UML Class Diagrams - State of the Art in Layout Techniques

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

Abstract

Even if the standard for specifying software, the Unified Modeling Language, is known in different versions to everybody, CASE tool vendors did not implement all basic features. Even with class diagrams, many features of the standard are ignored. Applying the layout algorithms of these CASE tools to the user defined diagrams, usually horrible results are produced, because state-of-the-art techniques in drawing these diagrams are not respected by the vendors, too.
In this paper we give an overview on the current UML tool implementations, the research in the field of drawing class diagrams automatically and the efforts in convincing the community of an agreement on basic aesthetical principles for UML class diagrams in order to simplify reading and understanding of standardized visualization of static aspects of software.

1. Introduction

In software engineering the Unified Modeling Language (*UML*) has advanced as the standard for graphically specifying static and dynamic aspects of software. In 2003 the Object Management Group (*OMG*) released the version 2.0 of the UML. The number of different diagram types increased from 9 to 13, new types of diagrams have been introduced and some of the older types have significantly increased in complexity. In [4] we compared the implementations of the UML features and the automatic layout facilities of 42 current computer aided software engineering (*CASE*) tools and agree to the informal statements of others, that most of the CASE tools have not reached the conformity of UML versions older than 1.3 Regarding the automatic layout features, the tools usually produce horrible results by transforming the layout and implicitely and accidentally changing the semantics of the complete diagram.
In the next section we summarize the results of the CASE tool overview, then we mention the set of basic aesthetic principles which we advocate as a set of basic rules to be respected by software engineers as well as tool vendors and finally we present the results of state-of-the-art graph drawing algorithms for class diagrams. References to other work are included in the individual sections.

2. UML Tools - an Overview

"For more then one decade vendors have under delivered the promises of object modeling technologies. As a result, object modeling tools are in disrepute in many development organizations."[10] "As for modeling tools, the author knows of none that fully implements the UML 1.1 semantics and notation (adopted three years ago), let alone one that completely or correctly implements the current UML 1.3 specification (which was adopted a year ago)" [10, october 2000]. As shown in [4] even in July 2002 the situation on UML conformity and implementation of layout algorithms was nearly the same.
For our test, we tried to use the diagram shown in figure 1 as input to the tools and then applied the automatic layout mechanisms if present. First problems arised while trying to define the test diagram within the tool:

1. Most tools are too implementation-specific. Model elements visualizing abstract concepts which cannot be directly realized in a programming language like association classes, higher associations, constraints and even comments are not present. Since most tools are not able to correctly produce code for the different responsibilities of associations, these tools should omit at least associations, too.

2. A lot of tools implement packaging mechanisms as logical view only. It is not possible to use classes within packages (extremely useful when visualizing coupling and cohesion or applying the facade design pattern). The alternative concept, the anchor edge, is

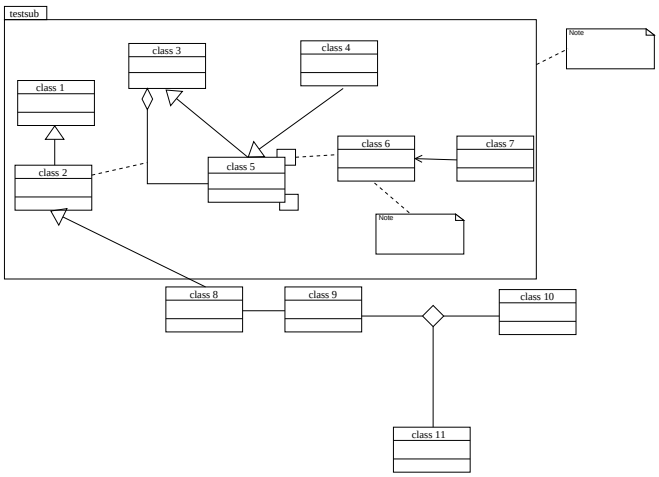

Figure 1. The test diagram in [4] which shows classes within a package, relations across package borders, two association classes in further relations, a ternary association, two nodes and two reflective associations.

