2;
    y1 = node.pos.y - node.size.y / 2;
    y2 = node.pos.y + node.size.y / 2;

    slope = [];
    newslope = [];
    slope = latex.removecommondivisor (integer(node.size.x/2),
integer(node.size.y/2));
    newslope = latex.adjustslope (slope[0], slope[1], node.size.x/2,
node.size.y/2);

    writeline(fd, concat('\put(', x1, ',', node.pos.y, '){\line(', newslope[0],
',', newslope[1], '){', newslope[2], '}}'));
    writeline(fd, concat('\put(', x1, ',', node.pos.y, '){\line(', newslope[0],
',', -newslope[1], '){', newslope[2], '}}'));
    writeline(fd, concat('\put(', x2, ',', node.pos.y, '){\line(', -
newslope[0], ',', newslope[1], '){', newslope[2], '}}'));
    writeline(fd, concat('\put(', x2, ',', node.pos.y, '){\line(', -
newslope[0], ',', -newslope[1], '){', newslope[2], '}}'));
};
#
# write out parallelogram node
#
latex.savenode.parallelogram = function (fd, gt, node) {
    local x1, x2, y1, y2;
    maxnum = 6;
    x1 = node.pos.x - node.size.x / 2;
    x2 = node.pos.x + node.size.x / 2;
    y1 = node.pos.y - node.size.y / 2;
    y2 = node.pos.y + node.size.y / 2;

    slope = [];
    newslope = [];
    slope = latex.removecommondivisor (integer(node.size.x/5),
integer(node.size.y));
    newslope = latex.adjustslope (slope[0], slope[1], node.size.x/5,
node.size.y);

    writeline(fd, concat('\put(', x1, ',', y1, '){\line(1,0){', node.size.x *
4/5, '}}'));
    writeline(fd, concat('\put(', x2, ',', y2, '){\line(-1,0){', node.size.x *
4/5, '}}'));
    writeline(fd, concat('\put(', x1, ',', y1, '){\line(', newslope[0], ',',
newslope[1], '){', newslope[2], '}}'));
    writeline(fd, concat('\put(', x2, ',', y2, '){\line(', -newslope[0], ',', -
newslope[1], '){', newslope[2], '}}'));
};
#
# write out trapezium node
#
latex.savenode.trapezium = function (fd, gt, node) {
    local x1, x2, y1, y2;
    maxnum = 6;
    x1 = node.pos.x - node.size.x / 2;
    x2 = node.pos.x + node.size.x / 2;
    y1 = node.pos.y - node.size.y / 2;
    y2 = node.pos.y + node.size.y / 2;

    slope = [];
    newslope = [];
    slope = latex.removecommondivisor (integer(node.size.x/5),
integer(node.size.y));
```

```
    newslope = latex.adjustslope (slope[0], slope[1], node.size.x/5,
node.size.y);

    writeline(fd, concat('\put(', x1, ',', y1, '){\line(1,0){', node.size.x,
'}}'));
    writeline(fd, concat('\put(', x2 - node.size.x/5, ',', y2, '){\line(-
1,0){', node.size.x * 3/5, '}}'));
    writeline(fd, concat('\put(', x1, ',', y1, '){\line(', newslope[0], ',',
newslope[1], '){', newslope[2], '}}'));
    writeline(fd, concat('\put(', x2, ',', y1, '){\line(', -newslope[0], ',',
newslope[1], '){', newslope[2], '}}'));
};
#
# write out Msquare node
#
latex.savenode.Msquare = function (fd, gt, node) {
    local x1, x2, y1, y2;
    x1 = node.pos.x - node.size.x / 2;
    x2 = node.pos.x + node.size.x / 2;
    y1 = node.pos.y - node.size.y / 2;
    y2 = node.pos.y + node.size.y / 2;

