Drawing Activity Diagrams

Martin Siebenhaller, Michael Kaufmann

WSI-2006-02

ISSN 0946-3852

Arbeitsbereich Paralleles Rechnen - Prof. Dr. Michael Kaufmann
Wilhelm-Schickard-Institut für Informatik,
Fakultät für Informations- und Kognitionswissenschaften,
Eberhard-Karls-Universität,
Sand 13, 72076 Tübingen, Germany

Email: {siebenha,mk}@informatik.uni-tuebingen.de

Drawing Activity Diagrams

Martin Siebenhaller*
University of Tübingen

Michael Kaufmann†
University of Tübingen

Abstract

Activity diagrams experience an increasing importance in the design and description of software systems. Unfortunately, previous approaches for automatic layout support fail or are just insufficient to capture the complexity of the related requirements. We propose a new approach tailored to the needs of activity diagrams which combines the advantages of two fundamental layout concepts called "Sugiyama's approach" [Sugiyama et al. 1981] and "topology-shape-metrics approach" [Tamassia et al. 1988], originally developed for layered layouts of directed graphs and for orthogonal layout of undirected graphs respectively.

Keywords: Activity Diagrams, Software Visualization, Graph Drawing

1 Introduction and Motivation

Activities and the corresponding activity diagrams belong to the basic concepts of the Unified Modeling Language (UML) which has become the standard modeling language for specifying, visualizing and documenting software systems. Activity diagrams are used for modeling behavioral logic like business processes or workflow. In the modern UML 2, they experienced an increasing importance. While they were considered as a special case of state diagrams in UML 1.x, they are now enriched with additional constructs that widens the range of their applicability. This make them more suitable for areas like economics or bio-informatics [Shegogue and Zheng 2005].

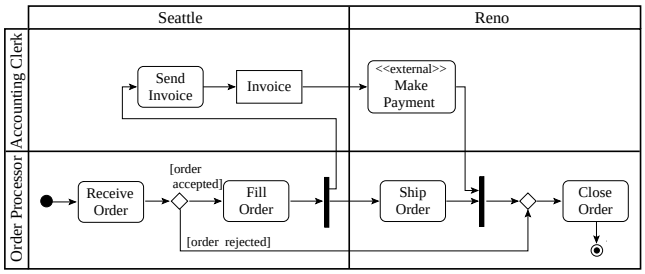


Figure 1: An UML activity diagram taken from the UML 2.0 Superstructure specification (http://www.uml.org).

While automatic layout for class diagrams received considerable attention by developers of corresponding software tools and designers of fundamental concepts of diagramming, activity diagrams were considered either to be just a special case for tools supporting state diagrams or to be too complex to handle them by a direct application of basic layout algorithms.

Activity diagrams require properties different from those for class diagrams such that the concepts developed there cannot be transferred, although the experiences gained help with the design. Here the emphasis lies on partitions and flow, some properties that are influenced by the semantics of activity diagrams. Those properties have to be supported by an automatic layout algorithm because it is hard, or sometimes even impossible to place all elements appropriately by hand. Furthermore, an automatic layout algorithm with customizable layout options allows to include different user preferences.

The contribution of this paper is a summary of the needed requirements and aesthetics for drawing activity diagrams, and the elaboration of sophisticated graph drawing algorithms for a new layout approach which satisfies them. We do not introduce a completely new concept, but we show for the first time how to combine the two layout concepts "Sugiyama approach" and "topological-shape-metrics approach", which were originally developed for layered layouts of directed graphs and for orthogonal layouts of undirected graphs respectively. We demonstrate how to extract the advantages from both methods and apply them successfully to the challenging problem of layout UML activity diagrams.

The rest of our paper is organized as follows: In Section 2 we discuss the requirements and aesthetic criteria of activity diagrams. Section 3 reviews the state-of-the-art of drawing activity diagrams and Section 4 describes the basic definitions and algorithms which provide the basis for our new visualization approach. In Section 5 we introduce our new concepts followed by a short discussion.

2 Requirements for the Layout

In this section we examine requirements for the automatic layout of activity diagrams. We therefore shortly introduce the notation elements. This is necessary to derive a graph theoretic concept to layout those diagrams. We also analyze aesthetics and standards for creating activity diagrams. The identified aesthetics and requirements are the basis for the design of our new layout algorithm.

2.1 Notation of UML 2 Activity Diagrams

In the following we give an overview of the visual notation of activity diagrams and identify the resulting requirements for the layout algorithm. We do not focus on semantics if not necessary. For further details have a look at the UML superstructure specification [OMG 2004].

An *activity* specifies the coordination of executions of actions using a control and data flow model. Each *activity diagram* shows exactly one activity. An activity is modeled as a labeled graph of *activity nodes* which are connected by edges denoting data or control flow. It can optionally be represented by a border rectangle containing all its elements and a name shown in the upper left corner. There are three different node types appearing in an activity - action, object and control nodes.

Action nodes (see Fig. 2(a)) are the basic elements of an activity. An action node may have outgoing and incoming edges denoting control or data flow to or from other nodes. There are two special kind of action nodes to handle signal events, the *accept event* and

*e-mail: siebenha@informatik.uni-tuebingen.de
†e-mail:mk@informatik.uni-tuebingen.de

send signal action. *Object nodes* indicate an instance of a particular classifier (see Fig. 2(c)).

A *control node* (see Fig. 2(b)) coordinates the flow between other nodes. If an activity is invoked, a flow starts at each *initial node*. If a flow reaches an *activity final node* the activity terminates. In contrast, a *flow final node* terminates only its incoming flow. A *decision node* has one incoming flow and multiple outgoing edges. The flow is forwarded to only one outgoing edge. This edge is commonly determined by *guards* which are edge labels representing conditions. A *merge node* has one outgoing edge and multiple incoming edges. Each flow arriving at an incoming edge is forwarded to the outgoing edge. A *fork node* is similar to a decision node, but it splits the incoming flow into multiple concurrent flows. A *join node* is similar to a merge node, but synchronizes multiple flows. Edges are always connected to the long side of the fork/join node. The nodes can be placed vertically or horizontally.

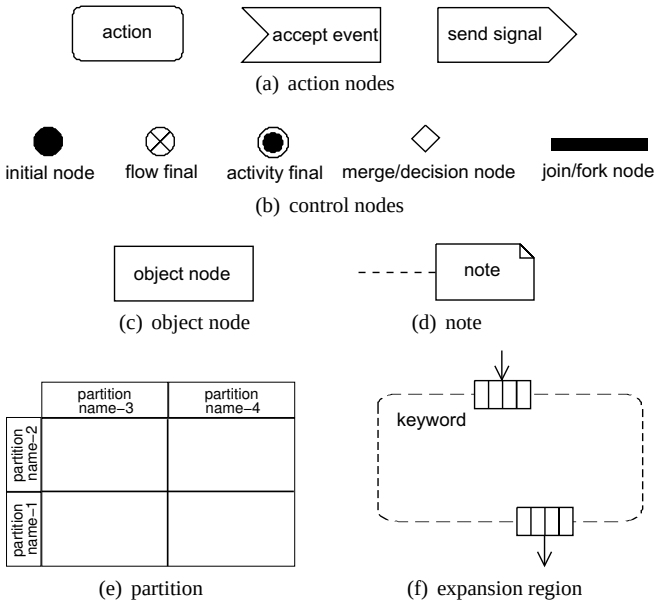


Figure 2: Activity diagram notation elements.

Node elements can be connected by two different edge types, one denoting control flow and one denoting data flow. Both edge types are represented by an arrowed line.

There are also notation elements for defining sets of nodes and edges in an activity. *Partitions (swim-lanes)* use vertical or horizontal boundaries to partition a diagram into logical areas, e.g. organizational units in a business model. Nodes and edges are placed inside the related partition. UML 2 diagrams can also be partitioned as depicted in Fig. 2(e). Such a partition is a combination of horizontal and vertical swim-lanes. Hence, the partition structure corresponds to a grid. Another grouping element is an *expansion region*. It is a strictly nested region of an activity with explicit input and output nodes called *expansion nodes* (see Fig. 2(f)). Each input is a collection of values. Expansion regions can have keywords in the upper left corner. An *interruptible activity region* has the same notation as an expansion region but without expansion nodes.

A notation element available to all UML diagrams is a *note*. Notes comment on some diagram elements. They are attached to the corresponding elements by a dashed line (see Fig. 2(d)). In activity diagrams notes are often used to show post- and preconditions of actions.

The visual notation leads to the following requirements:

- Nodes have different sizes (NODE_SIZE).
- We have to consider partitions (PARTITION).
- Regions could be nested and edges can start/end at a region (expansion input/output nodes). This corresponds to the concept of compound graphs (see Section 4.1) (CLUSTER).
- Handle notes that are attached to edges (NOTES).
- Join/fork nodes are two sided nodes (TWO_SIDED_NODES).
- Handle labels of activities, regions, nodes, edges (LABEL).

Since activities are based on a graph theoretic concept we are able to employ graph drawing techniques to draw activity diagrams. There is a wide variety of drawing algorithms taking different aesthetics and requirements into account [Di Battista et al. 1999b; Kaufmann and Wagner 2001]. There are also approaches for handling labels and clusters. Note, that in graph theory the nodes are usually called vertices. However, in this work we use the term nodes to be consistent with the UML specification.

2.2 Aesthetics

An aesthetics measures a graphical property of a drawing that should be optimized to increase readability. Unfortunately, up to now there are neither work nor empirical studies for aesthetics of activity diagrams. However, since the underlying structure of activity diagrams are graphs, we will discuss aesthetics that apply to abstract graphs (graphs without special semantics) and thus to activity diagrams, too.

An overview of aesthetic criteria applied to drawings of abstract graphs is given in [Di Battista et al. 1999b; Coleman and Parker 1996]. The most common are:

- Minimize the number of edge crossings (CROSSING).
- Minimize the area of the drawing (AREA).
- Minimize the maximum length of an edge (EDGE_LENGTH).
- Minimize the number of bends (BEND).
- Maximize the smallest angle between two edges incident on the same node (ANGLE).
- Minimize the deviation of an $1:1$ aspect-ratio (ASPECT_RATIO).
- Minimize the number of overlapping nodes and edges (OVERLAP).
- Maximize the number of orthogonal edges (edges that are drawn as a sequence of horizontal and vertical line segments) (ORTHOGONAL).
- Maximize the number of edges respecting flow (edges that are drawn monotonically in a prescribed direction) (FLOW).
- Maximize symmetry (SYMMETRY).

Note, that there are some contradicting aesthetics, e.g. FLOW and ASPECT_RATIO. The drawing of a graph with a given set of aesthetic criteria can be seen as a multi-objective optimization problem. The set of applied criteria depends on the semantics of the graph as well as on individual user preferences.

The affect of aesthetics BEND, CROSSING, ANGLE, ORTHOGONAL and SYMMETRY on the readability of drawings were analyzed empirically in [Purchase et al. 1997; Purchase 1997]. There, CROSSING was found out to be the most important, followed by the less important aesthetics BEND and SYMMETRY. ORTHOGONAL and ANGLE had no significant effects.

Since activity diagrams show control and data flows it seems to be natural to include aesthetics FLOW. In activity diagrams the edge

flow is usually from top to down or from left to right. OVERLAP is also important; especially overlaps between nodes should be avoided. SYMMETRY seems to be useless to activity diagrams because they do not have a symmetric structure.

2.3 Standards for Creating Activity Diagrams

Although the UML specification does not explicitly prescribe how to layout diagrams, there are a lot of standards, conventions and guidelines how to do this. A comprehensive collection of those principles that have been proven in practice is given in [Ambler 2005]. Their usage increases the usability and clarity of diagrams. Here we give an overview.

