GoVisual for CASE Tools Borland Together ControlCenter and Gentleware Poseidon – System Demonstration

Carsten Gutwenger, Joachim Kupke, Karsten Klein, and Sebastian Leipert

Research Center caesar
Ludwig-Erhard-Allee 2
D-53175 Bonn, Germany
{gutwenger|kklein|kupke|leipert}@caesar.de

Abstract. The Unified Modeling Language (UML) has become the software industry's standard notation for representing software architecture and design models. UML diagrams play an important role in the engineering and re-engineering processes of software systems. Of particular interest from the Graph Drawer's perspective are UML class diagrams whose purpose is to display class hierarchies (generalizations), associations, aggregations, and compositions in one picture. The combination of hierarchical and non-hierarchical relations poses a special challenge to a graph layout tool. We present an implementation of our technology within well-known modelling tools.

1 Introduction

The Unified Modeling Language (UML) by Booch, Rumbaugh and Jacobson (see [1]) provides a mainly graphical notation to represent the artifacts of a software system. The most important UML diagram type for software architects is the *UML class diagram* consisting of classes represented by rectangular regions containing the class name, attributes and operations of the class, and different kinds of relationships between classes that are represented as lines. Since these diagrams are a means of communication between customers, developers, and others involved in the software engineering and re-engineering process, it is critical that the diagrams present information clearly. An appropriate layout of these diagrams can assist in achieving this goal (see [7]).

We have developed an approach for automatically laying out UML class diagrams in an orthogonal fashion (see [6]). Our approach distinguishes between two kinds of relationships: *generalizations* representing inheritance in class hierarchies and *associations* including *aggregations* and *compositions*. Inheritance hierarchies are emphasized in several ways:

- generalizations belonging to the same hierarchy are drawn following the same direction,
- nesting of different hierarchies within each other is avoided,

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 123–128, 2004.

- generalizations leading to the same super class join prior to reaching the super class,
- highlight the various class hierarchies by different colors,
- and highlight the generalizations by color.

For a clear visualization of the specific combination of hierarchical and non-hierarchical components, we put special emphasis on meeting a balanced mixture of the above criteria plus the following aesthetic criteria: crossing minimization, bend minimization, orthogonality, and horizontal labels.

The layout functionalities are provided as plug-ins to achieve a tight integration into existing tools (in this case, software development tools). This allows the user to work within a familiar environment without concerning about an extra user interface for graph layout and the data exchange between the different tools. Following this strategy, we expect an increasing user acceptance for automatic layout algorithms.

2 The Plug-in Philosophy

Modern development tools typically come with a graphical software modelling interface. In order to support developers and designers in the software development process, the automatic layout component has to be accessible within their development environment. Therefore we integrated the graph drawing technology as plug-ins into existing CASE tools. The layout component supports the software engineer to manage the software projects by arranging the different model views from activity diagrams to class diagrams. This approach combines the core competencies of both the CASE tool provider and the layout tool provider.

This demonstration presents a seamless integration of our technology into the following two development tools (*applications* for short in the following):

- Gentleware Poseidon for UML ([5]), a low-cost UML CASE tool. It evolved from the Open Source project ArgoUML and has a large number of installations (currently over 400.000).
- Borland Together ControlCenter ([2]), an enterprise development platform that combines application design, development, and deployment.

After installing the plug-in, a new menu called *GoVisual* is available within the application. The menu comes along with new graph layout tool buttons (see Fig. 1) that give access to various diagram layout algorithms, including the orthogonal UML layout algorithm as described in [6]. The user is enabled to apply a layout algorithm to a currently active diagram. Moreover, the user may adjust options settings of the layout algorithms to meet his aesthetic preferences.

The orthogonal UML layout algorithm is part of the large GoVisual framework of layout algorithms and data structures for the automatic layout of diagrams. GoVisual is an object-oriented C++ class library. Our plug-ins work by accessing the GoVisual API. As both applications are pure Java, we use our API's Java Native Interface to access the library. The plug-in core and the

user interface components are written in Java using the application's plug-in interfaces.

To make the technology accessible to the end-user, a GoVisual menu is installed in the main menu bar of the applications. In addition, there are GoVisual layout buttons inserted into the tool bars (see Fig. 1) to allow direct access to the new layout functionality.