    writeline(fd, concat('\put(', x1, ',', y1, '){\line(1,0){', node.size.x,
'}}'));
    writeline(fd, concat('\put(', x1, ',', y2, '){\line(1,0){', node.size.x,
'}}'));
    writeline(fd, concat('\put(', x1, ',', y1, '){\line(0,1){', node.size.y,
'}}'));
    writeline(fd, concat('\put(', x2, ',', y1, '){\line(0,1){', node.size.y,
'}}'));
    writeline(fd, concat('\put(', x1 + 10, ',', y1, '){\line(-1,1){10}}'));
    writeline(fd, concat('\put(', x2 - 10, ',', y1, '){\line(1,1){10}}'));
    writeline(fd, concat('\put(', x1, ',', y2 - 10, '){\line(1,1){10}}'));
    writeline(fd, concat('\put(', x2, ',', y2 - 10, '){\line(-1,1){10}}'));
};
#
# add new menu 'save latex file'
#
latex.protovt.menus = [
       'general' = [
        0 = "undo";
        1 = "paste";
        2 = "do layout";
        3 = "cancel layout";
        4 = "redraw";
        5 = "new graph";
        6 = "load graph";
        7 = "reload graph";
        8 = "save graph";
        9 = "save graph as";
        10 = "open view";
        11 = "copy view";
        12 = "clone view";
        13 = "birdseye view";
        14 = "close view";
        15 = "set graph attr";
        16 = "set node attr";
        17 = "set edge attr";
        18 = "zoom in";
        19 = "zoom out";
        20 = "find node";
        21 = "print graph";
        22 = "text view";
```

```
        23 = "save latex file";
         24 = "quit";
    ];
    'node' = [
        0 = "cut";
        1 = "Cut";
        2 = "copy";
        3 = "Copy";
        4 = "group";
        5 = "Group";
        6 = "delete";
        7 = "Delete";
        8 = "remove";
        9 = "Remove";
        10 = "set attr";
        11 = "print attr";
    ];
    'edge' = [
        0 = "cut";
        1 = "Cut";
        2 = "copy";
        3 = "Copy";
        4 = "delete";
        5 = "Delete";
        6 = "set attr";
        7 = "print attr";
    ];
];

latex.protovt.keys = dotty.protovt.normal.keys;

latex.protovt.uifuncs = dotty.protovt.normal.uifuncs;

latex.main ();
```

Appendix F-2. Code for Drawlets Tool (Java)

```
package latex.main;

import com.rolemodelsoft.drawlet.*;
import com.rolemodelsoft.drawlet.basics.*;
import com.rolemodelsoft.drawlet.jfc.*;
import com.rolemodelsoft.drawlet.examples.*;
import com.rolemodelsoft.drawlet.examples.jfc.*;
import com.rolemodelsoft.drawlet.shapes.*;
import com.rolemodelsoft.drawlet.shapes.lines.*;
import com.rolemodelsoft.drawlet.shapes.rectangles.*;
import com.rolemodelsoft.drawlet.shapes.ellipses.*;
import com.rolemodelsoft.drawlet.shapes.polygons.*;
import com.rolemodelsoft.drawlet.text.LabelTool;
import com.rolemodelsoft.drawlet.text.TextLabel;
import com.rolemodelsoft.drawlet.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.lang.*;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Arrays;


/**
 * Latex Generator - A customized tool from Drawlets framework.
 *
 * @version  1.0.1
 *
 * Modification History:

********************************************************************************
 * Date              Author      Comments
 * ------------------------------------------------------------------------------
-
 * 13/07/2004     Jie Xiao      Creation
 * 28/08/2004     Jie Xiao      Add functions to handle arrow
 * 18/01/2005     Jie Xiao      Output Ellipse and TextLabel coordinates as
decimal
 */

public class LatexGenerator extends SimplePanel {

       /**
        * the model for this application.
        */
       protected SingleDrawingModel model;

       /**
        * the temporary file name that will be saved to/restore from.
        */
       protected static String TempFileName = "temp.rmd";
        /**
        * Latex code file name.
        */
       protected static String LatexFileName = "test.tex";
        /**
```