The principles for general UML diagrams are:

- Minimize the number of crossings (CROSSING).
- Draw edges orthogonal (ORTHOGONAL).
- Use only horizontal labels (HORIZONTAL_LABELS).
- Reorganize larger diagrams into several smaller ones. This principle is based on the fact that it is often easier to have several diagrams on various levels of detail than a single complex one. Hence, complex actions are often decomposed into subactivities. In practice there are almost no activity diagrams containing more than 50 nodes and 80 edges, which allows us to use more complex algorithms than for larger graphs.

The following principles were specific to activity diagrams:

- Incoming and outgoing edges enter a join (fork) node on different sides (BIMODALITY).
- Swim-lanes (partitions) should be ordered in a logical manner, the primary swim-lane is placed leftmost. Thus, we assume that the ordering of the partitions is given as input.

In this section we have identified requirements and aesthetics for drawing activity diagrams. Notation dependent requirements are CLUSTER, PARTITION, NODE_SIZE, NOTES, TWO_SIDED_NODES and LABEL. Furthermore, we have structure-dependent aesthetics like CROSSING, BEND, ORTHOGONAL, FLOW, AREA, EDGE_LENGTH and OVERLAP. BIMODALITY and HORIZONTAL_LABELS are additional requirements based on standards and conventions for creating activity diagrams. It is important to have a flexible layout approach that allows the user to customize requirements.

3 State-of-the-Art

In this section, we review the state-of-the-art of drawing activity diagrams. First we give an overview on related work in research followed by a brief evaluation of some popular commercial software products.

3.1 Related Conceptual Work

Most of the recent work on layout approaches for UML diagrams was dedicated to class diagrams. Representatives of two different approaches are UMLKandinsky/GoVisual [Eiglsperger et al. 2004] and SugiBib [Eichelberger 2002b]. The first one is based on refinements of the fundamental topology-shape-metrics (TSM-) approach which has been developed for orthogonal layouts. The second one uses the well known concept of Sugiyama for layered layouts. Both approaches will be described in Section 4. They support aesthetic criteria like FLOW, ORTHOGONAL, OVERLAP, CROSSING in various ways. SugiBib is also able to consider advanced properties like CLUSTER but on the other hand it is rather weak for basic properties like NODE_SIZE or ORTHOGONAL. None of the above approaches is able to handle PARTITION.

A layout approach for statecharts is given in [Castello et al. 2002a; Castello et al. 2002b]. Statecharts are extended finite state machines and support the repeated decomposition of states into substates. The approach is based on a combination of Sugiyama's approach with a recursive floorplanning algorithm and an integrated labeling method. It supports aesthetics AREA, CROSSING, OVERLAP, BEND, ASPECT_RATIO and CLUSTER. However, the clustering is specific to state decompositions and thus not applicable to activity diagrams.

There is also some work about the visualization of process diagrams. Process Diagrams are related to flowcharts and visualize the flow through a process or system. The layout approaches described in [Wittenburg and Weitzman 1997] and [Six and Tollis 2002] support FLOW and PARTITION. The second one is also able to support ORTHOGONAL, CROSSING, BEND and OVERLAP. However, both approaches are restricted to one-dimensional partitions (horizontal swim-lanes) and do not include CLUSTER.

As we have seen there is no conceptual approach covering all requirements for activity diagrams. On the other side there exist quite a few commercial products that claim to support activity diagrams. In the following subsection we review some of them.

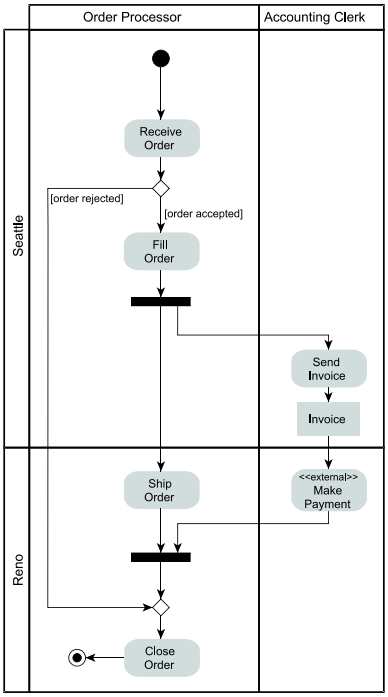
3.2 UML Tools

An evaluation of layout capabilities for different UML tools regarding class diagrams was given in [Eichelberger 2002a]. For most tools the contained layout algorithms did not produce satisfying results. Meanwhile some of the tools provide improved layout capabilities. In the following we review some popular UML tools with respect to their auto layout capabilities for activity diagrams. We are particularly interested in their ability to handle requirements FLOW, CLUSTER and PARTITION.

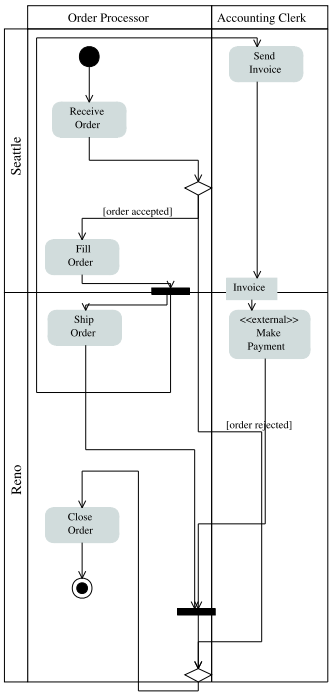
- EclipseUML Studio for Java 2.1.0 beta (Omondo)
  (`http://www.omondo.com`)
  Up to know EclipseUML does only support UML 1.x notation for activity diagrams without partitions. It does not provide an auto layouter for those diagrams.

- MagicDraw UML 11 Enterprise Edition (NoMagic)
  (`http://www.magicdraw.com`)
  MagicDraw fully supports UML 2 notation. It provides several standard auto layout approaches and edge routers (an edge router does not change node positions). The hierarchic layouter is able to consider FLOW, PARTITION and CLUSTER. However, when using partitions aesthetics FLOW is not always fulfilled satisfactorily (see Fig. 3(b)) and the resulting drawings are sometimes defective. Notes are always placed outside the corresponding partition cell/region, far away from the related element.

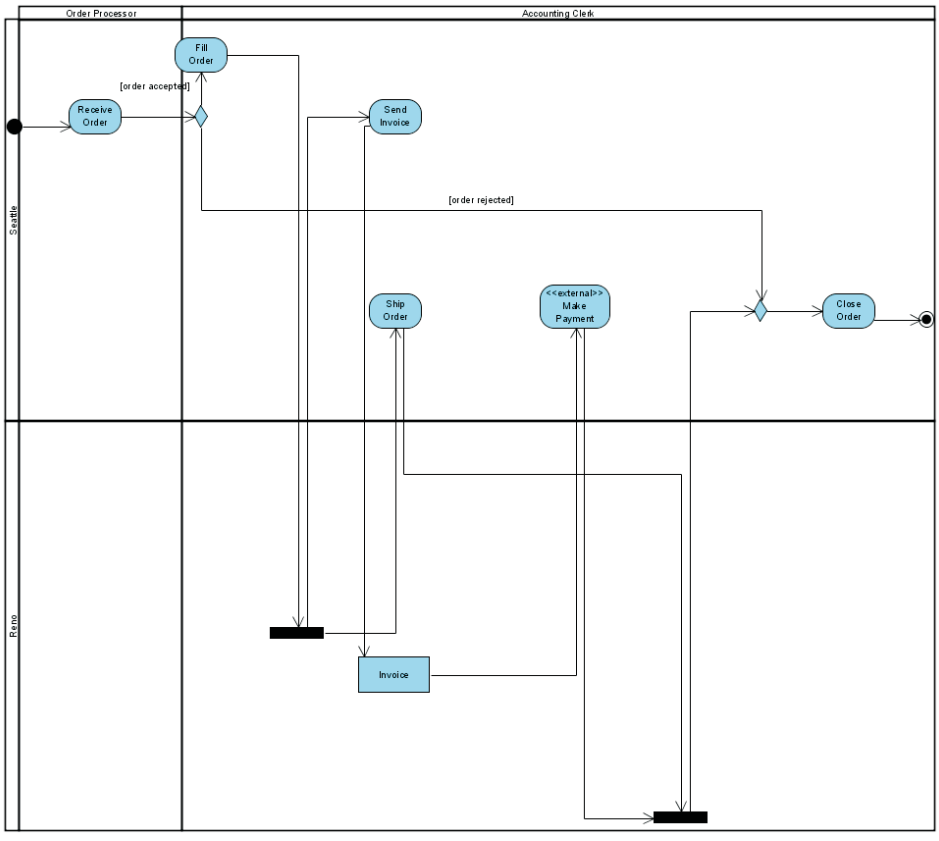
- Poseidon for UML 4.1 Professional (Gentleware)
  (`http://gentleware.com`)
  Poseidon fully supports UML 2 notation for activity diagrams. However, it does not provide an auto layouter for them.



(a) New Approach



(b) MagicDraw UML 11



(c) Visual Paradigm for UML 5.3

Figure 3: Comparison of different layout tools (using the example of Figure 1). Only our new approach produces an adequate drawing.

- Rational Software Architect V 6.0 (IBM)
  (`http://www.ibm.com/software/rational`)
  Rational supports UML 2 notation for activity diagrams but it only allows one-dimensional partitions (vertical swim-lanes). The provided auto layouter does not consider FLOW but it is able to consider CLUSTER. However, it does not layout elements inside a region. When using swim-lanes the layouter only includes elements of the leftmost swim-lane. The layout quality is very poor.

- Together Architect 2006 for Eclipse (Borland)
  (`http://www.borland.com/de/products/together`)
  Together does only support a subset of UML 2 notation. Partitions and regions are not available. Together provides several standard layout approaches. However, the layout results are poor. The layouter does not support aesthetics FLOW and often violates aesthetics OVERLAP and CROSSING.

- Visual Paradigm for UML 5.3 Enterprise Edition (Visual Paradigm)
  (`http://www.visual-paradigm.com`)
  Visual Paradigm fully supports UML 2 notation for activity diagrams. It provides a hierarchic layout approach as well as an orthogonal edge router. The hierarchic layouter is able to consider FLOW, CLUSTER and PARTITION but when using partitions with horizontal swim-lanes the elements are not placed correctly and the layout results are poor (see Fig. 3(c)). Furthermore, it sometimes produces unnecessary overlaps.

The layout capability and quality of the above UML tools heavily differs. None of them was able to produce satisfying results. Only MagicDraw and Visual Paradigm incorporate partitions. However, as Figure 3 shows, both produce poor results when using partitions even for small diagrams.

4 Basic Definitions and Algorithms

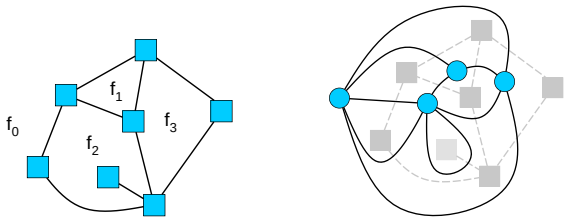
In this section we describe the basic definitions as well as the basic algorithms which provide the basis for our visualization. Our approach combines the layered approach of Sugiyama with a TSM-approach for orthogonal layouts.

