KIELER: Building on Automatic Layout for Pragmatics-Aware Modeling

Miro Spönemann, Christoph Daniel Schulze, Christian Motika, Christian Schneider, and Reinhard von Hanxleden
Dept. of Computer Science, Christian-Albrechts-Universität zu Kiel, Germany
{msp,cds,cmot,chsch,rvh}@informatik.uni-kiel.de

***Abstract*—Automatic layout is a key enabler for *pragmatics-aware modeling*, which refers to model-driven engineering with designer productivity in mind. This showpiece introduces an infrastructure for the integration of graph layout libraries and their configuration with regard to graphical views of modeling applications.**

I. Introduction

Graphical modeling languages, such as the UML, are an established means to design complex systems. However, the actual process of creating and maintaining graphical models is often rather tedious in practice. In particular, the standard paradigm of drawing graphical models manually typically results in a difficult trade-off between model degradation and continuous manual effort. Klauske estimates the effort required for manually creating layouts at 25% of the time spent for modeling automotive systems [1, p. 4].

Pragmatics-aware modeling [2] aims to help designer productivity by allowing the modeler to focus on the model, and having the modeling tool create customized views automatically. This is enabled by the *transient views approach*, which allows the efficient, automatic creation of graphical views, customized to specific requirements [3].

Contribution: We here present new techniques for the integration of automatic layout in graph-based modeling tools. This is an essential building block upon which the transient views approach as well as other methods addressing pragmatics-aware modeling are built. An essential aspect is the set of interfaces for the selection and configuration of layout algorithms (*meta layout*). We introduce four types of configuration interfaces that are implemented in the Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER)[1].

Related work: Of course there are already many applications that integrate automatic layout for visual languages. However, most of these tools are limited to only few, sometimes rather primitive layout algorithms, and hardly offer any configuration parameters. Maier and Minas propose to configure automatic layout with a pattern-based approach [4]. The focus of this paper, in contrast, is to find a configuration approach that integrates well with the capabilities of existing libraries and makes them available for a wide range of modeling applications.

II. An Infrastructure for Meta Layout

Numerous approaches for the automatic layout of graphs have been developed [5], but due to the even more numerous applications, requirements, and constraints none of the existing graph layout algorithms are applicable to all kinds of diagrams. Therefore it is desirable to be able to select from multiple algorithms and to customize them according to the requirements of particular applications. Since the implementation of graph layout algorithms requires a lot of effort, it is advisable to reuse existing libraries such as OGDF[2] or Graphviz[3]. However, each of these libraries has its own API, and integrating such C or C++ libraries in Java applications is an intricate task. We solve these problems in the KIELER Infrastructure for Meta Layout (KIML) [2] by offering a common API for multiple frontends, i. e. diagram editors and viewers, and multiple backends, i. e. layout algorithms from various libraries. With this infrastructure the effort for integrating automatic layout in Eclipse-based applications is greatly reduced.

An important question is how to design a common interface for the selection and configuration of layout algorithms. We propose a design where each configuration option (denoted as *layout option* in the following) is represented by a key-value pair that is attached to a graph or graph element. We call the set of key-value pairs attached as layout options to a graph G the *abstract layout* of G. In contrast, *concrete layout* refers to the actual positions of the elements of G. When a layout algorithm is executed on a graph, it transforms the abstract layout into a concrete layout. By *meta layout* we denote the process of generating an abstract layout. The choice of layout algorithm is also encoded as a layout option named "*Algorithm*", hence it can be processed in the same way as other options. Other examples for commonly used options are the minimal spacing between nodes, the overall direction of edges, or the edge routing style. In KIML, graphs are represented with a data structure that includes abstract as well as concrete layout data [3]. When a layout for a graph is requested, our infrastructure generates an abstract layout and then applies the selected algorithms to the graph to compute a concrete layout.

III. Layout Configuration Interfaces

We present four interfaces for the meta layout process, i. e. the generation of layout configuration data, of which some address the user and others address the application developer.

a) Defaults and user preferences: For many layout options it is reasonable to set fixed default values in a particular context. Tool developers may want to predefine the layout for their diagram editors, and users may want to modify