```
	 * Latex code generated from drawing.
	 */
	protected static String LatexCode;
	 /**
	 * Maxmium number used to define line slope in latex
	  * 6 for line and 4 for vector.
	 */
	protected static int maxnum;

	/**
	 * Default constructor.
	 */
	public LatexGenerator() {
		super();
	}
	/**
	 * @param layout the layout this application should use.
	 */
	public LatexGenerator(java.awt.LayoutManager layout) {
		super(layout);
	}
	/**
	 * LatexGenerator constructor comment.
	 * @param layout the layout this application should use.
	 * @param isDoubleBuffered determines whether the application will be
	 * double buffered or not.
	 */
	public LatexGenerator(java.awt.LayoutManager layout, boolean
isDoubleBuffered) {
		super(layout, isDoubleBuffered);
	}
	/**
	 * @param isDoubleBuffered  determines whether the application will be
	 * double buffered or not.
	 */
	public LatexGenerator(boolean isDoubleBuffered) {
		super(isDoubleBuffered);
	}
	/**
	 * @return Component the component holding the canvas.
	 */
	protected JComponent getCanvasComponent() {
		canvas = new SimpleDrawingCanvas(getModel().getDrawing());
		JComponent component = new JDrawingCanvasComponent(canvas);
		ValueAdapter adapter = new
ValueAdapter(getModel(),"getDrawing",canvas,"setDrawing");
		return component;
	}
	/**
	 * @return the model
	 */
	protected SingleDrawingModel getModel() {
		if ( model == null ) {
			model = new SingleDrawingModel();
		}
		return model;
	}
	/**
	 * Initialize class
	 */
	protected void initialize() {
		super.initialize();
		setSize(426, 300);
```

```
            toolBar.addSeparator();

            class ClearAction extends AbstractAction {
                  SingleDrawingModel model;
                  public ClearAction(String name,  Icon icon,
SingleDrawingModel model) {
                        super(name, icon);
                        this.model = model;
                  }
                  public void actionPerformed(ActionEvent e) {
                        model.clearDrawing();
                  }
            }
            class SaveAction extends AbstractAction {
                  SingleDrawingModel model;
                  String fileName;
                  public SaveAction(String name, Icon icon,
SingleDrawingModel model, String fileName) {
                        super(name, icon);
                        this.model = model;
                        this.fileName = fileName;
                  }
                  public void actionPerformed(ActionEvent e) {
                        model.saveDrawing(fileName);
                  }
            }
            class RestoreAction extends AbstractAction {
                  SingleDrawingModel model;
                  String fileName;
                  public RestoreAction(String name, Icon icon,
SingleDrawingModel model, String fileName) {
                        super(name, icon);
                        this.model = model;
                        this.fileName = fileName;
                  }
                  public void actionPerformed(ActionEvent e) {
                        model.restoreDrawing(fileName);
                  }
            }
            toolBar.add(new ClearAction("", new ImageIcon("images/clear.gif",
"Clear"), getModel())).setToolTipText("Clear");
            toolBar.add(new SaveAction("", new ImageIcon("images/save.gif",
"Save"), getModel(),TempFileName)).setToolTipText("Save drawing to temp.rmd");
            toolBar.add(new RestoreAction("Restore",new
ImageIcon("images/restore.gif", "Restore"),
getModel(),TempFileName)).setToolTipText("Restore drawing from temp.rmd");

               toolBar.addSeparator();
               // Class for Latex code generation
               class SaveLatexAction extends AbstractAction {
                   SingleDrawingModel model;
                       String fileName;
                   public SaveLatexAction(String name, Icon icon,
SingleDrawingModel model, String fileName) {
                        super(name, icon);
                        this.model = model;
                              this.fileName = fileName;
                   }
                   public void actionPerformed(ActionEvent e) {
                              // translate drawing to LatexCode, get boundary
first
```