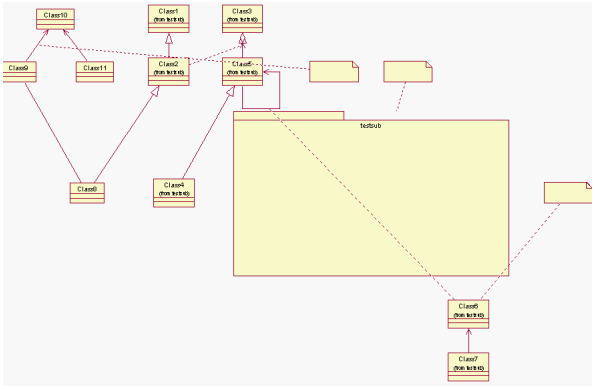

Figure 2. Rational AnalystStudio 2002.05.20

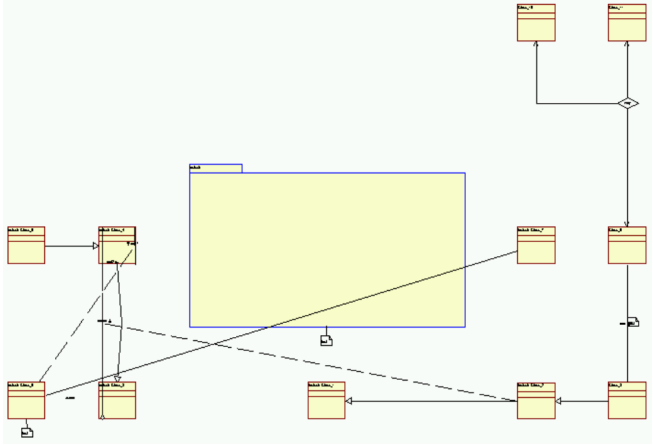

Figure 3. Popkin System Architect 8.5.16

usually not present. Other package-like elements like subsystems or models are not present in most tools, either.

3. Nearly each tool implements its own input philosophy - as a users wish to the vendors we propose the specification of a user interface standard for CASE tools in order to increase usability and to simplify the use of multiple tools.

A tool which claims to be conformant to the UML in any given version should realize the complete specification without any restrictions!
The result of applying the automatic layout mechanism on individual tools is depicted in the figures 2 to 5. Applying the layout mechanism twice or more times sometimes produced different results. Screenshots of the entire screen are shown in [4].

3. Rules for the Layout of a Class Diagram

Unfortunately most layout algorithms on class diagrams do not adhere to any aesthetic principles. Different surveys [13, 14, 15] on class diagrams have been published but most of them rely on aesthetical principles taken from graph drawing without respecting the underlying semantics. The latest results show that there is not a uniform user preference on the regarded aesthetic principles.
In [3, 6] we discussed basic issues for semantic based aesthetic criteria in order to provide intuitive rules for drawing class diagrams. The rules are validated by references to the UML specification, to HCI results and results from software engineering. Some of the rules are mentioned below in a compressed form:

1. Enforce hierarchy as the most appropriate ordering criterion for edges in a class diagram. Since software engineers are used to thinking hierarchically, containment, inheritance, realization, aggregation, composition and user defined hierarchies should be taken into account. Even the latest publications on other layout algorithms [6, 9] adhere to that principle.

2. Respect spatial relationships to encode coupling, cohesion and importance of parts of the diagram.

3. Visualize the natural clustering of nodes according to semantical reasons like containment, n-ary associations and patterns.