4.1 Definitions

A *graph* $G = (V,E)$ consists of a node set V and an edge set $E \subseteq V \times V$. If all pairs in E are ordered, we call G *directed graph*, if all pairs are unordered we call it *undirected graph*. A *self-loop* is an edge that starts and ends at the same node. If there are multiple edges between a pair of vertices, those edges are called *multiedges*. A (simple) *path* between node v and w is a node sequence $(v = u_1, u_2, \ldots, u_k = w)$, such that $u_1, .., u_k \in V$, $(u_i, u_{i+1}) \in E$, $1 \leq i \leq k-1$ and $u_i \neq u_j$, $1 \leq i \neq j \leq k$. It is denoted by $v \rightarrow_G^* w$. A *cycle* is a non-empty path with $u_1 = u_k$. If a graph contains no cycles it is called *acyclic*. An undirected graph is called *connected* if there is a path between every pair of nodes. A directed graph is connected if its undirected version is connected. A *tree* is an undirected, acyclic and connected graph ($\Rightarrow |E| = |V| - 1$). The *degree* $\delta_G(v)$ of a node $v \in V$ is the number of edges incident to v. If G is directed, $\delta_G(v)$ can be divided into the *out-degree* $\delta_G^+(v) = |\{w|(v,w) \in E\}|$ and the *in-degree* $\delta_G^-(v) = |\{w|(w,v) \in E\}|$.

A *drawing* of a graph $G = (V,E)$ is a mapping of the node set V to distinct points in the plane and the edge set E to open Jordan curves. G is called *planar*, if it has a drawing in the plane without edge crossings, that is no two Jordan curves have a common point. Such a drawing divides the plane into regions, called *faces* (see Fig. 4(a)). There is exactly one unbounded region which is called the *outer face*. An *embedding* of a graph is given by a clockwise cyclic ordering of the adjacent edges around each node. An embedding is called planar if there is a planar drawing of the graph which preserves this ordering.

The *dual graph* D_G of a planar embedding of G has a node v_f for each face f of G and an edge (v_f, v_g) for each edge of G separating two (not necessarily distinct) faces f and g (see Fig. 4(b)). The dual graph has always linear size and is planar, too.



(a) planar drawing (b) dual graph

Figure 4: Figure (a) shows a planar drawing of a graph and the corresponding faces (f_0 denotes the outer face). Figure (b) shows the related dual graph. Its nodes are denoted by circles and the edges by solid curves.

An *upward drawing* of a directed graph $G = (V,E_D)$ is a drawing of G such that all edges are represented by monotonically increasing curves in the vertical direction. Note, that such a drawing exists if and only if G is acyclic. G is called *upward planar* if it has an upward and a planar drawing at the same time. Note, that there are graphs which have an upward and a planar drawing but are not upward planar (see Fig. 5(a)). An *upward embedding* of a graph is given by a clockwise cyclic ordering of the adjacent edges around each node in which the incoming and outgoing edges form an interval. Such an ordering is called *bimodal*. An upward embedding is planar if there is an upward planar drawing of the graph which preserves the corresponding ordering.

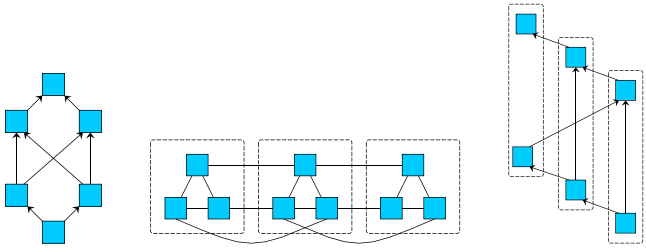

(a) (b) (c)

Figure 5: Example graphs. Dashed rectangles represent clusters.

A *mixed graph* is a tuple $G = (V,E_D,E_U) \subseteq (V,V \times V,V \times V)$, where V is the set of nodes, E_D the set of directed edges and E_U the set of undirected edges. A *mixed upward drawing* of G is a drawing such that the edges of E_D are represented by monotonically increasing curves in the vertical direction. G is called *mixed upward planar* if it has a mixed upward and a planar drawing at the same time. A *mixed upward embedding* is planar if there is a mixed upward planar drawing of the graph which preserves the corresponding ordering.

A *compound graph* $G_C = (G, T)$ consists of a graph $G = (B \cup C, E_G)$ (called *underlying graph*) and a directed rooted tree $T = (B \cup C, E_T)$ (called *inclusion tree*). The node set B contains the *base nodes* and the set C the *compound nodes*. Compound nodes (=clusters) contain base nodes or other compound nodes. T defines the containment relation. The tree edges $e \in E_T$ are directed from the root to the leafs. A directed path $v \rightarrow_T^* w$ indicates that node w belongs to node v. Thus, each base node is a leaf of T and the compound nodes are the inner nodes. The root of T is a special compound node which represents the whole drawing area and thus includes each element. An example is given in Figure 6.

In a *cluster drawing* of a compound graph G_C, each compound node $c \in C$ is drawn as a simple closed region (usually as rectangle). The region of c contains a base node $v \in B$ if and only if there is a path $c \rightarrow_T^* v$. Analogously, it contains the region of a compound node c' if and only if there is a path $c \rightarrow_T^* c'$. Furthermore, each edge crosses the boundary of a region at most once. A compound graph is called *c-planar* if it has a planar and a cluster drawing at the same time. There are graphs which have a cluster drawing as well as a planar drawing but are not c-planar (see Fig. 5(b)).

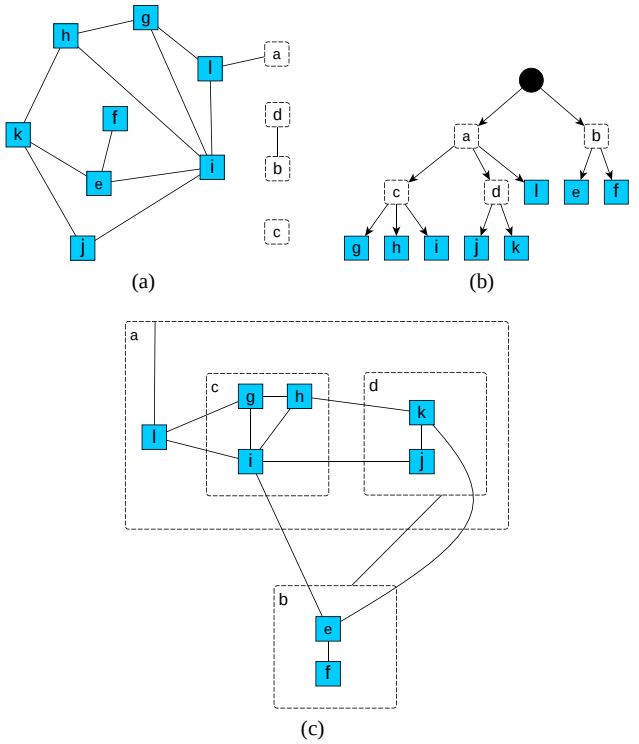


Figure 6: Example for a compound graph, where compound nodes are drawn as rounded rectangles with dashed border. Figure (a) shows the underlying graph, Figure (b) the corresponding inclusion tree and Figure (c) the resulting cluster drawing.

A *grid partition* $P_{m,n}$ subdivides the drawing area into m columns and n rows which are separated by $m+1$ vertical and $n+1$ horizonal lines, respectively (see Fig. 7(a)). Note, that in our approach the size of the rows (columns) is non-uniform. A cell of the grid is called *partition cell*. $p_{i,j}$ denotes the partition cell located in the i-th column and j-th row of the grid ($p_{1,1}$ is located in the upper left corner). The i-th grid column contains all partition cells $p_{i,j}$ with $1 \leq j \leq n$ and the j-th grid row contains all partition cells $p_{i,j}$ with $1 \leq i \leq m$. For a grid partition $P_{m,n}$ the corresponding *grid partition graph* $P_{m,n}^G$ is constructed by placing nodes on intersection points of horizontal and vertical grid lines (see Fig. 7(b)).

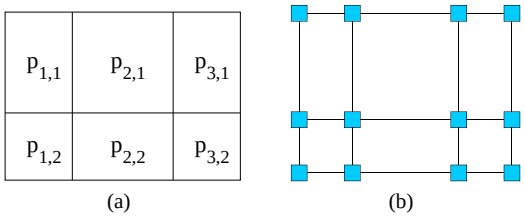


Figure 7: Example of a grid partition $P_{3,2}$ (a) and the corresponding grid partition graph $P_{3,2}^G$ (b).

Let $G = (V, E)$ denote a graph, $P_{m,n}$ a grid partition and $p\colon V \rightarrow \mathbb{N} \times \mathbb{N}$ a function assigning each node $v \in V$ to a partition cell.

Definition 1 *In a* partitioned drawing *of G each node $v \in V$ is drawn inside $p(v)$. G is called* p-planar *if it has a planar and a partitioned drawing at the same time.*

Theorem 1 *A graph $G = (V, E)$ is p-planar if and only if it is planar.*

Proof: A p-planar graph is planar by definition. Let us assume that each node $v \in V$ is assigned to an arbitrarily distinct location inside its partition cell $p(v)$. In [Pach and Wenger 1998] Pach and Wenger show that every planar graph admits a planar embedding in which each vertex is mapped to an arbitrarily prescribed distinct location. This implies that each planar graph is p-planar, too. □

Definition 2 *A mixed compound graph G_C has a* partitioned mixed upward cluster drawing *if it has a cluster, a mixed upward and a partitioned drawing at the same time. Furthermore, G_C is called* mixed upward pc-planar, *if it has a planar and a partitioned mixed upward cluster drawing at the same time.*

There are graphs which are upward planar and c-planar but not upward c-planar (see Fig. 5(c)). In our approach cluster regions are not allowed to cross partition cells. Thus each compound node can be assigned uniquely to a partition cell. Under this assumption each graph has a partitioned cluster drawing. A graph has a partitioned upward drawing if it is acyclic and for each $e = (v, w) \in E_D$ with $p(v) = p_{i,j}$ and $p(w) = p_{k,l}$ holds $j \geq l$.

4.2 Sugiyama's Approach

The most common approach for producing layered drawings for directed graphs is the abstract framework developed by *Sugiyama* et al [Sugiyama et al. 1981]. It indicates a common direction of the edges by producing layered layouts. Sugiyama's framework consists of four phases:

1. **Cycle Removal**: In the cycle removal phase, the directed input graph $G = (V, E)$ is made acyclic by reversing appropriate edges (see Fig. 8(b)).
2. **Layer Assignment**: During the layer assignment phase, the nodes are assigned to horizontal layers L_1,..,L_h such that for each edge $e = (v, w) \in E$ with $v \in L_i$ and $w \in L_j$ holds $i < j$. After this assignment, edges between nodes of non-adjacent layers ("long edges") are replaced by chains of dummy nodes and edges between the corresponding adjacent layers. This process is called *normalization* and the result is the normalized graph $N_G = (V_N, E_N)$ (see Fig. 8(c)).
3. **Crossing Reduction**: In the crossing reduction phase, an ordering of the nodes within a layer is computed such that the number of edge crossings is reduced (see Fig. 8(d)). The result is a directed acyclic compaction graph $C_G = (V_N, \{(a, b) :$

