The Graphlet System (System Demonstration)

Michael Himsolt *

Universität Passau, 94030 Passau, Germany
himsolt@fmi.uni-passau.de

Abstract. Graphlet is a portable, object oriented toolkit for graph editors and graph drawing algorithms, and is the successor of the GraphEd system. Graphlet is based on LEDA and Tcl/Tk. Algorithms can be implemented in C++ and LedaScript, a new scripting language based on Tcl/Tk. The GML format is a portable file format for graphs.

The implementation and visualization of graph algorithms is an important area in research and applications. There is a growing number of systems to support this process. LEDA [8] and GraphBase [7] concentrate on the implementation of a library of graph algorithms. Systems like daVinci [4], DynaDAG [11], D-ABDUCTOR [14] and EDGE [10] concentrate on graph visualization and/or graph editing, while CABRI [3], GD-Workbench [2], GraphEd [6], the Graph Editor Toolkit [9] or VCG [13] combine an editor and a library of algorithms.

The scope of these systems ranges from small and on-purpose systems to large, complex ones. However, the user interfaces of many systems are not easy to customize. Therefore, adding new features or exploring new classes of problems remains difficult. Also, implementing new features usually requires a compiler and a development environment. Furthermore, the application programmer interfaces and file formats of most of these systems are mutually incompatible.

Graphlet is a portable[1], object oriented toolkit for implementing graph editors and graph drawing algorithms. Graphlet is the successor of GraphEd [6]. Besides portability and general overhaul, there are several important improvements from GraphEd to Graphlet. Graphlet is based on three external toolkits, LEDA, Tcl and Tk, which gives us a more powerful design. Graphlet is much more oriented towards rapid prototyping and experiments with algorithms. Graphlet provides a scripting language to make the implementation of tools and customization easier. The GML format provides a powerful and extensible mechanism for writing and exchanging graph files.

* This research is partially supported by the Deutsche Forschungsgemeinschaft, Grant Br 835/6-2, research cluster "Efficient Algorithms for Discrete Problems and Their Applications"

[1] Graphlet is available for machines running Sun OS 4 and 5, Linux, Microsoft Windows NT and Microsoft Windows 95.

1 The structure of Graphlet

Graphlet is structured as follows:

Core system Graphlet's core is the platform independent part of Graphlet. The core provides a large set of base data structures, including a parser for the GML file format (see below) and a universal data structure which allows algorithms to add arbitrary attributes to nodes and edges. Most of the algorithm interface resides in the core.

Tcl interface, LedaScript The Tcl interface is a separate layer, which implements the LedaScript language and the graphics interface. It also implements some performance critical user interface operations. The system is designed such that this interface layer can be replaced or augmented with another interface, e.g. for Java.

Editor The main component of Graphlet is a powerful graph editor, implemented in LedaScript. The user interface could be replaced by another interface written in Tcl/Tk. This feature enables us to use Graphlet as the graph visualization subsystem of other Tcl/Tk programs.

Algorithm modules Graphlet uses a modular design where algorithms are separate plugin modules. Algorithms may be written in C++ using LEDA, in LedaScript or in a mixture of both. They can even be separate programs that communicate with Graphlet. Another option is an interface for algorithms that are written with GraphEd's Sgraph data structure, so we can reuse algorithms from GraphEd.

GML Graphlet defines a new, flexible and portable file format for graphs, GML. External applications which understand GML can communicate with Graphlet and use it as an external viewer.

Due to the modular and layered structure, Graphlet can be used in several configurations, It can be used with the core system only, and then is an extension of LEDA. It can also be used as a scripting language, LedaScript, and as a graph editor.

2 Data Structures and Algorithms

Graphlet uses *LEDA* [8] for the implementation of its data structures and algorithms. LEDA is a library of efficient data structures and algorithms. It provides a wide range of basic data structures, such as strings, lists, sets, queues, dictionaries and hash tables. LEDA supports graphs and contains a number of graph algorithms, especially planarity tests and algorithms for computing a planar embedding. These algorithms help implementing graph drawing algorithms more easily. Another important addition are data structures from computational geometry, which help to implement the coordinate assignment in graph drawing algorithms.

Graphlet adds another layer of data structures to LEDA:

- Support for arbitrary attributes for graphs, nodes and edges. Attributes may be added or removed dynamically at runtime. Methods to store these attributes in files (GML format, see below) are also included.
- Support for graphical display of graphs. Most of this is done on a device independent level, which makes the application programmer interface much easier to use.
- Classes for implementing graph algorithms.