```
                                int drawWidth =
model.getDrawing().getBounds().width;
                                int drawHeight =
model.getDrawing().getBounds().height;
                                //System.out.print("drawWidth: " +
Integer.toString(drawWidth) + "drawHeight: " + Integer.toString(drawHeight)
+"\n");
                                LatexCode = "\\begin{picture}(" +
Integer.toString(drawWidth) + "," + Integer.toString(drawHeight) + ")" + "\n";
                                // process each element of the drawing, such as
Box, Line, Ellipase etc.
                                Rectangle figurebound = new Rectangle( 0, 0, 0,
0 );
                                Figure figure;
                                int figureWidth;
                                int figureHeight;
                                int originX;
                                int originY;
                                int figureNum = 0;
                                // adjusted slope for lines
                                int[] slope = {0, 0};
                                int[] newslope = {0,0,0};
                            for (FigureEnumeration fe =
model.getDrawing().figures(); fe.hasMoreElements() ;) {
                                    figure = fe.nextElement();
                                    figurebound = figure.getBounds();
                                    //figurebound =
fe.nextElement().getBounds();
                                figureWidth = figurebound.width;
                                    figureHeight = figurebound.height;
                                    originX = figurebound.x;
                                    // set origin from top-left to bottom-left
                                    originY = drawHeight - figurebound.y;
                                    LatexCode += "% draw figure " + figureNum +
"\n";
                                  //  System.out.print("figureWidth: " +
Integer.toString(figureWidth) + "figureHeight: " +
Integer.toString(figureHeight) + "originX" + Integer.toString(originX) +
"originY" + Integer.toString(originY) + "\n");
                                    if (figure instanceof RectangleShape) {
                                        LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) + "){\\line(1,0){"
+ Integer.toString(figureWidth) + "}}" + "\n";
                                        LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) + "){\\line(0,-1){"
+ Integer.toString(figureHeight) + "}}" + "\n";
                                        LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY - figureHeight) +
"){\\line(1,0){" + Integer.toString(figureWidth) + "}}" + "\n";
                                        LatexCode += "\\put(" +
Integer.toString(originX + figureWidth) + "," + Integer.toString(originY) +
"){\\line(0,-1){" + Integer.toString(figureHeight) + "}}" + "\n";
                                    }
                                    else if (figure instanceof PolygonShape) {
                                        PolygonShape polygon = (PolygonShape)
figure;
                                        int sideNum =
polygon.getPolygon().npoints;
                                        int x0, x1, y0, y1;
                                        maxnum = 6;
                                        //System.out.print("number of side is:
" + Integer.toString(sideNum) + "\n");
```

```
                                        // draw lines between consecutive
points for triangle and pentagon
                                        for(int i = 0; i <
polygon.getPolygon().npoints-1; i++){
                                            x0 =
polygon.getPolygon().xpoints[i];
                                            x1 =
polygon.getPolygon().xpoints[i+1];
                                            y0 = drawHeight -
polygon.getPolygon().ypoints[i];
                                            y1 = drawHeight -
polygon.getPolygon().ypoints[i+1];
                                            if(x0 == x1) {
                                                if (y0 > y1)
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(0,-1){" +
Integer.toString(y0-y1) + "}}" + "\n";
                                                else
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(0,1){" +
Integer.toString(y1-y0) + "}}" + "\n";
                                            }
                                            else if (y0 == y1) {
                                                if (x0 > x1)
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(-1,0){" +
Integer.toString(x0-x1) + "}}" + "\n";
                                                else
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(1,0){" +
Integer.toString(x1-x0) + "}}" + "\n";
                                            }
                                            else if (x0 < x1) {
                                                if (y0 <y1) {
                                                    slope =
removeCommonDivisor(x1-x0, y1-y0);
                                                    newslope =
adjustSlope(slope[0], slope[1], x1-x0, y1-y0);
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(" +
Integer.toString(newslope[0]) + "," + Integer.toString(newslope[1]) + "){" +
Integer.toString(newslope[2]) + "}}" + "\n";
                                                }
                                                else {
                                                    slope =
removeCommonDivisor(x1-x0, y0-y1);
                                                    newslope =
adjustSlope(slope[0], slope[1], x1-x0, y0-y1);
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(" +
Integer.toString(newslope[0]) + "," + Integer.toString(-newslope[1]) + "){" +
Integer.toString(newslope[2]) + "}}" + "\n";
                                                }
                                            }
                                            else if (x0 > x1) {
                                                if (y0 <y1) {
                                                    slope =
removeCommonDivisor(x0-x1, y1-y0);
                                                    newslope =
adjustSlope(slope[0], slope[1], x0-x1, y1-y0);
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(" +
```