4. Avoid crossings and overlappings of model elements.

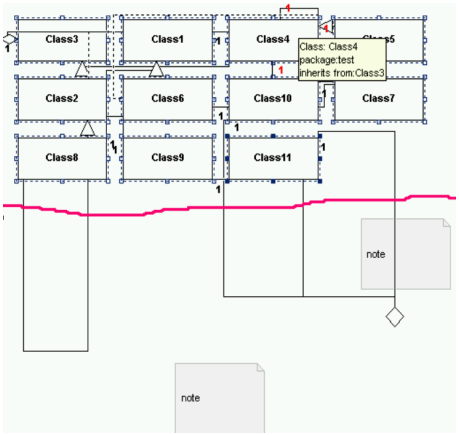

Figure 4. TNI OpenTool 3.2.15 (poor layout award) - part of the diagram was cutted of due to space limitations (red line)

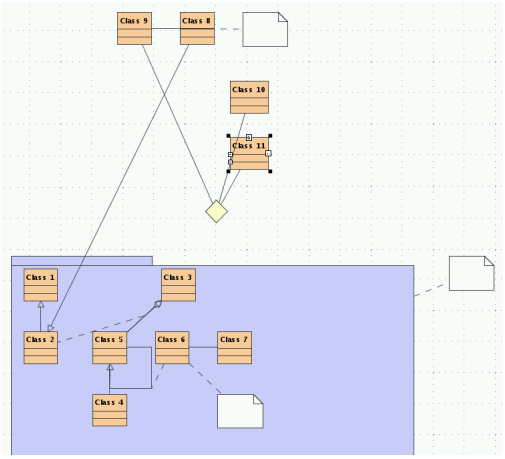

Figure 5. NoMagic MagicDrawUML 7.0 beta (best layout and UML conformity award)

5. Center position of selected nodes (n-ary associations, pattern nodes).
6. Respect the vicinity of association classes, notes and constraints.
7. Clearly assign adornments to edges and reflective associations to the connected classes.
8. With the minimum priority respect other graph drawing criteria.

Intuitively these rules lead to readable diagrams and therefore can reduce the cost of communication when interchanging software development diagrams. Additionally these rules can be used as definition of a measurement framework for the objective comparison of tool features and layout algorithms. Unfortunately validating these rules by user experiments is a hard task: high degree of freedom induced by the number of criteria, the need for qualified and experienced software engineers instead of UML-novice students as users to be questioned and low UML tool support so far since no standardized diagram exchange format for UML diagrams was defined.

4. Drawing a Class Diagram

For other UML diagrams like activity diagrams (flow layout) and state charts [2, 1] appropriate algorithms have been proposed but unfortunately these algorithms are usually not implemented in CASE tools so far. The large variety of model elements available for use in UML class diagrams are not respected by most of the algorithms proposed so far [6, 9, 8, 16, 17]. These algorithms mainly focus on classes, inheritance relations and association relations but not on nested package and class structures and more sophisticated model elements like association classes, higher associations or constraints.
The following listing is a brief description of our current (revised) approach. Detailed descriptions and relations between the algorithm and the aesthetic rules mentioned in section 3 can be found in [3, 5, 7].

1. Identify a pseudohierarchy by heuristics or by respecting a user defined hierarchy.
2. Perform a semantic ordering to release implicit dependencies between the sequence of definitions of the model elements in the input and the layout result.
3. Insert containment relations of model elements as hierarchical edges.
4. Compress association classes and their edge connector nodes into compound nodes.
5. Convert annotations and connected model elements to compound nodes.
6. Remove reflexive associations in order to simplify the implementation. Represent the edge information within the connected classes in order to be drawn as edges later on.
7. Transform the graph to an acyclic graph.
8. Guarantee a virtual root.
9. Calculate the ranking of the hierarchically connected nodes in one step, calculate the layer positions of only

non-hierarchically connected nodes in a second step. Optimize the layered structure of the graph for UML class diagram layout.

10. Calculate edge crossing minimization on hierarchical and non hierarchical edges by an incremental crossing reduction approach. Respect cluster and containment relations in this step.

11. Remove containment information.

12. Calculate the coordinates of nodes and edges. Contained model elements are treated in the same step in order to respect non-hierarchical edges.

