Demonstration of Advanced Layout of UML Class Diagrams by SugiBib

Holger Eichelberger
chair of computer science II
Würzburg University
Am Hubland, 97074 Würzburg, Germany
eichelberger@informatik.uni-wuerzburg.de

Jürgen Wolff von Gudenberg
wolff@informatik.uni-wuerzburg.de

1 Introduction

The Unified Modeling Language (UML) [7] has become the standard for specifying object-oriented software systems. Some of the tools are primarily designed to work on a direct mapping between the design diagrams and the software and vice versa. Since understanding the software is usually using more abstract concepts than those defined in programming languages, restriction of the UML used in these tools is not permissible. On the other side, visualization of changes to the software implementation and design documents require sophisticated layout algorithms. In [2] we have shown that most of the tools are not sufficient in that field.

2 Layout of UML class diagrams

Obviously a UML class diagram can be described as graph $G = (V, E)$ with nodes V and edges E. V can then be partitioned into nodes of different types: packages and classes may be nested (nested nodes might be structured according to different criteria like coupling [3], subsystems [7] or component classifiers in the new UML 2.0), annotations can be attached to all model elements, association classes (classes attached to an association can be part of other associations or generalization relations) and natural or arbitrary clusters (like design patterns or higher associations). According to [1, 3] E should be partitioned into a set of hierarchical edges E_H and a set of non-hierarchical edges E_N using heuristics or user defined preferences. Different types of edges have to be respected: Generalizations, associations, aggregations, compositions and dependencies with different textual and symbolic adornments. Adornments of edges may not overlap adornments of other edges or node boxes. Constraints concerning two or more associations (denoted by a dashed line connecting the associations) have to be laid out.
The layout calculated by the algorithm should be optimized for a clear representation of a software design diagram, easy to read, understandable and therefore a large set of criteria for optimal readability according to semantical reasons must hold beside usual graph drawing criteria like overall number of edge crossings and bends [1, 3].
The layout algorithm is clearly explained in [1, 5].

3 Architecture of the framework

SugiBib is a pure Java framework which primarily was designed to implement a general, highly configurable, component-based version of the Sugiyama algorithm [10]. The components can be combined in different sequences to implement other layout algorithms. Because of the component architecture information hiding is preserved between two consecutive steps. Nodes and edges of the framework are parametrized by their individual graphical information.
SugiBib was instantiated to represent UML class diagrams and provides interactive frontends in AWT and Swing as well as online and batch rendering servers. Advanced features of UML class diagrams like association classes and annotations are treated by an extension of the Seemann algorithm [9].
Currently SugiBib accepts input in UMLscript [4], a programming language for object oriented design. The standard XML Metadata Interchange format XMI [8] is extended by different vendors with their proprietary notation for layout information. Therefore a general import of XMI into SugiBib can be realized only by extensive XSLT preprocessing. The current implementation is prepared for the XMI version produced by MagicDrawUML (`www.nomagic.com`). As an intermediary format while processing XMI, the XML version of UMLscript called XUMLscript is produced. Therefore SugiBib is able to accept plain XUMLscript as input, too. Additionally we are working to read diagrams in the new UML 2.0 Diagram Interchange format [6] XMI[DI] or XMI[UML+DI], respectively.
The output format we produce is the laid out graph and as a proprietary textual XML representation of the internal

graph to be postprocessed by diff, e.g., currently for debugging purpose only. Output of standardized XML (XMI[DI] or XMI[UML+DI]) is planned for the near future.

Current information on SugiBib can be obtained from `www.sugibib.de`.

References

[1] H. Eichelberger. Aesthetics of class diagrams. In *Proceedings of the First IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 23–31. IEEE, 2002.

[2] H. Eichelberger. Evaluation-report on the layout facilities of UML tools. TR 298, Institut für Informatik, Universität Würzburg, jul 2002. Institut für Informatik, Universität Würzburg.

[3] H. Eichelberger. Nice class diagrams admit good design? In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 159–ff. ACM Press, 2003.

[4] H. Eichelberger and J. W. von Gudenberg. UMLscript sprachspezifi kation. TR 272, Institut für Informatik, Universität Würzburg, feb 2001. Institut für Informatik, Universität Würzburg.

[5] H. Eichelberger and J. W. von Gudenberg. On the visualization of Java programs. In S. Diehl, editor, *Software Visualization, State-of-the-Art Survey*, volume 2269 of *Lecture Notes in Computer Science*, pages 295–306. Springer, Springer, 2002.

[6] OMG. UML 2.0 Diagram Interchange. Second (Final) Revised Submission, OMG document number ad/2002-12-20, Version 1.0, January 6, 2003, via `http://www.xml-strategie.de/files/UML2DIRevSub.pdf`

[7] OMG. Unifi ed Modeling Language Specifi cation. Version 1.5, March 2003 via `http://www.omg.org`.

[8] OMG. XML Metadata Interchange (XMI). Version 1.1, via `http://cgi.omg.org/docs/ad/99-10-02.pdf`.

[9] J. Seemann. Extending the sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. *Lecture Notes in Computer Science*, 1353:415–423, 1997.

[10] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109–125, Feb. 1981.

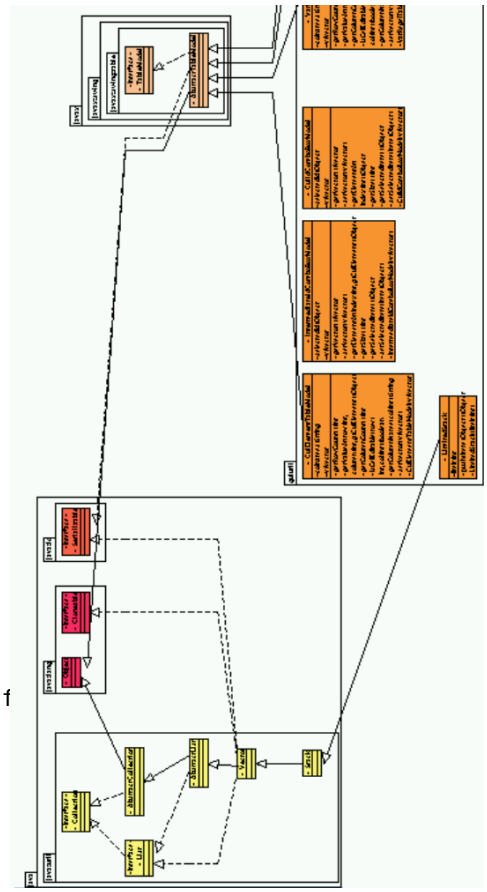

Figure 1. A part of the static structure of a GUI application drawn by SugiBib.