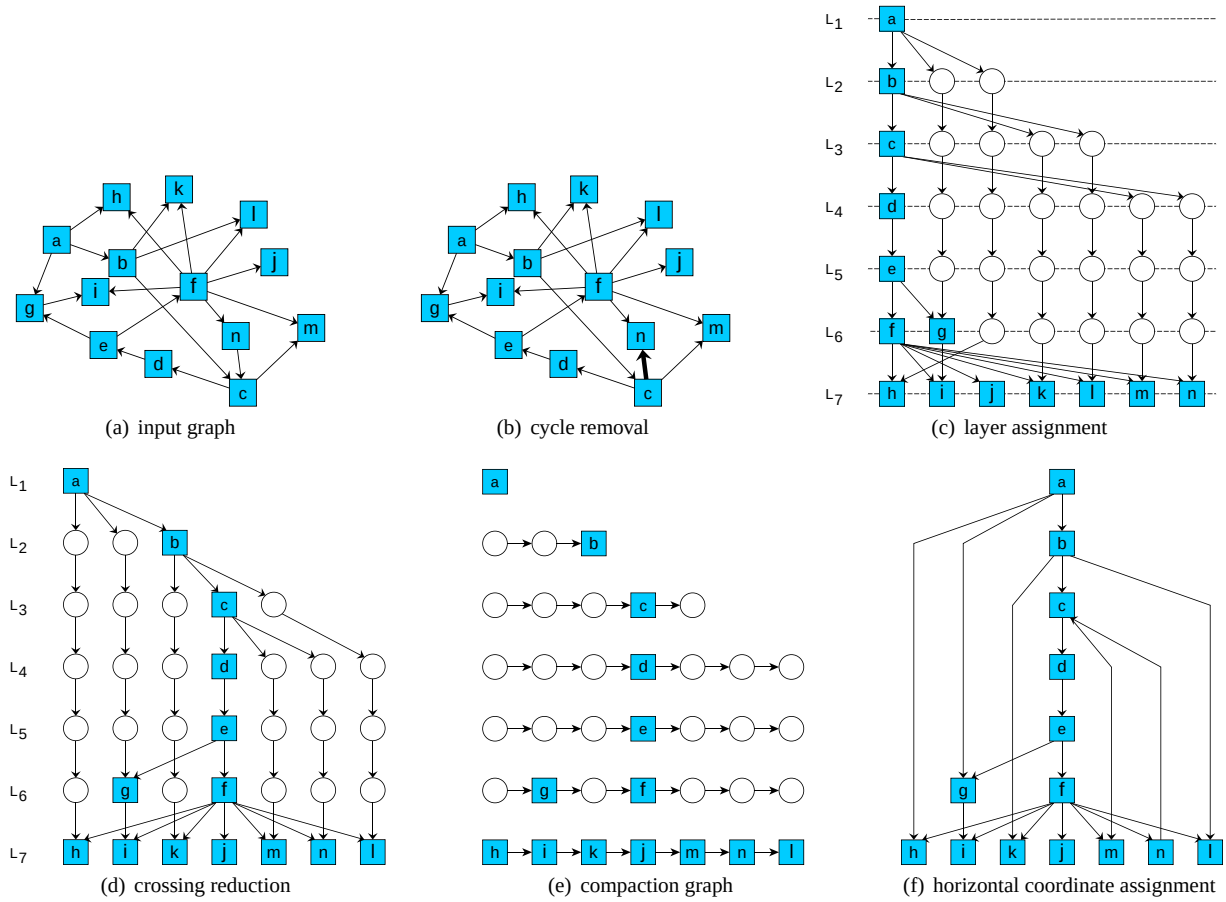


Figure 8: The different phases of Sugiyama's framework. Dummy nodes are drawn by white circles.

a, b consecutive in L_i, $1 \leq i \leq h\}$). It gives the left-to-right order of the nodes in a layer and hence defines a total ordering for those nodes (see Fig. 8(e)). Note, that crossing minimization is NP-hard [Garey and Johnson 1983].

4. **Horizontal Coordinate Assignment**: Finally, the horizontal coordinate assignment phase, uses the compaction graph to calculate an x-coordinate for each node. Long edges are represented by polygonal lines with the dummy nodes as intermediate points. Thus, in order to reduce the number of edge bends, all dummy nodes belonging to the same long edge should preferably be aligned vertically. After the final layout the reversed edges are restored to their original direction and the dummy nodes are removed (see Fig. 8(f)).

Crossing reduction is usually done by a layer-by-layer sweep where each step minimizes the number of edge crossings for a pair of adjacent layers. It is performed as follows: We start with an arbitrary node order of the first layer L_1. Then, while the node ordering of layer L_{i-1} is kept fixed, we put the nodes of L_i in an order that minimizes crossings. This step is called *one-sided two-layer crossing minimization* and is repeated for $i = 2, .., h$. After we have processed the bottommost layer, we reverse the sweep direction and go from bottom to top. These steps are repeated until no further crossings can be eliminated for a certain number of iterations.

The two-layer crossing minimization is NP-hard [Eades and Wormald 1994]. An established heuristic strategy to tackle this problem is the following: The position of a node v in layer L_i depends on the position of its adjacent nodes in layer L_{i-1}. First, a measure is calculated for each node v in L_i. The measure of a node v is e.g. the barycenter (average) [Sugiyama et al. 1981] or median [Eades and Wormald 1994] of the positions of its neighbors in the above layer. We get the positions of the nodes in L_i by sorting them according to their measure.

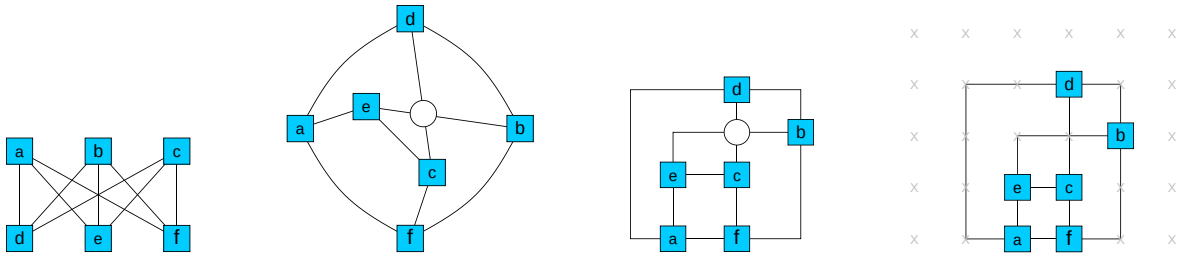
The complexity of algorithms implementing Sugiyama's framework heavily depends on the number of dummy nodes inserted. Although this number can be minimized efficiently, it may still be in the order of $\Theta(|V||E|)$ [Frick 1997]. The overall time complexity is therefore $O(|V||E|\log|E|)$.

4.3 Topology-Shape-Metrics Approach

The most effective concept for orthogonal layouts of undirected graph structures is the topology-shape-metrics (TSM-) approach. This concept has been introduced by Tamassia [Tamassia 1987; Tamassia et al. 1988] and later-on refined and extended by several groups of authors [Fößmeier and Kaufmann 1996; Klau and Mutzel 1998; Didimo and Liotta 1998; Bertolazzi et al. 2000].

The topology-shape-metrics approach consists of three phases:

1. **Planarization**: The planarization phase determines the topology of a drawing which is described by a planar embedding. Non-planar graphs are made planar by introducing dummy nodes that represent edge crossings (see Fig. 9(b)). Common



(a) input graph (b) planarization (c) orthogonalization (d) compaction

Figure 9: The three phases of the TSM-approach. The white dummy node represents a crossing.

approaches try to minimize the number of edge crossings during this phase.

2. **Orthogonalization**: The orthogonalization phase fixes the shapes of the edges in a drawing. It therefore determines edge bends and angles between adjacent edges. In orthogonal drawings each edge is drawn as a sequence of horizontal and vertical line segments. Thus, each angle is a multiple of 90° (see Fig. 9(c)). Orthogonalization algorithms usually try to minimize the number of edge bends.
3. **Compaction**: The compaction phase determines the final coordinates of graph elements such that nodes and bends are placed on grid points. Finally, the inserted dummy nodes are removed (see Fig. 9(d)). During this phase most approaches try to minimize the area or the total edge length of the drawing.

In [Tamassia 1987] Tamassia introduced a network flow based approach which computes a bend-minimal orthogonal point drawing for plane 4-graphs. We use the Kandinsky model [Fößmeier and Kaufmann 1996] for the orthogonalization which is an extension of Tamassia's algorithm and copes with general planar graphs and nodes of prescribed size [Di Battista et al. 1999a]. Even though the complexity for finding a bend-minimal solution in the Kandinsky model is not known, there is an effective network flow formulation which has at most twice the number of bends of the optimal solution [Eiglsperger 2003]. Furthermore there is a heuristic network solver that produces satisfying results in practice [Eiglsperger 2003; Wiese et al. 2002] in time $O(n^2 \log n)$. There are several extensions of the Kandinsky model that are applicable to our purpose, e.g. the usage of prescribed angles and bends [Brandes et al. 2002] as well as the usage of prescribed edge shapes [Eiglsperger et al. 2004; Eiglsperger 2003].

4.4 Discussion

Both, the layered approach and the TSM-approach, have their advantages. The layered approach is especially suited to satisfy the FLOW criteria. It is highly adaptable and there are sophisticated approaches to handle requirement CLUSTER. The TSM-approach is particularly suited to satisfy aesthetics ORTHOGONAL. Since edges are paths of horizontal/vertical segments it also optimizes aesthetics ANGLE. Furthermore, NODE_SIZE can be realized in a more natural way than for layered approaches. There are sophisticated methods to produce mixed upward drawings which yield less crossings than Sugiyama-based approaches [Eiglsperger 2003]. Both are able to avoid overlaps and thus optimize aesthetics OVERLAP. Our idea is to combine both concepts to take advantage of synergies. More precisely, we take Sugiyama's approach to create a planarization and then continue with the TSM-approach.

5 New Approach

In this section we present our new approach for the automatic layout of activity diagrams. Besides the common aesthetics ORTHOGONAL, OVERLAP, CROSSING, BEND, AREA, FLOW it is also able to consider requirements like CLUSTER, PARTITION, BIMODALITY, NODE_SIZE, LABELING and TWO_SIDED_NODES. To our knowledge there is no existing algorithm that is able to handle such a complex combination of requirements.

5.1 Input and Interface

First we introduce the interface to our visualization approach. The input of the layout algorithm is:

- a mixed compound graph $G_C = (G,T)$, $G = (V,E_G)$, $T = (V,E_T)$, $V = B \cup C$, $E_G = E_D \cup E_U$
- a set of labels L,
- a function $s : B \cup L \to \mathbb{N} \times \mathbb{N}$ denoting the size of the nodes and labels in the drawing,
- a function $p_y \colon V \to \mathbb{N}$ assigning each node $v \in V$ the row index of the corresponding partition cell,
- a function $p_x \colon V \to \mathbb{N}$ assigning each node $v \in V$ the column index of the corresponding partition cell.

To each base node one of the following node types is assigned: initial node, final node, expansion node, object node, activity node, fork/join node, decision/merge node or note.

In a transformation step the activity diagram graph is transformed into a suitable input for the layout algorithm. Due to semantic reasons, the input graph does never contain self-loops or multiedges.

The algorithm can be customized to fit individual user preferences and different views on a diagram. The user can, for example, decide if PARTITION should be applied, if CLUSTER is more important than aesthetics FLOW or if FLOW should not be considered at all.

Note, that in activity diagrams the edges of E_D are drawn downward instead of upward. All edges of E_G are directed except edges incident to notes. However, only directed edges that are added to E_D are drawn downward. We propose to add all edges to E_D that are not connected to a node of type initial node, final node or note. It is not always possible to fully satisfy aesthetics FLOW, but our algorithm guarantees BIMODALITY for each base node.