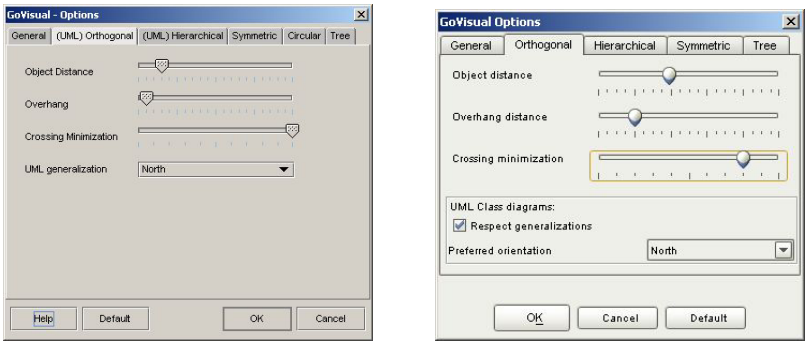
(a) ControlCenter (b) Poseidon

Fig. 1. Easy access through GoVisual menu and toolbar buttons.

Since layout algorithm parameters are usually hard to understand for users that do not have an appropriate graph drawing background, we decided to use a simple option interface that prevents the software-developer from being confronted with these parameters. The options are presented in a user-friendly way, i.e., the parameters are hidden behind an easy-access interface. The quality of the crossing minimization procedure, for example, can be selected using a slider bar, though, behind the scenes, the selection is translated into a distinguished graph algorithms setting (see Fig. 2).

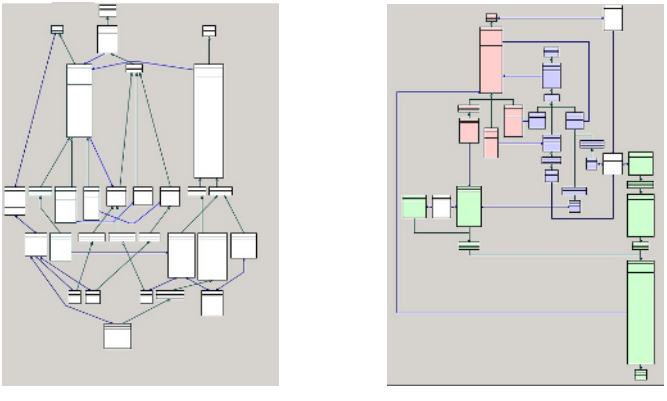
3 Automatic Layout vs. Built-in Techniques

Automatic layout functionalities should give the user a clear and concise view of the software model. It integrates into the user's familiar environment as described in Sect. 2, and it offers superior layout capabilities compared to the layout functions given by the integration platform. Figure 3 shows a sample UML class diagram. The layout in 3(a) has been created with the automatic layout functionality of ControlCenter itself. It contains 13 crossings, including two crossings of generalizations and nine crossings between generalizations and associations. The diagram is difficult to read and does not reveal enough information to the user in order to easily understand the structure of the software project. It is especially difficult to identify the inheritance hierarchies. The layout given in Figure 3(b) has been computed by the GoVisual plug-in. It shows only one crossing of two associations. Moreover, the project's structure becomes visible at glance. Especially, the three inheritance hierarchies are easy to recognize, since the generalizations within the same class hierarchy are drawn in the same direction and the three class hierarchies are highlighted by different colors.



(a) ControlCenter (b) Poseidon

Fig. 2. GoVisual option pages.

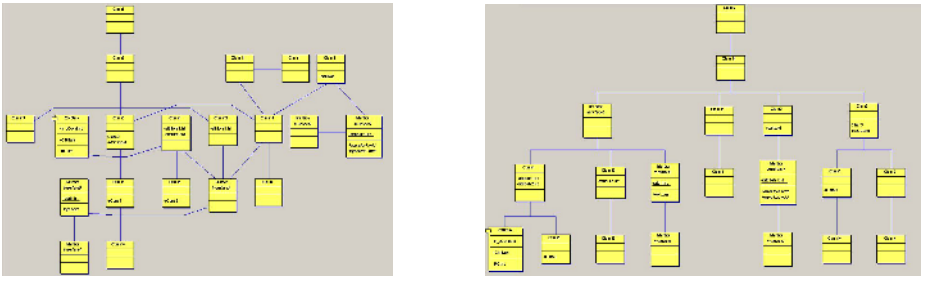


(a) Automatic layout by Together ControlCenter.