```
Integer.toString(-newslope[0]) + "," + Integer.toString(newslope[1]) + "){" +
Integer.toString(newslope[2]) + "}}" + "\n";
                                                }
                                                else {
                                                    slope =
removeCommonDivisor(x0-x1, y0-y1);
                                                    newslope =
adjustSlope(slope[0], slope[1], x0-x1, y0-y1);
                                                    LatexCode += "\\put(" +
Integer.toString(x0) + "," + Integer.toString(y0) + "){\\line(" +
Integer.toString(-newslope[0]) + "," + Integer.toString(-newslope[1]) + "){" +
Integer.toString(newslope[2]) + "}}" + "\n";
                                                }
                                            }
                                        }
                                    }
                                    else if (figure instanceof Ellipse) {
                                        LatexCode += "\\put(" +
Double.toString((double)originX + (double)figureWidth/2) + "," +
Double.toString((double)originY - (double)figureHeight/2) + "){\\oval(" +
Integer.toString(figureWidth) + "," + Integer.toString(figureHeight) + ")}" +
"\n";
                                        //LatexCode += "\\put(" +
Integer.toString(originX + figureWidth/2) + "," + Integer.toString(originY -
figureHeight/2) + "){\\oval(" + Integer.toString(figureWidth) + "," +
Integer.toString(figureHeight) + ")}" + "\n";
                                    }
                                    else if (figure instanceof AdornedLine) {
                                        Arrow adornment1 = new
Arrow((AdornedLine)figure);
                                        int x = adornment1.getLocator().x();
                                        int y = adornment1.getLocator().y();
                                        maxnum = 4;
                                     //   System.out.print("x() is: " +
Integer.toString(x) + "y() is:" + Integer.toString(y));
                                      //   System.out.print("figureWidth: " +
Integer.toString(figureWidth) + "figureHeight: " +
Integer.toString(figureHeight) + "originX" + Integer.toString(originX) +
"originY" + Integer.toString(figurebound.y) + "\n");

                                        if (figureWidth == 8) {
                                            // vertical line ends at origin,
default width is 8
                                            if(y == figurebound.y)
                                                LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY - figureHeight) +
"){\\vector(0,1){" + Integer.toString(figureHeight) + "}}" + "\n";
                                            else
                                                LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) + "){\\vector(0,-
1){" + Integer.toString(figureHeight) + "}}" + "\n";
                                        }
                                        else if (figureHeight == 8) {
                                            // horizontal line ends at origin,
default height is 8
                                            if(x == originX)
                                                LatexCode += "\\put(" +
Integer.toString(originX + figureWidth) + "," + Integer.toString(originY) +
"){\\vector(-1,0){" + Integer.toString(figureWidth) + "}}" + "\n";
                                            else
                                                LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) +
"){\\vector(1,0){" + Integer.toString(figureWidth) + "}}" + "\n";
```

