# Automatic Layout of State Diagrams

Maxim Korotkov

eVelopers corp.
mkorotkov@evelopers.com

**Abstract.** Consider the problem of automatically generating layouts for state diagrams (statecharts). Such diagrams appear in automation engeneering and CASE tools. Automatic layout of these diagrams leads to better understanding and saves time spend on their development.
State diagram layout problem seems to have some differences from general graph layout problem. In this article adaptation of two graph layout algorithms to state diagrams layout problem is described. Results of this work are used in opensource UniMod project for diagram layout.
Force-directed method is adapted to work with state diagrams and extended with orthogonalization step. Orthogonal layout algorithm for drawing nonplanar graphs is also adapted to work with state diagrams.

## 1 Introduction

Opensource project UniMod [1] is focused on designing and implementing applications behavior using state diagrams. UniMod plug-in for Eclipse lets user create and edit state diagrams and run them as programs. Such development approach is called automata-based programming [2].

In any contemporary Integrated Development Environment (IDE) user has an ability to format code automatically. In case you have no code, but a set of diagrams, automatic layout stands for code formatting. There are several aesthetics which value quality of diagram: minimization of total number of crossings, minimization of the area of diagram drawing, etc. Let's take into consideration two aproaches to drawing a readable diagram: force-directed approach and topology-shape-metrics approach [3]. We need to remember that state diagrams unlike general graphs contain superstates (see Fig. 1).

So our aim is to adapt graph layout algorithms to state diagram layout problem. Some ideas of such adaptation are given in [4].

## 2 Force-directed Method

The first algorithm uses force-directed approach [5]. Layout process can be divided into two stages:

- Minimization of penalty function.
- Orthogonalization of the layout.

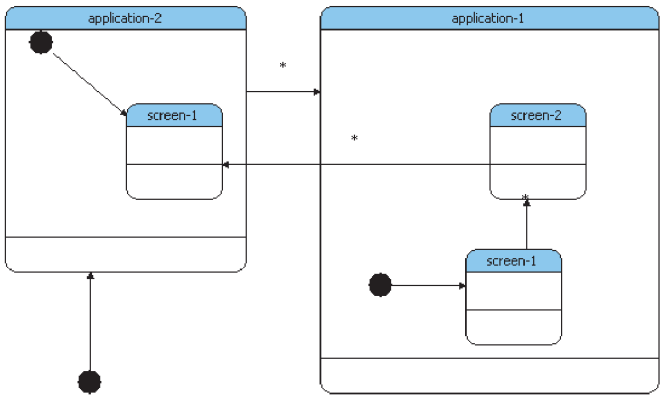


**Fig. 1.** Superstates

Let's construct a penalty function to be minimized. It will contain components responsible for different "problems" in the drawing.

- Edge intersection.
- Deviation of edge length from optimal length.
- Vertices overlapping (in case when one of corresponding states is not nested into another).
- Deviation of distance between states from the optimal distance.
- Too much space occupied by drawing.
- Too large superstates.

We are neither going to give a complete explanation for each component of penalty function nor to list constants used to calculate these components, because there are enough implementations of force-directed method and materials about it available. Minimization of penalty function gives us a simple layout (with vertices coordinates calculated and edges represented as line segments). Then we'd like to orthogonalize drawing of this diagram. The original method described below is used for this purpose and it can be divided into three stages:

- Distribution of edges between vertex sides.
- Distribution of loops between vertex sides.
- Sorting of edges and loops on each side (distribution of ports on the side).

After we assign sizes and coordinates to vertices we need to route every edge, because proper edge-routing has a great influence on diagram readability. Let's set maximum number of bending for an edge to 1. In case two vertices lay on a vertical or horizontal line there is only one way to route an edge. Otherwise there are two and only two (general) ways to route an edge (see Fig. 2).

Let's perform following steps to distribute of edges and loops between vertex sides.

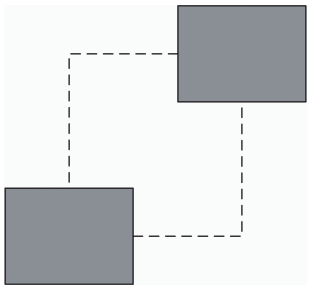

**Fig. 2.** Two routes for an edge

– Route edges with no bendings.
– Route other edges to distribute them uniformly between vertex sides.
– Place loops (one by one) at sides with minimal number of edges.