5.2 Overview

We continue with an overview of the single steps of our algorithm:

1. **Preprocessing**
   (a) Recall, that notes are attached to edges by dashed lines. We use a dummy node as attachment point (see Figure 14(c)) (NOTES). Notes attached to node elements are handled like usual nodes.
   (b) We replace each compound node $c \in C$ by two nodes c^t and c^b representing the top and bottom border of the corresponding cluster region. All incoming edges $(v,c) \in E_G$, $v \in V$ are connected to c^t and all outgoing edges $(c,v) \in E_G$, $v \in V$ to c^b.
   (c) If the subgraph induced by the nodes of B is not connected, we add additional edges between its connected components. We therefore choose an action node of each component and then connect those nodes by a spanning tree. Due to semantical reasons each component has at least one action node.
   (d) For each edge $e = (u,v) \in E_D$ we check if $p_y(u) \leq p_y(v)$. If not, e cannot be drawn downward and thus is removed from E_D and inserted into E_U.
2. **Planarization**
   will be described in Section 5.3.
3. **Orthogonalization**
   will be described in Section 5.4.
4. **Compaction**
   (a) First, labels are inserted into the orthogonalized graph as described in Section 5.5.
   (b) The compaction phase uses the fast constructive compaction algorithm described in [Eiglsperger and Kaufmann 2002] that is able to handle nodes of prescribed size (NODE_SIZE). An experimental study comparing different orthogonal compaction algorithms [Klau et al. 2001] shows that the results of different constructive compaction heuristics are almost the same when using a flow-based post-processing step. Thus, we use the flow-based visibility post-processing strategy described in [Eiglsperger 2003] to further optimize aesthetics AREA and EDGE_LENGTH. During the compaction all dummy nodes, except those representing labels, get unit size.
5. **Postprocessing**
   (a) All dummy nodes and edges inserted during the previous steps are removed.
   (b) Rectangles denoting cluster regions as well as the grid partition are inserted into the drawing. Finally, the rectangle denoting the border of the activity is drawn and the activity's label is placed (optionally).

5.3 Planarization

Our planarization strategy combines the layered approach of Sugiyama with a rerouting strategy known from the TSM-approach. This section is partly based on our previous work, see [Siebenhaller and Kaufmann 2006; Blochinger et al. 2006]. We start with describing the specific modifications of the different phases of Sugiyama's approach with respect to the identified requirements and aesthetics.

5.3.1 Layer Assignment and Cycle Removal

Common layer assignment approaches are designed to realize aesthetics FLOW. In our application the layering has to deal with requirements CLUSTER and PARTITION, too.

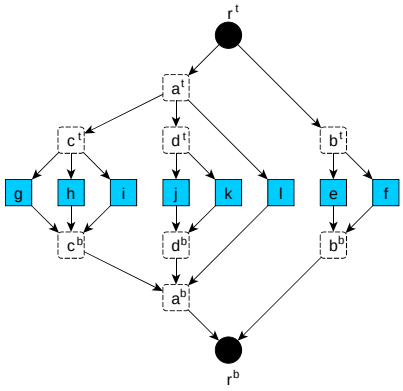


Figure 10: Nesting graph for the compound graph of Figure 6.

Let $L(v)$ denote the index of the layer of a node v ($L(v) = i \Leftrightarrow v \in L_i$, $1 \leq i \leq h$). For a compound node $c \in C$ the layering has to meet the following condition: For each base node $w \in B$ with $c \rightarrow_T^* w$ is $L(c^t) < L(w) < L(c^b)$ and for each compound node $d \in C$ with $c \rightarrow_T^* d$ is $L(c^t) < L(d^t) < L(d^b) < L(c^b)$.

Sander [Sander 1996] guarantees this by introducing the concept of a *nesting graph* G_n which has the node set $B \cup \{c^t, c^b : c \in C\}$ and the following edges:

- an edge (c^t, v) and (v, c^b) for each $(c,v) \in E_T$, $c \in C$, $v \in B$
- an edge (c^t, d^t) and (d^b, c^b) for each edge $(c,d) \in E_T$, $c,d \in C$

Note, that G_n is acyclic by construction (see Fig. 10). Each layer assignment on G_n guarantees the above condition.

Let $p_y^{\max}$ ($p_x^{\max}$) denote the number of grid rows (columns). We obtain a layering that preserves the grid partition by inserting $p_y^{\max}+1$ additional nodes p_j^y into G_n. A node p_j^y denotes the horizontal grid line separating grid row $j-1$ and j. The layering has to meet the following condition: For each base node $v \in B$ with $p_y(v) = j$ is $L(p_j^y) < L(v) < L(p_{j+1}^y)$ and for each compound node $c \in C$ with $p_y(c) = j$ is $L(p_j^y) < L(c^t) < L(c^b) < L(p_{j+1}^y)$. We realize this by simply inserting two edges (p_j^y, u) and (u, p_{j+1}^y) for each node u of G_n with $j = p_y(u)$. Furthermore, we insert edges (p_j^y, p_{j+1}^y), $1 \leq j \leq p_y^{\max}$ to guarantee that $L(p_i^y) < L(p_j^y)$ if $i < j$. Note, that the resulting graph is still acyclic. We call it the *partitioned nesting graph* $G_{pn} = (V_{pn}, E_{pn})$.

We also have to consider expansion nodes of expansion regions (see Section 2.1). The input nodes have to appear at the top border of the enclosing rectangle of the expansion region and the output nodes at the bottom border. Thus, during the layering all input nodes of an expansion region $c \in C$ are represented by c^t and all output nodes by c^b. The edges adjacent to those nodes are redirected correspondingly.

Now, we insert the edges of E_D. We have not yet checked if the set contains cycles. Even if E_D is acyclic, the insertion of an edge $e \in E_D$ into G_{pn} may cause a cycle, because for each $c \in C$ there is an implicit directed edge between the nodes c^t and c^b. Most directed cycles arise from modeling loops via decision nodes. Thus, outgoing edges of decision nodes do often participate in cycles. If we have to break a cycle, it is natural to take those edges. Hence, we assign a weight of 1 to each outgoing edge of a decision node, weight $5|E_G|+1$ to each edge $e \in E_{pn}$ and weight 5 otherwise. We

solve a weighted feedback arc (=edge) set problem, which consists of finding a minimum weight edge set $A \subseteq \{E_{pn} \cup E_D\}$, such that $(V, \{E_{pn} \cup E_D\} \setminus A)$ is acyclic. This problem is NP-hard [Garey and Johnson 1979] but several heuristics were proposed in practice [Flood 1990]. We use the heuristics of [Demetrescu and Finocchi 2003] that runs in time $O(|V||E_G|)$. Note, that the edge weights guarantee that $A \subseteq E_D$ and thus $A \cap E_{pn} = \emptyset$ which is necessary for a proper layering. All edges of $E_D \setminus A$ are inserted into G_{pn}. The edges of A are moved from E_D to E_U, because they cannot be drawn downward.

The edges of E_U are not yet inserted into G_{pn}. Since those edges need not to be directed downward the only restriction on the layering is that $L(u) \neq L(v)$ for each edge $e = (u,v) \in E_U$. Therefore, we calculate a topological order π of the nodes of G_{pn} and temporarily orient all edges $e \in E_U$ from the lower to the higher node index in π. This guarantees that G_{pn} remains acyclic after inserting those edges. The layer assignment can be done with a common layering approach, e.g. a longest path layering or the network-simplex layering [Gansner et al. 1993].

After the layer assignment, all edges $e \notin E_G$ are removed. We normalize the graph by replacing long edges by chains of dummy nodes and edges (see Section 4.2). Let p_j^x, $1 \leq j \leq p_x^{\max} + 1$ denote the vertical grid line which separates grid column $j-1$ and j. For each p_j^x we additionally insert a dummy node $u_i^{p_j^x}$ on each layer L_i, $1 \leq i \leq h$. Furthermore, for each compound node $c \in C$ we insert two dummy nodes $u_i^{c^l}$ and $u_i^{c^r}$ on each layer $L(c^t) \leq i \leq L(c^b)$ representing the left/right border of the cluster region. The number of dummy nodes and edges inserted during normalization is $O(|V||E_G|)$.

5.3.2 Crossing Reduction

Unlike traditional crossing reduction, we also have to consider requirements CLUSTER and PARTITION here.

Our crossing reduction is based on the method described in [Forster 2002]. First a *layer hierarchy tree* T_{L_i} is computed for each layer L_i, $1 \leq i \leq h$. T_{L_i} is the subgraph of T that is induced by all nodes relevant for layer L_i, that are all base nodes $v \in B$ with $L(v) = i$ and all compound nodes $c \in C$ with $L(c^t) \leq i \leq L(c^b)$.

Regarding clusters, there are two new restrictions during a one-sided two-layer crossing minimization step of layer L_{i-1} and L_i:

- All (compound and base) nodes of layer L_i belonging to the same compound node have to be consecutive.
- Let $c, d \in C$ denote two compound nodes with $L(c^t), L(d^t) \leq i-1$ and $i \leq L(c^b), L(d^b)$. Then the relative position of c and d must be the same on both layers.

In [Forster 2002] the following theorem has been shown:

Theorem 2 *An order of the base nodes has a minimal number of crossings if and only if the corresponding layer hierarchy tree has a minimal number of crossings at each node.*

Thus, during a one-sided two-layer crossing minimization step the number of crossings can be minimized by independently computing an order of the children for each compound node without loosing quality. The corresponding algorithm is based on a conventional algorithm for weighted one-sided two-layer crossing minimization.

The preservation of the relative position of two clusters is guaranteed by using a constraint crossing minimization [Forster 2005] where the order of some node pairs is already given. Thus, the relative ordering of two compound nodes can be preserved by inserting a constraint between them.

When the layer hierarchy trees have bounded degree the crossing reduction of a compound graph G_C runs in time $O(|V||E_G|)$ [Forster 2002]. Since G_C can include $O(|V||E_G|)$ dummy nodes and edges, the overall running time of the crossing reduction is $O(|V|^2|E_G|^2)$.

We use the following strategy to include partitions: During a one-sided two-layer crossing minimization step, we first sort all nodes of the non-fixed layer according to their $p_x(v)$ values in ascending order. Now, we simply apply the above algorithm to all nodes with the same p_x value (nodes in the same grid column) independently. Recall, that cluster regions do not intersect partition cells. Regarding the quality of the separate handling of grid columns we can state the following theorem:

Theorem 3 *The order of the base and compound nodes can be calculated independently for each column of the grid partition without loosing quality.*

Proof: Regarding grid columns as clusters with fixed positions, we can simply apply Theorem 2. □

Since compound nodes as well as columns of the grid partition are considered independently, the placement of the border nodes inserted during normalization is simple. When we process a compound node $c \in C$ in layer L_i, we simply restrict $u_i^{c^l}$ to be the leftmost node and $u_i^{c^r}$ to be the rightmost node. When we process the j-th grid column in layer L_i, the resulting node sequence is placed between $u_i^{p_j^x}$ and $u_i^{p_{j+1}^x}$.

Note, that our crossing reduction algorithm does not depend on the one-sided two-layer crossing minimization strategy. An optimal strategy would imply that our modified two-layer crossing minimization is optimal, too. However, this does not imply global optimality because the layer-by-layer sweep may introduce unnecessary bends. The overall running time of our crossing reduction is $O(|V|^2|E_G|^2)$.

5.3.3 Horizontal Coordinate Assignment

Let C_G denote the (Sugiyama based) compaction graph constructed as described in Section 4.2. Each cluster should be represented by an enclosing rectangle. Therefore we have to vertically align the left (right) border nodes $u_i^{c^l}$ ($u_i^{c^r}$), $L(c^t) \leq i \leq L(c^b)$ for each compound node $c \in C$ as well as the nodes $u_i^{p_j^x}$, $1 \leq i \leq h$ for each p_j^x, $1 \leq j \leq p_x^{\max} + 1$. The alignment is realized by simply mapping all nodes that should be aligned to the same node of C_G. Due to the modified crossing minimization this never introduces cycles in C_G.