(b) Automatic layout by GoVisual.

Fig. 3. A Sample UML class diagram automatically laid out by Together ControlCenter (a) and by GoVisual orthogonal UML layout (b). The GoVisual layout also automatically highlights the three inheritance hierarchies using different colors.

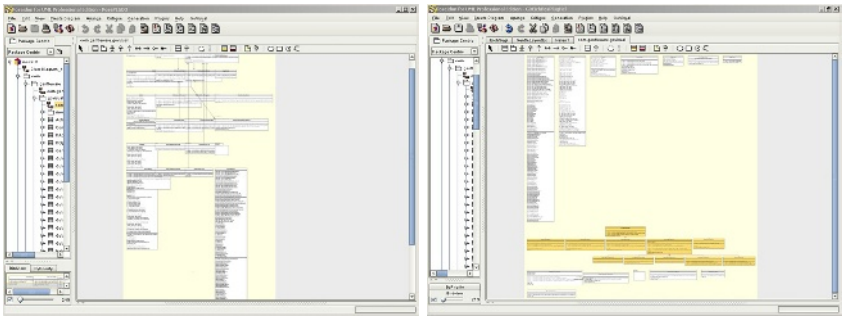
Our techniques clearly outperform the built-in automatic layout algorithms (see Figs. 3, 4 and 5). The application's built-in standard layout algorithms may even hide the diagram structure to an extent where it is nearly unreadable (see Fig. 4). The Gentleware Poseidon for UML application does not provide an advanced automatic layout capability (see Fig. 5 (a)).



(a) Automatic layout by Together ControlCenter.

(b) Automatic layout by GoVisual

Fig. 4. A Sample UML class diagram with a tree structure automatically laid out by Together ControlCenter (a) and by GoVisual Tree Layout (b).



(a) Random layout by Poseidon.

(b) Automatic layout by GoVisual.

Fig. 5. A Sample UML class diagram with no layout given by Poseidon for UML (a) and by GoVisual UML Orthogonal Layout (b).

Some of the GoVisual layout styles are specialized for a certain type of diagram, e.g. the UML class diagram layout styles, others are variations of standard algorithms that can be applied to all kinds of available (UML) diagrams.

The GoVisual plug-ins provide the following layout styles:

- UML Orthogonal Layout: The unique GoVisual layout technique (see [6]).
- UML Hierarchical Layout: A hierarchical layout especially suited for UML Class Diagrams (based on [8]).
- Orthogonal Layout having a focus on the minimization of crossings between edges and the minimization of the number of bends of the edges (based on [9]).
- Symmetric Layout using an energy-based layout technique (based on [4]).
- Tree Layout for the visualization of non-circular structures (based on [3]).

All layout styles provide a set of drawing and optimization algorithms that can be combined to a layout algorithm that provides the best results for the user within the chosen layout style. Furthermore a variant of parameters can be manipulated within each style.

More information about the plug-ins and GoVisual software in general can be found at `http://www.oreas.com`.

References

1. G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide*. Addison Wesley Longman, 1999.
2. Borland Software Corporation. http://www.borland.de/together/controlcenter/.
3. C. Buchheim, M. Jünger, and S. Leipert. Improving Walker's algorithm to run in linear time. In M. Goodrich, editor, *Graph Drawing (Proc. GD 2002)*, volume 2528 of *LNCS*, pages 344–353. Springer-Verlag, 2002.
4. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.
5. Gentleware AG. http://www.gentleware.de/products/.
6. 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 1st ACM Symposium on Software Visualization (SoftVis 2003), June 11-13, 2003, San Diego, CA*, 2003. To appear.
7. H. Purchase, J.-A. Allder, and D. Carrington. User preference of graph layout aesthetics: A UML study. In J. Marks, editor, *Graph Drawing (Proc. GD 2000)*, volume 1984 of *LNCS*, pages 5–18. Springer-Verlag, 2001.
8. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.
9. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.