13. Expand compound nodes for association classes.

14. Expand and layout notes.

Preserving the the mental map [11] is extremely important when iteratively changing class diagrams while analysis and design phase as well as in roundtrip-engineering. To implement incremental layout, phases for compressing and preserving the positions of unchanged nodes can be inserted at the beginning and the end of the algorithm. As

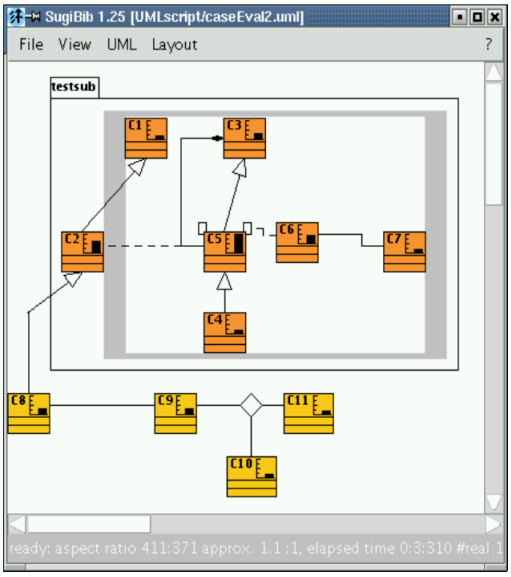

Figure 6. The test diagram from [4] drawn by SugiBib. Visualization of coupling and cohesion is enforced (the shaded area is a non-UML feature and drawn for demonstration purpose only), the relative complexity of classes is shown as a decorative stereotype. Notes are not implemented so far.

a proof of concept the algorithm has been implemented as

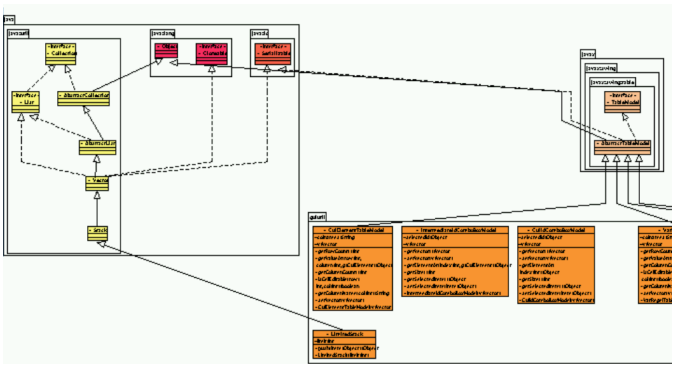

Figure 7. Part of the relations between different Java library packages and a simple graphical user interface implementation.

a prototypical framework written in pure Java. The figures 6 to 8 show different results produced by our algorithm. Current information on SugiBib can be obtained from `www.sugibib.de`.

4.1. Conclusions

In this paper we have shown that the current implementations of CASE tools neither implement an appropriate version of the UML nor provide layout algorithms which represent the state-of-the-art in drawing class diagrams. The first restricts the user to software models which are implementation-specific and far away from the desired level of abstraction. Defining an own subset of the UML restricts the usability and conformance to future standards like MDA [12]. The second requires more manual adjustments, requires more time and disables effective engineering techniques like reverse-engineering and roundtrip-engineering. Since designing layout algorithms is not one of the core competences of a CASE tool vendor, it is advisable to disable their individual algorithms (especially if the undo function is not fully functional) or to implement a warning message as long as more appropriate algorithms are implemented.

Since different algorithms may produce different drawings which might be nice to different individuals, a UML based standard for diagram layout and interchange aesthetics should be proposed. We have shown a subset of our proposal for aesthetic principles for class diagrams. As a consequence of applying these principles the readablility and understandability is enhanced.