We use C_G to perform the horizontal coordinate assignment with the algorithm proposed by Brandes and Köpf [Brandes and Köpf 2002]. It produces pleasing results and runs in linear time. The edge straightening reduces the input size of the following planarization step which is based on a sweep-line algorithm. Finally, all dummy nodes inserted during normalization are removed.

If the subgraph G_D induced by the edges of E_D is not yet connected, we connect it by adding additional edges of E_U to E_D. This is necessary for a common orientation of those edges. Therefore, we calculate a minimum weight spanning tree in which edges of E_D have lower weight. The edges of the spanning tree are called *skeleton edges*. The additional edges of added to E_D never introduce cycles in G_D (otherwise, the spanning tree can not be minimum).

Note, that after the horizontal coordinate assignment we have a drawing that already satisfies important aesthetics and requirements like FLOW, CLUSTER and PARTITION. However, when we continue and take the resulting drawing as input for the TSM-approach, we can also include NODE_SIZE, LABEL, ORTHOGONAL, BIMODALITY and ANGLE in a natural way. In the TSM-approach nodes are not placed on fixed layers which additionally improves aesthetics AREA and EDGE_LENGTH. Aesthetics CROSSING can be further minimized by using a rerouting step for edges $e \in E_U$ (see Section 5.3.5). Furthermore, if CLUSTER is more important than FLOW, the rerouting guarantees a cluster drawing.

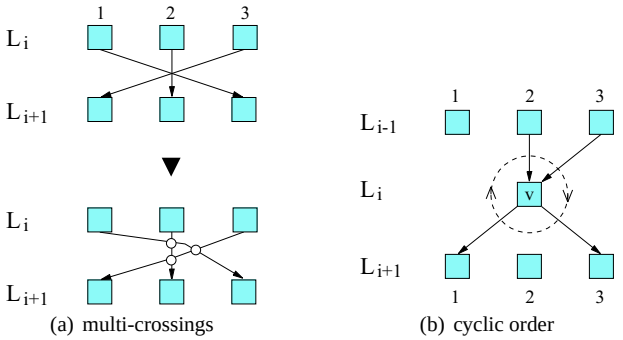


Figure 11: Planarization problems.

5.3.4 Creating the Planar Embedding

In this step we create a planar embedding of the drawing produced in the above steps. We "materialize" compound nodes and the grid partition in the following way: For each compound node $c \in C$, we know the coordinates c^t, c^b, $u_i^{c^l}$, $u_i^{c^r}$ of the top, bottom, left and right border. The cluster region of c is represented by placing a boundary rectangle on this coordinates. The boundary rectangle consists of four nodes with coordinates $(u_i^{c^l}, c^t)$,$(u_i^{c^l}, c^b)$, $(u_i^{c^r}, c^t)$ and $(u_i^{c^r}, c^b)$ respectively connected by two horizontal and two vertical edges. Note, that input and output nodes of expansion regions split the corresponding horizontal edges. Similar, we insert the grid partition graph into the drawing using the coordinates $u_i^{p_i^x}$ (p_i^y) for the vertical (horizontal) grid lines. In both cases, all vertical edges point downward and all horizontal edges to the right. The vertical edges are added to E_D.

To create a planar embedding, we have to replace crossings by nodes and give the cyclic order of the edges around each node. We realize this in the following way: First we detect crossings with a sweep-line algorithm. This can be done in time $O(|S|\log|S|+x)$ where S denotes the set of segments and x the number of crossings. The number of segments $|S|$ is $O(|V||E_G|)$. For each crossing, we store its coordinates and the two participating edges. A crossing is realized by splitting the corresponding edges with a dummy node. For an edge $e=(u,v)$ let $x_1^e,..,x_k^e$ denote the ordered sequence of crossings as they appear when walking along the edge from u to v. If we split e in this order, we always know the next segment to split (the i-th crossing of (u,v) splits segment (x_{i-1}^e, v)). If the crossings are processed arbitrarily, the determination of the segment to split demands a more complex data structure.

Note, that in the drawing of the graph, all edges are strictly monotone except the horizontal edges denoting the top/bottom border of cluster regions or the horizontal grid lines. Thus, we sort the crossings according to their y-coordinate. Crossings with the same y-value are additionally sorted according to their x-value. Hence, horizontal edges are processed in the ordered crossing sequence, too.

A problem arising during this step are multi-crossings which are points where more than two edges cross. They are allowed in Sugiyama's approach but not in the TSM-approach. Multi-crossings are detected by the sweep-line algorithm and then replaced as shown in Fig. 11(a).

After inserting the dummy nodes which represent crossings, the cyclic order of the edges around a node can easily be determined by means of the positions of its adjacent nodes (see Fig. 11(b)).

5.3.5 Rerouting

The result of the above step is a mixed upward p-planar embedding of G_C. Next, we perform a rerouting step based on shortest paths computations in the dual graph [Di Battista et al. 1999b]. Note, that we always reroute a complete edge and not just segments. The edges of E_U need not be drawn downward but they are also directed (except edges adjacent to notes) and thus have to be considered to satisfy BIMODALITY. Let $E_R = \{e | e=(u,v) \in E_G, L(u) > L(v)\}$ denote the set of edges reversed during the layering. Since the planarization is based on a layered drawing, the subgraph induced by edges of $E \setminus E_R$ is upward planar (after inserting crossing nodes) and thus the edge ordering around the nodes is bimodal. To guarantee BIMODALITY for all base nodes, we have to reroute all edges of E_R. Let E_x denote the set of edges that crosses a region boundary twice. To get a cluster drawing we have to reroute those edges, too. Since the following rerouting approach cannot guarantee that edges are inserted upward planar, we only reroute edges $e \in E_x \cap E_D$ if requirement CLUSTER is more important than FLOW. We can also improve aesthetics CROSSING by rerouting all edges of E_U and check if there is a route with fewer crossings. If we have to reroute a skeleton edge, we create a clone of it and reroute the clone. This is necessary because the skeleton edges have to be inserted downward and thus are not allowed to be rerouted. The copy representing the skeleton edge is removed after orthogonalization.

Summarizing, the special requirements for the rerouting are the following: An edge route have to guarantee that

- the edge crosses a region boundary at most once (CLUSTER),
- the edge is inserted bimodal (BIMODALITY),
- the edge does not leave the outer border of the grid partition.

A modification of the dual graph that guarantees the first point is described in [Di Battista et al. 2002]. First a common dual graph D_G of G_C is computed. Each node of D_G is enclosed by a certain set of boundary rectangles of clusters (see Fig. 12). Let c denote the innermost cluster that encloses a node u_f of D_G. Then each node on the path $r \rightarrow_T^* c$ encloses u_f, too (r is the root of T). When we route an edge (v,w) we first calculate the undirected path $v \rightarrow_T^* w = v, c_1, .., c_k, w$. Each edge (u_f, u_g) of D_G is handled as follows: Let c_i denote the innermost cluster that encloses u_f and c_j the innermost cluster that encloses u_g. If c_i or c_j is not contained in $\{c_1,..,c_k\}$, (u_f,u_g) is removed temporarily. Otherwise, if $i<j$ we orient (u_f,u_g) from u_f to u_g (edge can only be passed from u_f to u_g) and if $i>j$ we orient it from u_g to u_f. If $i=j$, then (u_f,u_g) is bidirectional. Now, we compute a directed shortest path between the set of face nodes incident to v and the set of face nodes incident to w. Edge (v,w) is inserted by following the shortest path and replacing crossings with dummy nodes. The old route of the

edge is discarded. After each step, the dual graph has to be updated and the temporarily removed and oriented edges are restored. The running time of this step is $O(|E_G| \cdot x + |E_G|^2 \cdot (|C| + |P|))$, where x denotes the number of crossings and $|P| = p_y^{\max} \cdot p_x^{\max}$ the number of partition cells.

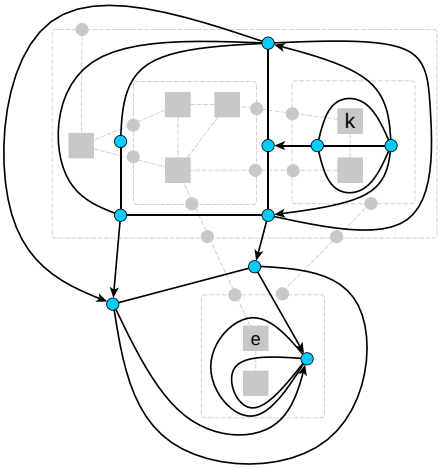


Figure 12: The modified dual graph for the example shown in Fig. 6 when routing edge (k,e). Edges without arrows are bidirectional. The embedding of the underlying graph is shown in the background (gray nodes and dashed edges).

To realize BIMODALITY we restrict the shortest path to start (end) only at face nodes incident to v (w) that allow a bimodal edge insertion.

The third point has only to be considered, if we use partitions. If so, all nodes and edges lie inside the grid partition. The outer border of the grid partition is a closed region. Thus, we simply remove the node representing the outer face from the dual graph.

If FLOW is more important than CLUSTER, the result of the rerouting is a mixed upward p-planar embedding. Otherwise, the result is a mixed upward pc-planar embedding. In both cases the base nodes are bimodal.

5.4 Orthogonalization

The orthogonalization phase calculates the edge bends and the angles between adjacent edges. Besides aesthetics BEND, ORTHOGONAL and ANGLE, it has to observe FLOW, CLUSTER, BIMODALITY, PARTITION and TWO_SIDED_NODES. Its input is the mixed upward p(c)-planar embedding calculated in the above step where all dummy nodes representing crossings are marked accordingly. The whole phase is divided into three steps:

1. Let G_D denote the subgraph induced by the downward edges $e \in E_D$. In the first step, we assign a fixed downward shape to each edge of G_D. Therefore, we calculate a tail- and a head-shape for each edge segment. The tail-shape (head-shape) depends on the type of the corresponding source (target) node. The algorithm iteratively chooses a node of G_D and assigns the head-shape to the incoming edges and the tail-shape to the outgoing edges. The different shapes are shown in Fig. 13. Note, that if one of the edge segments adjacent to a crossing node (see Fig. 13(c)) represents a vertical segment of a boundary rectangle or the grid partition graph it is chosen to be the straight segment. The final shape of an edge segment is constructed by concatenating the corresponding tail- and head-shape. In [Eiglsperger et al. 2004] we successfully applied a similar strategy and showed that there is always a valid drawing in which the edge shapes correspond to the calculated shape.
2. In the second step we consider all edges. To satisfy BIMODALITY and TWO_SIDED_NODES we put restrictions on the angles between specific edges. Angles between pairs of consecutive incoming and outgoing edges around nodes are restricted to be at least 90° (see Fig. 14(a)) (BIMODALITY). For fork and join nodes those angles are fixed to 180° as shown in Fig. 14(b) (TWO_SIDED_NODES). The orientation of fork/join nodes is determined after orthogonalization. If edges are connected horizontally (vertically) the fork/join is drawn vertically (horizontally). Note, that if there is at least one edge $e \in E_D$ adjacent to a fork/join node it can always be drawn horizontally. The angles around a dummy node caused by an edge note (see Section 5.2, step 1a) are fixed to the values shown in Fig. 14(c) and the angles of input/output nodes of expansion regions to the values shown in Fig. 14(d). Furthermore, Fig. 14(e) shows the angles around nodes of the grid partition graph. The angles around nodes of a boundary rectangle correspond to that of a grid partition graph $P^G_{1,1}$. Note, that the angle assignment is consistent with the fixed shapes calculated in the first step. This is necessary because there could be both edge types on a node.
3. In the third step, we calculate a shape for each edge of E_U. The fixed shapes and angles assigned in the previous steps are not allowed to change. Furthermore, all edge segments of a boundary rectangle or the grid partition graph are not allowed to bend. Therefore, we assign high bend costs to them. We transform fixed shapes into fixed angles and bends. The shape calculation of the edges of E_U is done with the network flow approach described in [Fößmeier and Kaufmann 1996] applying the modifications of [Brandes et al. 2002; Eiglsperger 2003] that guarantees the conservation of prescribed angles and bends. The algorithm has running time $O((|V| + x)^2 \log(|V| + x))$, where x denotes the number of crossings.