```
                                        }
                                        else if (figure.contains(originX,
figurebound.y)) {
                                            slope =
removeCommonDivisor(figureWidth, figureHeight);
                                          newslope = adjustSlope(slope[0],
slope[1], figureWidth, figureHeight);
                                            // line ends at origin
                                            if(x == originX)
                                                LatexCode += "\\put(" +
Integer.toString(originX + figureWidth) + "," + Integer.toString(originY -
figureHeight) + "){\\vector(" + Integer.toString(-newslope[0]) + "," +
Integer.toString(newslope[1]) + "){" + Integer.toString(newslope[2]) + "}}" +
"\n";
                                            else
                                                LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) + "){\\vector(" +
Integer.toString(newslope[0]) + "," + Integer.toString(-newslope[1]) + "){" +
Integer.toString(newslope[2]) + "}}" + "\n";
                                        }
                                        else {
                                            slope =
removeCommonDivisor(figureWidth, figureHeight);
                                          newslope = adjustSlope(slope[0],
slope[1], figureWidth, figureHeight);
                                            // line ends at origin
                                            if(x == originX)
                                                LatexCode += "\\put(" +
Integer.toString(originX + figureWidth) + "," + Integer.toString(originY) +
"){\\vector(" + Integer.toString(-newslope[0]) + "," + Integer.toString(-
newslope[1]) + "){" + Integer.toString(newslope[2]) + "}}" + "\n";
                                            else
                                                LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY - figureHeight) +
"){\\vector(" + Integer.toString(newslope[0]) + "," +
Integer.toString(newslope[1]) + "){" + Integer.toString(newslope[2]) + "}}" +
"\n";
                                        }
                                    }
                                    else if (figure instanceof Line) {
                                      //  System.out.print("figureWidth: " +
Integer.toString(figureWidth) + "figureHeight: " +
Integer.toString(figureHeight) + "originX" + Integer.toString(originX) +
"originY" + Integer.toString(originY) + "\n");
                                        maxnum = 6;
                                        Line line = (Line) figure;
                                        Locator dest = line.getLocator(0);
                                        Locator source = line.getLocator(1);
                                        // need the locator for the two nodes
                                        /*protected Locator getArrowLocator(
Locator loc ) {
                                            Locator dest =
getDestinationLocator();
                                            Locator source =
getSourceLocator();
                                            Locator relative = new
DrawingPoint(dest.x()-source.x(), dest.y()-source.y());
                                            PolarCoordinate coord = new
PolarCoordinate( loc.r(), loc.theta() + relative.theta() );
                                            return new DrawingPoint( dest.x()
+ coord.x(), dest.y() + coord.y() );
                                        }*/
                                        if (figureWidth == 0) {
```

```
                                        //System.out.print("width is 0 ");
                                        LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) + "){\\line(0,-1){"
+ Integer.toString(figureHeight) + "}}" + "\n";
                                    }
                                    else if (figureHeight == 0) {
                                        //System.out.print("height is 0 ");
                                        LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) + "){\\line(1,0){"
+ Integer.toString(figureWidth) + "}}" + "\n";
                                    }
                                    else if (figure.contains(originX,
figurebound.y)) {
                                        // line starts from left-top to
right-bottom
                                        //System.out.print("contains point
");
                                        slope =
removeCommonDivisor(figureWidth, figureHeight);
                                        newslope = adjustSlope(slope[0],
slope[1], figureWidth, figureHeight);
                                        LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY) + "){\\line(" +
Integer.toString(newslope[0]) + "," + Integer.toString(-newslope[1]) + "){" +
Integer.toString(newslope[2]) + "}}" + "\n";
                                    }
                                    else {
                                        // line starts from left-bottom to
right-top
                                        //System.out.print("doesn't contain
point ");
                                        slope =
removeCommonDivisor(figureWidth, figureHeight);
                                        newslope = adjustSlope(slope[0],
slope[1], figureWidth, figureHeight);
                                        LatexCode += "\\put(" +
Integer.toString(originX) + "," + Integer.toString(originY - figureHeight) +
"){\\line(" + Integer.toString(newslope[0]) + "," +
Integer.toString(newslope[1]) + "){" + Integer.toString(newslope[2]) + "}}" +
"\n";
                                    }
                                }
                                else if (figure instanceof TextLabel) {
                                    TextLabel tl = (TextLabel)figure;
                                    LatexCode += "\\put(" +
Double.toString((double)originX + (double)figureWidth/2) + "," +
Double.toString((double)originY - (double)figureHeight/2) + "){\\makebox(0,0)"
+ "{" + tl.getString() + "}}" + "\n";
                                }
                                figureNum ++;
                            }
                            // save LatexCode to the tex file
                            saveLatex(LatexFileName, LatexCode);
                        }
                    }
                    toolBar.add(new SaveLatexAction("", new
ImageIcon("images/latex.gif", "Save"), getModel(),
LatexFileName)).setToolTipText("Save drawing to Latex file");
        }
        /**
         * @return the ToolBar with tooltips.
         */
        public JToolBar getToolBar() {
```