Finally we have mentioned the problems of the other approaches to realize the automatic layout of class diagrams. Most of the other algorithms extend the topology-shape-metrics approach without a description on how to realize

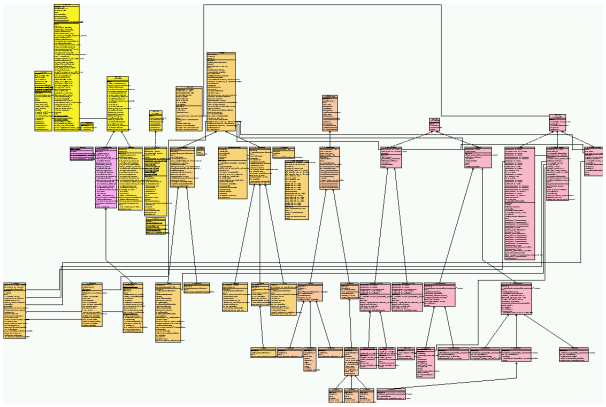

Figure 8. A class hierarchy (respecting inheritance relations). The nodes are colored and grouped according to package containment without showing the packages itselves.

the more complex situations which arise from using more sophisticated model elements. Sometimes other hierarchical or even force-directed approaches are mentioned in literature but usually only the layout of classes and simple relations is respected.

Since most tool vendors do not fully implement older versions of the UML it will be a long road for a complete realization of the new UML version 2.0 Since the complexity of most diagrams has increased, most of the traditional algorithms (only working on structure not on semantics) like flow-layout for activity diagrams are not appropriate any more. For class diagrams at least component classifiers (class-like model elements furtherly structured by classes or components) and the graphical representation for provided and required interfaces/components have to be respected by a layout algorithm. Since our algorithm is capable of working on nested and structured elements it can easily be updated. Even if it is known, that adding more constraints to an algorithm the runtime is increased and low-quality results (if even a result can be computed) is the risk we believe, that this can be respected by introducing additional criteria into the new edge crossing reduction algorithm. Additionally a slight update to our set of aesthetic principles is necessary.

References

[1] R. Castello, R. Mili, and I. G. Tollis. Automatic layout of statecharts. *Software – Practice and Experience*, 32(1):25–55, 2002.

[2] R. Castello, R. Mili, and I. G. Tollis. A framework for the static and interactive visualization of statecharts. *Journal of Graph Algorithms and Applications*, 6(3):313–351, 2002.

[3] 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, IEEE, 2002.

[4] 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.

[5] H. Eichelberger. Sugibib. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, 9th International Symposium, GD '02*, volume 2265 of *Lecture Notes in Computer Science*, pages 467–468. Springer, Springer, 2002.

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

[7] 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.

[8] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel. Caesar automatic layout of UML class diagrams. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, 9th International Symposium, GD '02*, volume 2265 of *Lecture Notes in Computer Science*, pages 461–462. Springer, Springer, 2002.

[9] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel. A new approach for visualizing UML class diagrams. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 179–188. ACM, ACM Press, 2003.

[10] C. Kobryn. Modeling components and frameworks with UML. *Communications of the ACM*, 43(10):31–38, 2000.

[11] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[12] OMG. Model driven architecture specifi cation. Version 1.0, May 2003 via `http://www.omg.org`.

[13] H. Purchase, J.-A. Allder, and D. Carrington. User preference of graph layout aesthetics: A UML study. In J. Marks, editor, *Graph Drawing - 8th Internation Symposium*, volume 1984 of *Lecture Notes in Computer Science*, pages 5–18. Springer, Springer, 2001.

[14] H. Purchase, J.-A. Allder, and D. Carrington. Graph layout aesthetics in UML diagrams: User preferences. *Journal of Graph Algorithms and Applications*, 6(3):255–279, 2002.

[15] H. Purchase, M. McGill, L. Colpoys, and D. Carrington. Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study. *Proceedings of the Australian Symposium on Information Visualisation*, 9, 2001.

[16] D. Spinellis. On the declarative specifi cation of models. *IEEE Software*, 20(2):94–96, 2003. March/April.

[17] R. Wiese, M. Eiglsperger, and M. Kaufmann. yfi les: Visualization and automatic layout of graphs. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, 9th International Symposium, GD '02*, volume 2265 of *Lecture Notes in Computer Science*, pages 453–454. Springer, Springer, 2002.