[1] http://www.informatik.uni-kiel.de/rtsys/kieler/

[2] http://www.ogdf.net/

[3] http://www.graphviz.org/

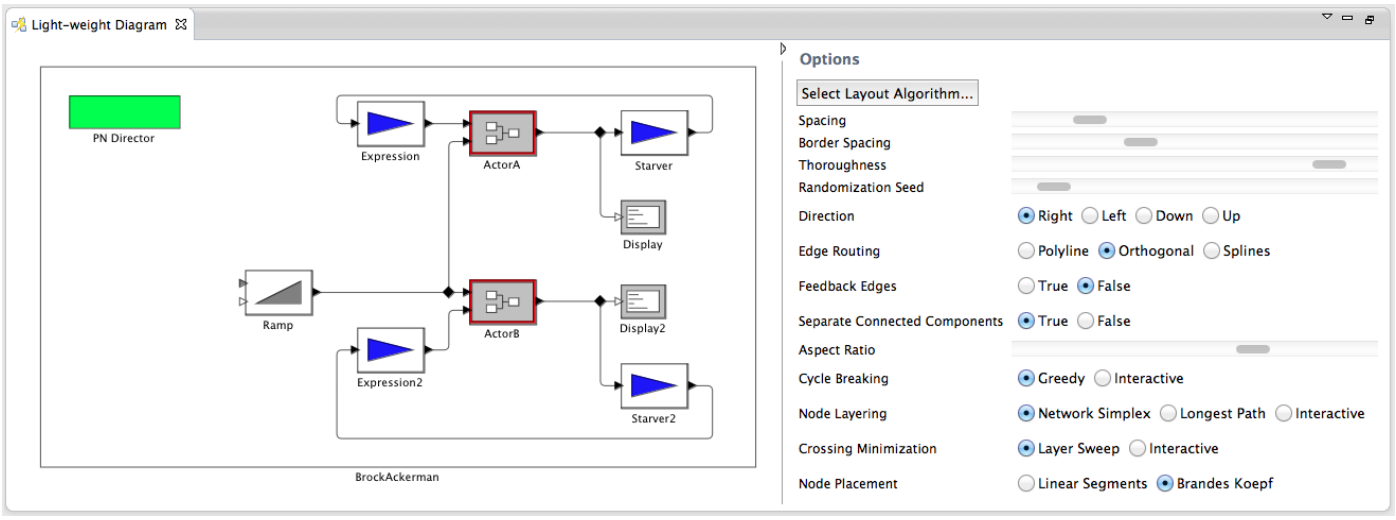

Figure 1. A process network diagram imported from the Ptolemy project [6] with sliders and buttons for direct manipulation of layout options.

some parameters for their own environment. For each affected layout option o, a specific diagram element class C is stored together with the respective value v. The class C can either represent a component of the concrete syntax, e. g. a diagram editor canvas, or a component of the abstract syntax, e. g. a meta model class of a domain-specific language. The option mapping $o \mapsto v$ is applied to the layout of all instances of C. The tool developer's settings are shipped together with the diagram editor, while the user's settings are kept in the local preference storage.

b) Diagram-bound settings: In order to allow a customized configuration for each diagram, layout option mappings must be linked directly with diagram elements. This can be done with annotations of either the concrete syntax model or the abstract syntax model. In the former case the options are applied only to the respective diagram, while in the latter case they are applied to all diagrams derived from the model. This distinction only makes sense if the model and view are separated, like in most UML editors. The diagram-bound configuration is displayed in the user interface in form of a table similar to the "*Properties*" view of Eclipse.

c) View management: The handling of graphical views can be improved with the concept of *view management* [2], where a simple interface for the combination of triggers and effects of the modeling environment is provided. In this context it is often necessary to perform automatic layout as an effect on a graphical view, and to apply different configurations to the layout depending on the state of the overall system. This is done with an interface for setting layout option mappings in a single layout execution; these mappings are held in a hash map, which is discarded after the layout is applied.

d) Transient views: The light-weight approach of creating transient graphical views allows a much faster application of automatic layout compared to editor-based solutions [3]. As a consequence, it is possible to manipulate layout options in a more interactive manner: Figure 1 shows a process network diagram together with a section of controls for directly changing a number of layout options. Moving a slider or clicking a button immediately triggers a re-layout of the view with the updated option value. For diagrams that are not too large (up to several hundreds of nodes on normal computers) the computation of a new layout and its application to the view is fluid enough to respond seamlessly to the user's actions.

IV. Conclusion

We presented concepts for the design of layout configuration interfaces. These interfaces allow a highly flexible integration of graph layout algorithms into modeling applications, both from the application programmer's perspective and from the user's perspective. By employing light-weight transient views it is possible to directly interact with the automatically computed layout through sliders and buttons. This is a confirmation of the usefulness of the transient views approach, which aims at supporting pragmatics-aware modeling with graph-based views that are optimized for the integration of automatic layout. Since the speed and fluidity of this approach is difficult to convey in a paper, we demonstrate it in a video that is available on the KIELER web page (see Section I) and distributed at the VL/HCC conference.

References

[1] L. K. Klauske, "Effizientes Bearbeiten von Simulink Modellen mit Hilfe eines spezifisch angepassten Layoutalgorithmus," Ph.D. dissertation, Technische Universität Berlin, 2012.

[2] H. Fuhrmann and R. von Hanxleden, "Taming graphical modeling," in *Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, ser. LNCS, vol. 6394. Springer, Oct. 2010, pp. 196–210.

[3] C. Schneider, M. Spönemann, and R. von Hanxleden, "Just model! – Putting automatic synthesis of node-link-diagrams into practice," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, San Jose, CA, USA, Sep.15–19 2013.

[4] S. Maier and M. Minas, "Combination of different layout approaches," in *Proceedings of the Second International Workshop on Visual Formalisms for Patterns (VFfP'10)*, ser. Electronic Communications of the EASST, vol. 31, Berlin, Germany, 2010.

[5] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.

[6] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity—the Ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.