```
            if ( toolBar == null ) {
                  toolBar = new JToolBar();
                  toolBar.setFloatable( false );
                  toolBar.add(new CanvasAction("", new
ImageIcon("images/copy.gif", "Copy"), canvas,
"copySelections")).setToolTipText("Copy Selections");
                  toolBar.add(new CanvasAction("", new
ImageIcon("images/cut.gif", "Cut"), canvas,
"cutSelections")).setToolTipText("Cut Selections");
                  toolBar.add(new CanvasAction("", new
ImageIcon("images/paste.gif", "Paste"), canvas,
"paste")).setToolTipText("Paste");
                  toolBar.add(new CanvasAction("", new
ImageIcon("images/systempaste.gif", "Paste From System"), canvas,
"pasteFromSystem")).setToolTipText("Paste From System");
            }
            return toolBar;
      }
       /**
        * @return the ToolPalette with tooltips.
        */
      public JToolBar getToolPalette() {
            if ( toolPalette == null ) {
                  toolPalette = new JToolBar();
                  toolPalette.setLayout( new BoxLayout( toolPalette,
BoxLayout.Y_AXIS ) );
                  toolPalette.setFloatable( false );

                      toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/select.gif", "Select"), canvas, (InputEventHandler)new
SelectionTool(canvas))).setToolTipText("Select");
                  toolPalette.addSeparator();
                      toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/label.gif", "Label"), canvas, (InputEventHandler)new
LabelTool(canvas))).setToolTipText("Label");
                  toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/line.gif", "Line"), canvas, (InputEventHandler)new
ConnectingLineTool(canvas))).setToolTipText("Line");
                      toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/arrowline.gif", "ArrowLine"), canvas, (InputEventHandler)new
AdornedLineTool(canvas))).setToolTipText("ArrowLine");
                  toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/box.gif", "Box"), canvas, (InputEventHandler)new
RectangleTool(canvas))).setToolTipText("Box");
                  //toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/rrect.gif", "Rounded"), canvas, (InputEventHandler)new
RectangularCreationTool(canvas,RoundedRectangleShape.class))).setToolTipText("R
ounded");

                      toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/ellipse.gif", "Ellipse"), canvas, (InputEventHandler)new
EllipseTool(canvas))).setToolTipText("Ellipse");
                  toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/triangle.gif", "Triangle"), canvas, (InputEventHandler)new
PolygonTool(canvas,3))).setToolTipText("Triangle");
                  toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/pentagon.gif", "Pentagon"), canvas, (InputEventHandler)new
PolygonTool(canvas,5))).setToolTipText("Pentagon");
                  //toolPalette.add(new CanvasToolAction("", new
ImageIcon("images/nsided.gif", "N-Sided"), canvas, (InputEventHandler)new
AnySidedPolygonTool(canvas))).setToolTipText("N-Sided");
```

```
                    //toolPalette.add(new CanvasToolAction("", new
ImageIcon("greygrey/freehand.gif", "Smooth Line"), canvas,
(InputEventHandler)new DrawLineTool(canvas)));
                }
                return toolPalette;
        }
        /**
         * Store the Latex code of drawings into tex file
         *
         * @param filename       the name of the file containing latex code.
        */
        public void saveLatex(String filename, String latexcode) {
            try {
                String userdir = System.getProperty("user.dir");
                String filesep = System.getProperty("file.separator");
                String LOG_PATH = userdir + filesep;
                // if file already exists, delete it
                File file = new File(LOG_PATH + filename);
                if (file.exists())
                      file.delete();

                FileWriter out = new FileWriter(LOG_PATH + filename, true);