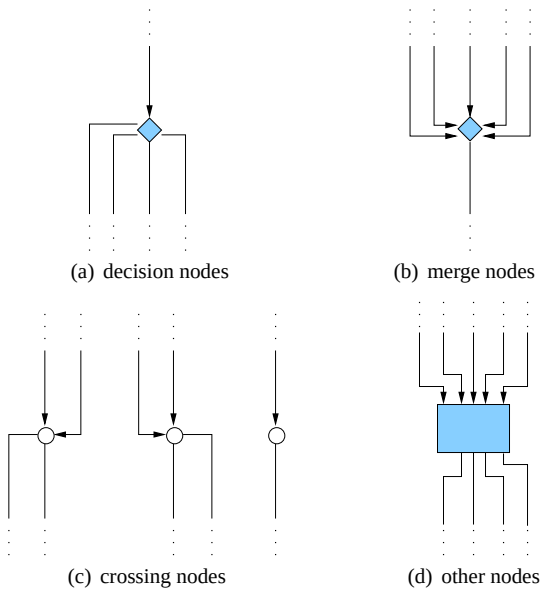
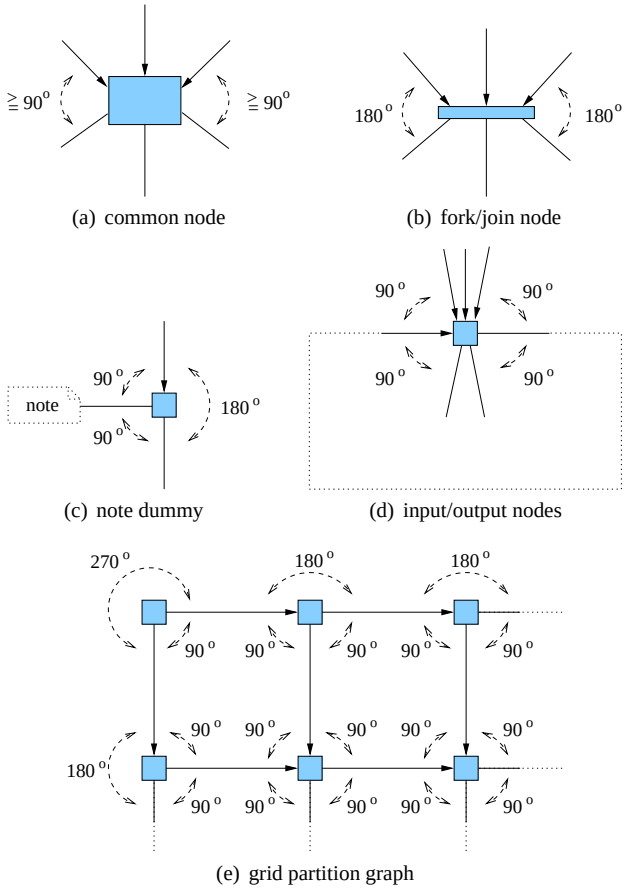


Figure 13: The different shapes of downward edges.



(a) common node (b) fork/join node (c) note dummy (d) input/output nodes (e) grid partition graph

Figure 14: The different angles.

5.5 Placement of Labels

Labels can be placed at nodes, edges, clusters, (grid) partitions and activities (LABEL). Node labels are placed inside a node and thus do not need a special handling. Activity labels are handled during a postprocessing step.

Edge labels are placed as follows: After orthogonalization, edges are paths of horizontal and vertical segments. For each labeled edge e we choose one of its middle segments and split it into two parts by inserting a dummy node. The dummy node represents the label l_e of e and thus has size $s(l_e)$. We always place labels horizontally as shown in Fig.15 (HORIZONTAL_LABELS). After compaction the dummy nodes are removed and replaced by the corresponding label. If there are edge labels with preferred placement at the source or target node, they are placed after the final layout by an efficient map labeling algorithm [Wagner and Wolff 1998].

Analogously, for labels of clusters and columns of the grid partition we split the edge representing the top side of the enclosing rectangle (see Fig. 15(a)). For labels of the grid rows we split the edge representing the left border (see Fig. 15(b)).

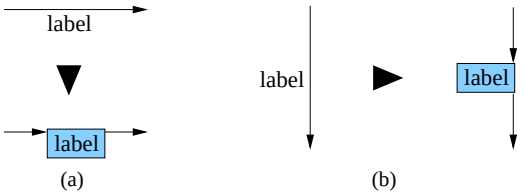


Figure 15: Placement of labels on a horizontal/vertical edge segment.

6 Discussion and Conclusion

First we discuss the time complexity of our new approach. The planarization step is dominated by the running time of the crossing reduction with $O(|V|^2|E_G|^2)$ and the rerouting of edges with $O(|E_G| \cdot x + |E_G|^2 \cdot (|C| + |P|))$. The orthogonalization step takes time $O((|V| + x)^2 \log(|V| + x))$ and the compaction step including the flow-based post-processing has quadratic running time. Graphs representing activity diagrams are always sparse, thus we can assume that $|E_G|$ and x are linear in $|V|$. Furthermore, the number of clusters $|C|$ and partition cells $|P|$ is $O(|V|)$. Under this assumption the overall running time is $O(|V|^4)$.

In our complexity analysis we assume that the number of dummy nodes and edges inserted during normalization is $O(|V||E_G|)$. We can further improve the time complexity by using the efficient implementation of Sugiyama's algorithm described in [Eiglsperger et al. 2005]. It avoids introducing dummy vertices for each layer that an edge spans and reduces the number of dummy nodes and edges to $O(|V| + |E_G|)$ without loosing quality. Thus, the time complexity of the crossing minimization is reduced to $O(|E_G|^2)$ which results in a new overall running time of $O(|V|^3)$. We used a common Sugiyama algorithm in this work because it allows an easier description of the crossing minimization and horizontal coordinate assignment phase. However, the implementation of our approach uses the fast variant.

For the implementation we used Java 1.5 and the yFiles library [Wiese et al. 2002]. Some results can be seen in the appendix. For diagrams with about 50 node and 80 edge elements we typically need a runtime of less than 3 seconds on a 3 GHz Pentium IV System with 1 GB RAM.

In this work we identified requirements and aesthetics for the layout of activity diagrams. We presented a new automatic layout approach for activity diagrams which combines Sugiyama's algorithm with the topology-shape-metrics approach to take advantage of synergies. Our approach satisfies a complex set of different aesthetics and requirements like PARTITION, CLUSTER, FLOW, CROSSING, ORTHOGONAL, BIMODALITY, TWO_SIDED_NODES, OVERLAP and AREA which makes it also applicable to other kinds of diagrams, e.g. class or process diagrams (see Fig. 18).

Future work might comprise an interactive version of this approach where the user can continuously change the diagram by adding or removing elements. When we use our current approach to calculate a new drawing of an updated diagram, this drawing can be significantly different from the previous one even for small updates of the diagram structure. The main goal of an interactive approach is the preservation of the user's mental map [Eades et al. 1991] of the diagram. Here, the concepts of sketch-driven layout developed in [Brandes et al. 2002] fit into this framework. However, the concepts have to be extended to satisfy the complex requirements of activity diagrams.

References

AMBLER, S. W. 2005. *The Elements of UML 2.0 Style*. Cambridge University Press.

BERTOLAZZI, P., BATTISTA, G. D., AND DIDIMO, W. 2000. Computing orthogonal drawings with the minimum number of bends. *IEEE Transactions on Computers 49*, 8, 826–840.

BLOCHINGER, W., KAUFMANN, M., AND SIEBENHALLER, M. 2006. Visualization aided performance tuning of irregular task-parallel computations. *Information Visualization 5*, 2, 81–94.

BRANDES, U., AND KÖPF, B. 2002. Fast and simple horizontal coordinate assignment. In *Proceedings of the 9th Symposium on Graph Drawing (GD '01)*, Springer, vol. 2265 of *LNCS*, 31–44.

BRANDES, U., EIGLSPERGER, M., KAUFMANN, M., AND WAGNER, D. 2002. Sketch-driven orthogonal graph drawing. In *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, Springer, vol. 2528 of *LNCS*, 1–12.

CASTELLO, R., MILI, R., AND TOLLIS, I. G. 2002. Automatic layout of statecharts. *Software – Practice and Experience 32*, 1, 25–55.

CASTELLO, R., MILI, R., AND TOLLIS, I. G. 2002. A framework for the static and interactive visualization of statecharts. *Journal of Graph Algorithms and Applications 6*, 3, 313–351.

COLEMAN, M. K., AND PARKER, D. S. 1996. Aesthetics-based graph layout for human consumption. *Software – Practice and Experience 26*, 12, 1415–1438.

DEMETRESCU, C., AND FINOCCHI, I. 2003. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters 86*, 3, 129–136.

DI BATTISTA, G., DIDIMO, W., PATRIGNANI, M., AND PIZZONIA, M. 1999. Orthogonal and quasi-upward drawings with vertices of prescribed size. In *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, Springer, vol. 1731 of *LNCS*, 297–310.

DI BATTISTA, G., EADES, P., TAMASSIA, R., AND TOLLIS, I. G. 1999. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall.

DI BATTISTA, G., DIDIMO, W., AND MARCANDALLI, A. 2002. Planarization of clustered graphs (extended abstract). In *Proceedings of the 9th Symposium on Graph Drawing (GD '01)*, Springer, vol. 2265 of *LNCS*, 60–74.

DIDIMO, W., AND LIOTTA, G. 1998. Computing orthogonal drawings in a variable embedding setting. In *Proceedings of the 9th Annual International Symposium on Algorithms and Computation (ISAAC'98)*, vol. 1533 of *LNCS*, 79–88.

EADES, P., AND WORMALD, N. C. 1994. Edge crossings in drawings of bipartite graphs. *Algorithmica 11*, 379–403.

EADES, P., LAI, W., MISUE, K., AND SUGIYAMA, K. 1991. Preserving the mental map of a diagram. In *Proceedings of Compugraphics '91*, 24–33.

EICHELBERGER, H. 2002. Evaluation-report on the layout facilities of UML tools. Tech. Rep. 298, Institut für Informatik, Würzburg University, July.

EICHELBERGER, H. 2002. SugiBib. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, Springer, vol. 2265 of *LNCS*, 467–468.

EIGLSPERGER, M., AND KAUFMANN, M. 2002. Fast compaction for orthogonal drawings with vertices of prescribed size. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, Springer, vol. 2265 of *LNCS*, 124–138.

EIGLSPERGER, M., GUTWENGER, C., KAUFMANN, M., KUPKE, J., JÜNGER, M., LEIPERT, S., KLEIN, K., MUTZEL, P., AND SIEBENHALLER, M. 2004. Automatic layout of uml class diagrams in orthogonal style. *Information Visualization 3*, 3, 189–208.

EIGLSPERGER, M., SIEBENHALLER, M., AND KAUFMANN, M. 2005. An efficient implementation of sugiyama's algorithm for layered graph drawing. In *Proceedings of the 12th Symposium on Graph Drawing (GD'04)*, Springer, vol. 3383 of *LNCS*, 155–166.