3 Tcl and Tk

Tcl [12] is a universal scripting language. *Tk* is a graphical user interface toolkit, which can be used from Tcl and serves for the implementation of Graphlet's user interface. Unlike other toolkits in its class, Tk supports drawing operations.

Tcl and Tk provide several high level functions such as file management, dynamic loading of modules, process communication and, most important, access to the window system. Another advantage of using Tcl/Tk is that there are a large number of software packages, which add various capabilities to Tcl/Tk. These packages can be used in conjunction with Graphlet to provide extended features. In turn, other programs may use Graphlet as a subsystem.

Since LEDA, Tcl and Tk have been ported to UNIX and Microsoft Windows operating systems, Graphlet has a large base of systems to run on. We use Tcl/Tk for all system dependent parts, which minimizes the effort to port Graphlet to other platforms. Actually, taking the code from UNIX to Microsoft Windows systems required only a few changes, most of them due to compiler incompatibilities.

Graphlet provides two programming interfaces: a C++ interface and a scripting language, *LedaScript*. LedaScript is implemented with Tcl and can be used for rapid prototyping and user interface implementation. Except for a few performance critical routines, Graphlet's user interface is written in Tcl/Tk/LedaScript.

4 LedaScript

LedaScript is an extension of Tcl with support for graph operations. Figure 1 shows a LedaScript sample code. Graphs are created in LedaScript with the command `graph`. Each graph is a Tcl command, which is used to change or retrieve parameters such as the label or the coordinates. This is common practice in Tcl/Tk and implements an object oriented programming interface for graphs. Graph algorithms are available as LedaScript commands. Graphlet provides C++ classes which make it easy to add a Tcl interface to algorithms which are implemented in C++.

Of course, algorithms can also be implemented in LedaScript. LedaScript is powerful enough for rapid prototyping of algorithms. Since LedaScript reduces the edit–compile–link–run cycle to edit–run, it is much easier to test variations and debug algorithms. This may even be done without quitting from the program, since LedaScript provides a interactive shell. Also, LedaScript enables users to write batch scripts that run algorithms on graphs. Finally, since Tcl does not have pointers and can continue even after runtime errors, error handling is much easier than in traditional programming languages.

5 User Interface

We designed Graphlet's user interface using our experiences from Graphed. Figure 2 shows two sample screens. Especially, we enhanced the following:

- By default, there is only one graph per window. GraphEd had the ability to display multiple graphs in a window, which turned out to be more confusing than helpful, both for users and programmers. However, the capability to display more than one Graph in a window remains, which may still be useful in non-interactive applications.
- GraphEd's three types of selections were *single node*, *single edge* or *subgraph*. Graphlet uses a more general model. A selection can be any set of nodes and edges.
- Graphlet offers much wider customization options. For example, self loops or multiple edges can be disabled to avoid undesired structures. All keyboard and mouse bindings are customizable. Graphlet implements modes similar to those in `emacs`. Also, Graphlet works if only one mouse button is available (e.g. on Macintosh systems).
- Graphlet provides a mode to edit labels.
- Graphlet now offers a much wider scale of graphical representation options for nodes and edges, notably colors and line width.
- Many successful features from GraphEd were kept, as the general window layout with a menubar at the top and a toolbar at the right, the menu layout or a the double click to pop up a window to edit object parameters.

Graphlet's user interface will be continuously enhanced and extended. However, it is obvious that Graphlet cannot satisfy all possible future requirements. One major idea is that it is easy to add features to Graphlet. The easiest way to do that is to add some LedaScript code. LedaScript is efficient enough for small or medium tasks, with only a fraction of the time needed for a C++ implementation. For example, it takes less than an hour to add a routine which outputs a graph in a new format.