                // generate LatexCode Prefix
                out.write("\\documentclass{article}" + "\n");
                out.write("\\begin{document}" + "\n\n");
                // write LatexCode of the drawing
                out.write(latexcode + "\n");
                // generate LatexCode Suffix
                out.write("\\end{picture}" + "\n");
                out.write("\\end{document}" + "\n");

                out.close();
            }
            catch (java.io.IOException ioexcep){
              ioexcep.printStackTrace();
            }
        }
        /**
         * remove common divisors for line slope
         *
         * @param width       width of the line.
         * @param height      height of the line.
        */
        public int[] removeCommonDivisor(int width, int height) {
            //System.out.print("in removecommondivisor\n");
            int[] temp = {0,0};
            int divisor, maxdivisor;
            maxdivisor = Math.min(width, height);
            temp[0] = width;
            temp[1] = height;

            for(divisor = 2; divisor <= maxdivisor; divisor++) {
                if ((temp[0] % divisor  == 0) && (temp[1] % divisor == 0)) {
                  temp[0] = temp[0]/divisor;
                  temp[1] = temp[1]/divisor;
                  maxdivisor = maxdivisor/divisor;
                }
            }
            return temp;
          }
        /**
         * adjust slope and its length
```

```
     *
     * @param sx          original units of x coordinate of the line.
     * @param sy          original units of y coordinate of the line.
     * @param width       width of the line.
     * @param height      height of the line.
     */
    public int[] adjustSlope(int sx, int sy, int width, int height) {
        // tmep[] used to store the adjusted units length in x and y
cocordinates,
        // and the adjusted width
        int[] temp = {0,0,0};
        int[] tempslope = {0,0};
        int slopex, slopey;
        float tempsd;
        float sd = maxnum;

        // if slope is bigger than 2 times of maxnum
        if ((float)sy/sx >= (float)2 * maxnum) {
            temp[0] = 0;
            temp[1] = 1;
           temp[2] = height;
            return temp;
        }
        if ((float)sy/sx <= (float)1 /(2 * maxnum)) {
            temp[0] = 1;
            temp[1] = 0;
            temp[2] = width;
            return temp;
        }
        for(slopex = 1; slopex <= maxnum; slopex++) {
            for (slopey = 1; slopey <= maxnum; slopey++) {
                if ((slopex == slopey) & (slopex >1))
                   continue;

                tempslope = removeCommonDivisor(slopex, slopey);
                tempsd = (float)sy/sx - (float)tempslope[1]/tempslope[0];
                if (tempsd < 0)
                   tempsd = - tempsd;
                if (tempsd <= sd) {
                    sd = tempsd;
                    temp[0] = tempslope[0];
                    temp[1] = tempslope[1];
                    temp[2] = width * sy/sx * tempslope[0]/tempslope[1];
                    //temp[2] = width;
                }
             }
        }
        return temp;
     }
    /**
     * main entrypoint - starts the Latex Generator when it is run as an
application
     *
     * @param args the arguments passed to the application on entry.
     */
    public static void main(java.lang.String[] args) {

          try {
                javax.swing.JFrame frame = new ExitingFrame( "Latex
Generator" );
                LatexGenerator lg;
                lg = new LatexGenerator();
                frame.getContentPane().add("Center", lg);
```

```
                    frame.setSize(lg.getSize());
                    frame.setVisible(true);
                        // test functions
                        /*maxnum = 6;
                        int[] test = lg.removeCommonDivisor(220, 115);
                        System.out.print("x= " + Integer.toString(test[0]) +
"y= " + Integer.toString(test[1]) +"\n");
                        int[] temp = lg.adjustSlope(test[0],test[1],220,115);
                        System.out.print("x1= " + Integer.toString(temp[0]) +
"; y1= " + Integer.toString(temp[1]) + "; w= " + Integer.toString(temp[2]));
                        */
            } catch (Throwable exception) {
                    System.err.println("Exception occurred in main() of
latex.main.LatexGenerator");
                    exception.printStackTrace(System.out);
            }
      }
}
```