EIGLSPERGER, M. 2003. *Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach*. PhD thesis, Universität Tübingen.

FLOOD, M. M. 1990. Exact and heuristic algorithms for the weighted feedback arc set problem: A special case of the skew-symmetric quadratic assignment problem. *Networks 20*, 1–23.

FORSTER, M. 2002. Applying crossing reduction strategies to layered compound graphs. In *Proceedings of the 10th Symposium on Graph Drawing (GD'02)*, Springer, vol. 2528 of *LNCS*, 276–284.

FORSTER, M. 2005. A fast and simple heuristic for constrained two-level crossing reduction. In *Proceedings of the 9th Symposium on Graph Drawing (GD'04)*, Springer, vol. 3383 of *LNCS*, 206–216.

FÖSSMEIER, U., AND KAUFMANN, M. 1996. Drawing high degree graphs with low bend numbers. In *Proceedings of the 3rd International Symposium on Graph Drawing (GD'95)*, Springer, F. J. Brandenburg, Ed., vol. 1027 of *LNCS*, 254–266.

FRICK, A. 1997. Upper bounds on the number of hidden nodes in sugiyama's algorithm. In *Proceedings of the 4th International Symposium on Graph Drawing (GD'96)*, Springer, vol. 1190 of *LNCS*, 169–183.

GANSNER, E., KOUTSOFIOS, E., NORTH, S., AND VO, K.-P. 1993. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering 19*, 3, 214–230.

GAREY, M. R., AND JOHNSON, D. S. 1979. *Computers and Intractability, A guide to the theory of NP Completeness*. Freeman, New York.

GAREY, M. R., AND JOHNSON, D. S. 1983. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods 4*, 312–316.

KAUFMANN, M., AND WAGNER, D., Eds. 2001. *Drawing Graphs: Methods and Models*. LNCS Tutorial. Springer.

KLAU, G. W., AND MUTZEL, P. 1998. Quasi-orthogonal drawing of planar graphs. Tech. Rep. 98-1-013, Max-Planck-Institut für Informatik, Saarbrücken.

KLAU, G. W., KLEIN, K., AND MUTZEL, P. 2001. An experimental comparison of orthogonal compaction algorithms (extended abstract). In *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*, Springer, vol. 1984 of *LNCS*, 37–51.

OMG, 2004. UML 2.0 superstructure specification. `http://www.omg.org/technology/documents/formal/uml.htm` (accessed April 5, 2006).

PACH, J., AND WENGER, R. 1998. Embedding planar graphs at fixed vertex locations. In *Proceedings of the 6th International Symposium on Graph Drawing (GD'98)*, Springer, vol. 1547 of *LNCS*, 263–274.

PURCHASE, H. C., COHEN, R. F., AND JAMES, M. 1997. An experimental study of the basis for graph drawing algorithms. *ACM Journal of Experimental Algorithmics 2*, 4.

PURCHASE, H. C. 1997. Which aesthetic has the greatest effect on human understanding? In *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, Springer, vol. 1353 of *LNCS*, 248–261.

SANDER, G. 1996. Layout of compound directed graphs. Tech. Rep. A/03/96, Universität des Saarlandes, FB 14 Informatik, 66041 Saarbrücken, June.

SHEGOGUE, D., AND ZHENG, W. J. 2005. Integration of the gene ontology into an object-oriented architecture. *BMC Bioinformatics 6*, 113.

SIEBENHALLER, M., AND KAUFMANN, M. 2006. Mixed upward planarization - fast and robust. In *Proceedings of the 13th Symposium on Graph Drawing (GD'05)*, Springer, vol. 3843 of *LNCS*, 522–523.

SIX, J. M., AND TOLLIS, I. G. 2002. Automated visualization of process diagrams. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, Springer, vol. 2265 of *LNCS*, 45–59.

SUGIYAMA, K., TAGAWA, S., AND TODA, M. 1981. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics SMC-11*, 2, 109–125.

TAMASSIA, R., BATTISTA, G. D., AND BATINI, C. 1988. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics 18*, 1, 61–79.

TAMASSIA, R. 1987. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing 16*, 3, 421–444.

WAGNER, F., AND WOLFF, A. 1998. A combinatorial framework for map labeling. In *Proceedings of the 6th International Symposium on Graph Drawing (GD'98)*, Springer, vol. 1547 of *LNCS*, 316–331.

WIESE, R., EIGLSPERGER, M., AND KAUFMANN, M. 2002. yfiles: Visualization and automatic layout of graphs. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, Springer, vol. 2265 of *LNCS*, 453–454.

WITTENBURG, K., AND WEITZMAN, L. 1997. Qualitative visualization of processes: Attributed graph layout and focusing techniques. In *Proceedings of the 4th International Symposium on Graph Drawing (GD'96)*, Springer, vol. 1190 of *LNCS*, 401–408.

A Examples

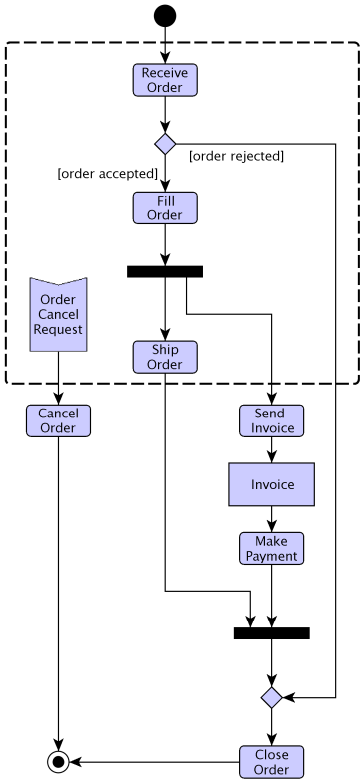


Figure 16: UML activity diagram drawn with our new layout approach. It shows another view of the example of Figure 1.

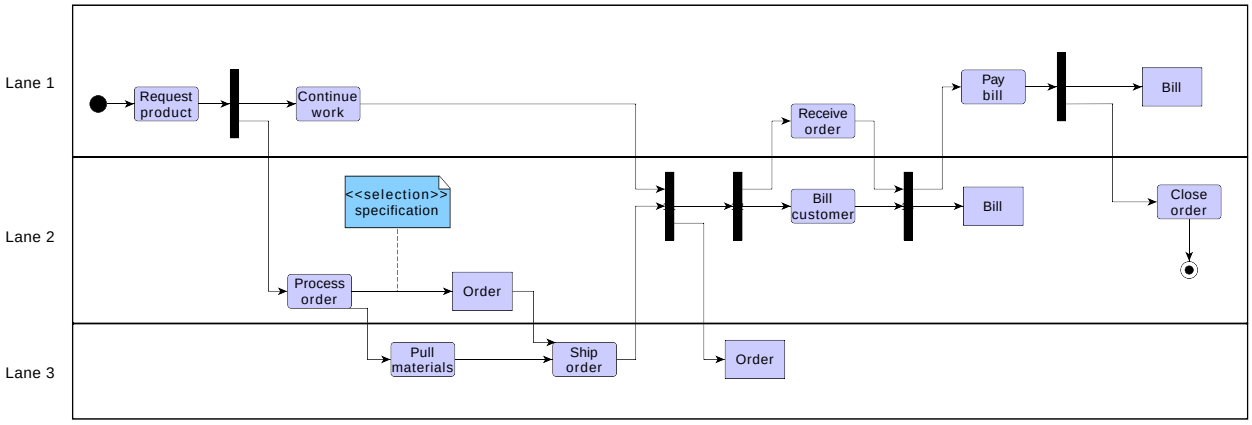


Figure 17: Another UML activity diagram taken from the Schematic tool page (http://www.hypergraphics.co.uk).

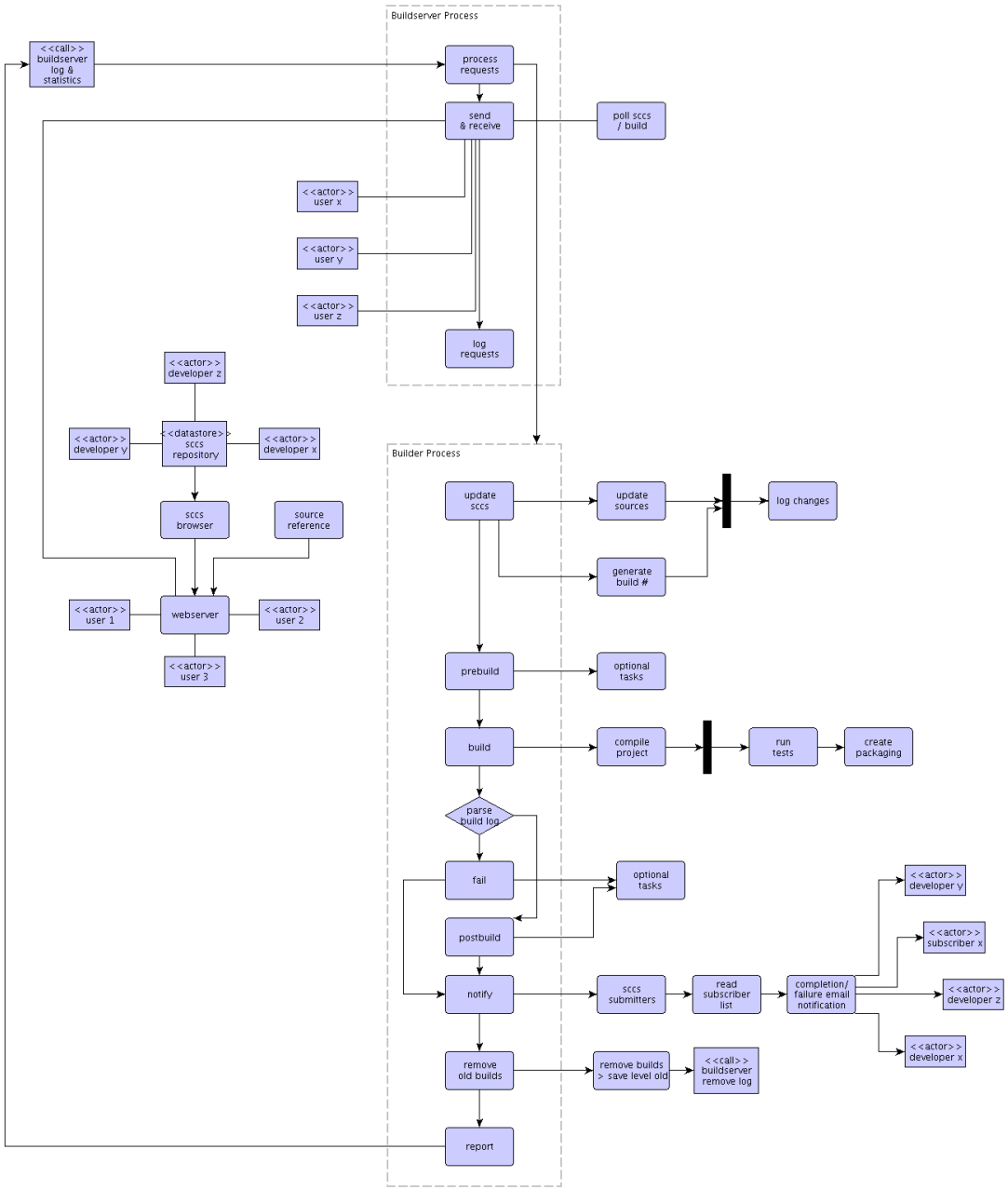


Figure 18: Our layout approach applied to a process diagram. The diagram was taken from the CABIE project page (http://cabie.tigris.org).