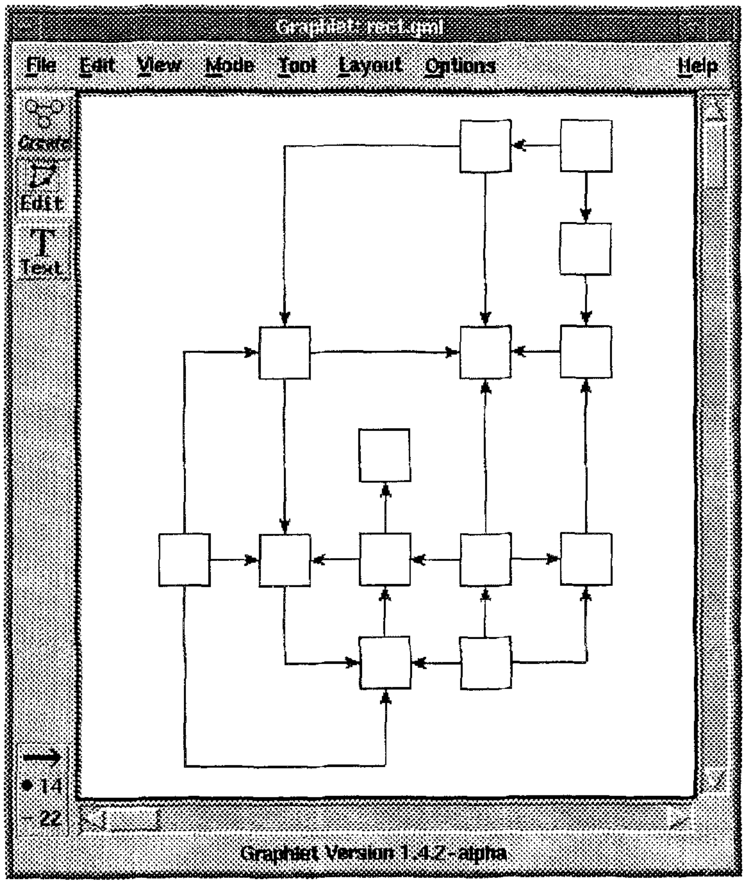
6 The GML File Format

We have developed a new file format for Graphlet, GML. Figure 3 shows a simple graph, a circle of three nodes. This example shows several key issues of GML:

ASCII Representation for Simplicity and Portability. A GML file is 7-bit ASCII file. This makes it simple to write files through standard routines.

```
#
# create a canvas ``.c'' (Tcl/Tk drawing area)
#
canvas .c
pack .c

#
# create a graph
#
set g [graph]

#
# draw the graph on canvas .g
#
$g canvases .c

#
# create a node "a" at position (21,21)
#
set n1 [$graph create node]
$g nodeconfigure $n1 -label "a"
$g nodeconfigure $n1 -x 21
$g nodeconfigure $n1 -y 21

#
# create a node "b" at position (42,42)
#
set n2 [$graph create node]
$g nodeconfigure $n2 -label "b"
$g nodeconfigure $n2 -x 42
$g nodeconfigure $n2 -y 42

#
# create an edge from node n1 to node n2
#
set e [$graph create edge source $n1 target $n2]

#
# now draw the graph
#
$g draw
```

Fig. 1. Sample LedaScript code



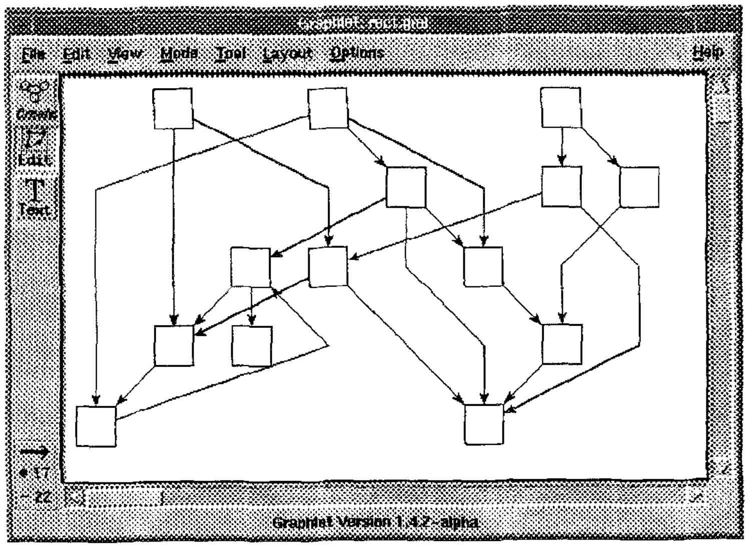


Fig. 2. Graphlet's user interface

```
graph [
  comment "This is a sample graph"
  directed 1
  IsPlanar 1
  node [ id 1 label "Node 1" ]
  node [ id 2 label "node 2" ]
  node [ id 3 label "node 3" ]
  edge [ source 1 target 2 label "Edge from node 1 to node 2" ]
  edge [ source 2 target 3 label "Edge from node 2 to node 3" ]
  edge [ source 3 target 1 label "Edge from node 3 to node 1" ]
]
```

Fig. 3. GML description of a circle of three nodes

Parsers are easy to implement, either by hand or with standard tools like `lex` and `yacc`. Also, since GML files are text files, they can be exchanged between platforms without special converters.