Then we need to sort edges on each side (so to determine the precise position of edge ports on a vertex side) trying to minimize the number of edge intersection. For this purpose we will assign ports to edges as it is shown at Fig. 3.

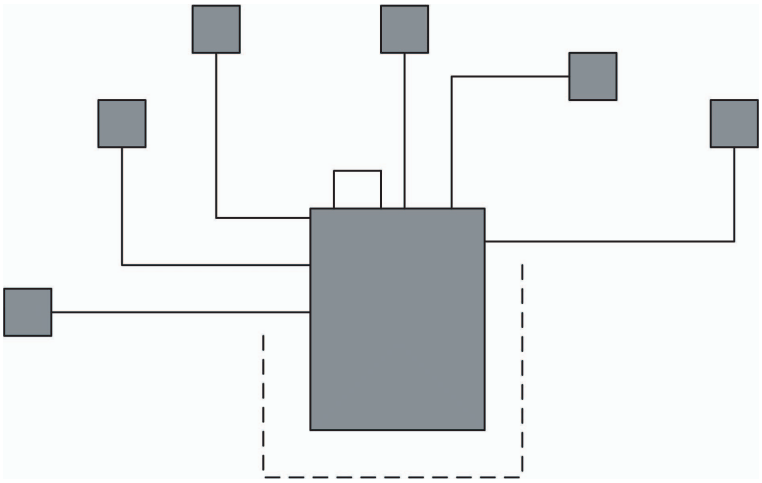

**Fig. 3.** Edge-routing

This algorithm layouts small diagrams good enough (see Fig. 4). It was used in UniMod plugin for Eclipse version 1.1.7, but it appeared to produce not very good drawings and to work extremely slow (for diagrams containing 100 and more states). So we came to other layout techniques.

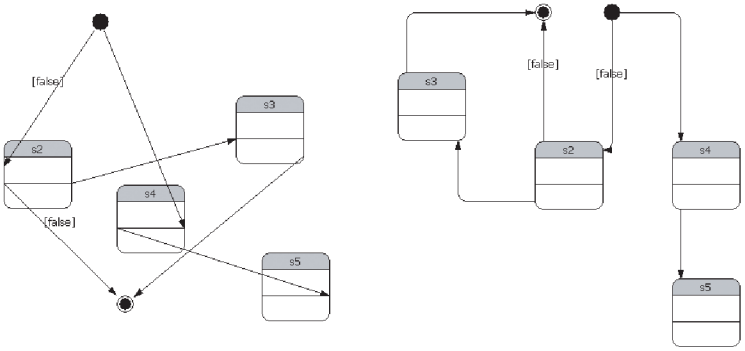


**Fig. 4.** Force-directed method. Results

## 3 Nonplanar Orthogonal Orientations

The following part of this article is about adaptation of techniques introduced in [6] for drawing nonplanar graphs vith vertices of degree higher then four without a planarization step. The adapted version of this algorithm will be used in next version of UniMod plugin for Eclipse (1.1.13) and contains following steps:

- Splitting graph into layers.
- Layouting graphs of each layer.
- Joining layers into one graph.

Sets of inner states of each superstate are treated as separate graphs and layout for each of them is constructed independently. So our graph with superstates is separated into a set of flat graphs (without superstates). Edges between vertices of different graphs are temporary hidden (in the state diagram from Fig. 1 edge between *screen-2* in *application-1* and *screen-1* in *application-2* are hidden). Space is reserved for ports of hidden edges.

Each flat graph is made biconnected [7] (by adding dummy edges) and st-graph is build from it [8]. Vertices are added to drawing of the graph in st-numbering order. Then we use algorithm based on one described in [6] to determine sizes and coordinates of any vertex. Let's consider that we are placing vertex $V$. All outgoing edges will have ports at the top side, and we place this vertex horizontally so that it will have equal (or nearly equal) number of incoming edges at each side. We place $V$ vertically in a new "row". Every edge will always have one and only one bending. The idea of algorithm is illustrated at Fig. 5.