Simple Structure. A GML file consists entirely of hierarchically organized tag-value pairs. A tag is a sequence of alphanumeric characters, such as `graph` or `id`. A value is either an integer, a floating point number, a string or a list of tag-value pairs enclosed in square brackets.

Extensibility & Flexibility. GML can represent arbitrary data, and it is possible to attach additional information to any object. For example, the graph in Figure 3 adds an `IsPlanar` attribute to the graph.

Clearly, this can lead to a situation where one application adds some data which cannot be understood by another application. Therefore, applications are free to ignore data they do not understand, except for a basic set which describes graphs. They should, however, save unknown data and write them back.

Representation of Graphs. Graphs are represented by the tags `graph`, `node` and `edge`. The topological structure is modeled with the node's `id` and the edge's `source` and `target` attributes: the `id` attributes assign unique numbers to nodes, which are then referenced by `source` and `target`.

Consistency Consider the following situation: a file includes information of some graph theoretical property, say the existence of a Hamiltonian circle. It is easy to see that this information may become invalid if an edge is removed, but not if an edge is added. However, a program that does not know about Hamilton cycles will not be able to check and guarantee this property. Another example is if a node is moved, then the coordinates of its adjacent edges must be updated. Even more, a hypothetical attribute `IsDrawnPlanar` might become invalid when node or edge coordinates are changed.

As these examples show, both changes in the structure and in the values of attributes can falsify other attributes. GML therefore specifies that if the name of a tag starts with a capital letter, then it should be considered "unsafe" and discarded whenever changes have occurred in the graph or the attributes, unless the program knows how to deal with the case. A simple strategy could be to not to load unsafe information at all if the program is going to change the graph. This feature gives programs the opportunity to add critical data, and makes at the same time sure that everything remains consistent.

7 Conclusion

The Graphlet system is currently under development at the University of Passau. Versions for several UNIX systems, Microsoft Windows NT and Microsoft Windows 95 are available.

References

1. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Algorithms for drawing graphs: an annotated bibliography.* Comput. Geom. Theory Appl. **4** (1994) 235–282
2. Buti, L., Di Battista, G., Liotta, G., Tassinari, E., Vargiu, F., Vismara, L.: GD-Workbench: A System for Prototyping and Testing Graph Drawing Algorithms. In Lecture Notes in Computer Science **1027**, pp. 111–122 (1995)
3. Carbonneaux, Y., Laborde, J.-M., Madani, M.: CABRI-Graph: A Tool for Research and Teaching in Graph Theory. In Lecture Notes in Computer Science **1027**, pp. 409-418 (1995)
4. Fröhlich, M, Werner, M.: Demonstration of the Interactive Graph-Visualization System da Vinci. In Lecture Notes in Computer Science **894**, pp. 266–269. (1994)
5. Information on Graphlet is available on the world wide web at `http://www.fmi.uni-passau.de/Graphlet`.
6. M. Himsolt: GraphEd: A Graphical Platform for the Implementation of Graph Algorithms. In Lecture Notes in Computer Science **894**, pp. 182–193. (1994)
7. Knuth, D.E.: *The Stanford GraphBase: A Platform for Combinatorial Algorithms.* Stanford University (1993)
8. Mehlhorn, K., Näher, S.: LEDA: A library of efficient data types and algorithms, Comm. ACM 38(1), pp. 96–102 (1995).
9. Madden, B., Madden, P., Powers, S., Himsolt, M.: Portable Graph Layout and Editing, In Lecture Notes in Computer Science **1027**, pp. 385–395 (1995)
10. Newberry Paulisch, F.: The Design of an Extendible Graph Editor, Lecture Notes in Computer Science **704** (1993)
11. North, S.: Incremental Layout in DynaDAG, In Lecture Notes in Computer Science **1027**, pp. 409-418 (1995)
12. Ousterhout, J.: Tcl and the Tk Toolkit. Addison-Wesley (1994).
13. Sander, G.: Graph Layout through the VCG Tool, In Lecture Notes in Computer Science **894**, pp. 194–205. (1994)
14. Sugiyama, K., Misue, K.: A Generic Compound Graph Visualizer/Manipulator: D-ABDUCTOR, In Lecture Notes in Computer Science **1027**, pp. 500–503 (1995)