At the last step we need to join drawings of flat graphs. For this we determine size of each drawing except the top-level one and draw it in the corresponding superstate. Finally we route hidden edges.

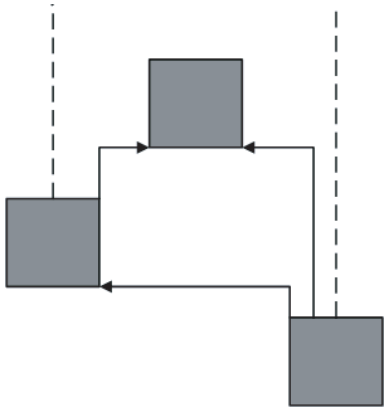

**Fig. 5.** Vertices placement

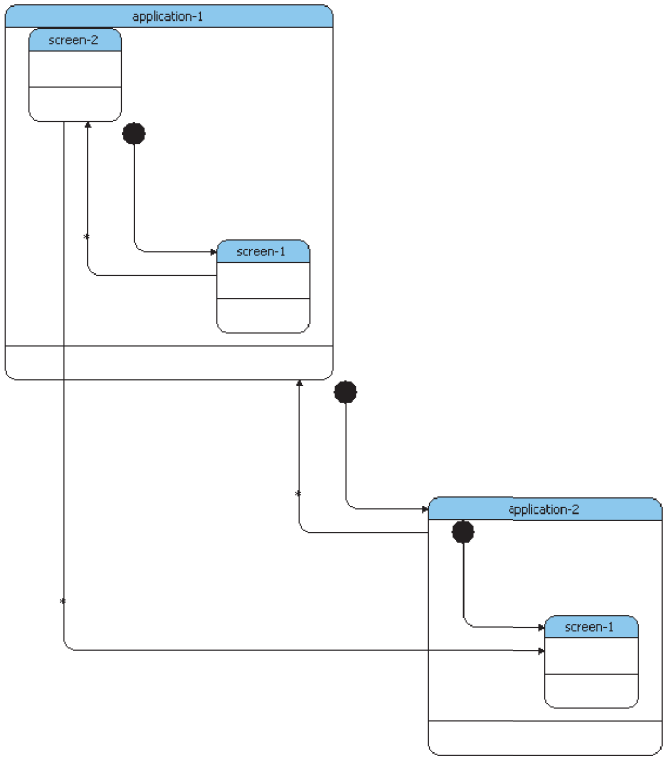


**Fig. 6.** Result

This algorithm provides suitable layout of a state diagram (see Fig. 6). Its main advantage is in high speed (even for large diagrams): the running time of this algorithm for graph with m edges is $O(m)$.

## 4 Conclusions

Research made leads to the conclusion that most popular tools working with UML diagrams have no good mechanism to layout them. At the other hand, contemporary graph drawing tools use some algorithms that produce good drawings and seem to be adaptable to the state diagram layout problem (for example GIOTTO algorithm). It opens possibility to adapt these algorithms and use them in various UML editors and UML-related tools.

## References

1. Gurov, V., Mazin, M.: UniMod project website. http://unimod.sourceforge.net
2. Shalyto, A.: Switch-technology. Algorithmization and programming of logical control problems. Saint-Petersburg, 1998. Nauka.
3. Sugiyama, K.: Graph Drawing and Applications for Software and Knowledge Engineers. Singapore, 2002. Mainland Press.
4. Klaw, G., Mutzel, P.: Automatic layout and labeling of state diagrams. Graph Drawing (proceedings GD'97).
5. Frutcherman, T., Reingold, E.: Graph Drawing by Force-directed Placement. Software – Practice And Experience 21 (1991) 1129-1164.
6. Battista, G., Eades, P., Tamassia R., Tollis, I.: Algorithms for the Visualization of Graphs. New Jersey, 1999. Prentice Hall.
7. Hsu, T., Ramachandran, V.: On finding a smallest augmentation to biconnect a graph. SIAM Journal on Computing 22 (1993) 889-912.
8. Maon, Y., Schieber, B., Vishkin, U.: Parallel ear decomposition search and st-numbering in graphs. Courant Institute of Mathematical Sciences. Computer Science Department Technical Report 222 (1986).