A Topology-Shape-Metrics Approach for the Automatic Layout of UML Class Diagrams*

Markus Eiglsperger† Universität Tübingen
Michael Kaufmann‡ Universität Tübingen
Martin Siebenhaller§ Universität Tübingen

Abstract

Class diagrams are among the most popular visualizations for object oriented software systems and have a broad range of applications. In many settings it is desirable that the placement of the diagram elements is determined automatically, especially when the diagrams are generated automatically which is usually the case in reverse engineering. For this reason the automatic layout of class diagram gained importance in the last years. Current approaches for the automatic layout of class diagrams are based on the hierarchic graph drawing paradigm. These algorithms produce good results for class diagrams with large and deep structural information, i.e., diagrams with a large and deep inheritance hierarchy. However, they do not perform satisfactorily in absence of this information. We propose in this work a new algorithm for automatic layout of class diagram which is based on the topology-shape-metrics approach. The algorithm is an adaption of sophisticated graph drawing algorithms which have proven their effectiveness in many applications. The algorithm works as well for class diagrams with rich structural information as for class diagrams with few or no structural information. It improves therefore the existing algorithms significantly. An implementation of the algorithm is used in the reverse engineering tool *JarInspector*.

CR Categories: D.2.m [Software Engineering]: Miscellaneous G.2.2 [Discrete Mathematics]: Graph Theory—Graph Algorithms

Keywords: Graph drawing, UML Diagrams

1 Introduction

Class diagrams describe the static structure of an object oriented system. They denote how the system is structured rather than how it behaves. They are used extensively in software engineering, ranging from design and documentation of software systems to reverse engineering. According to [Booch et al. 1999] class diagrams are the most common diagram found in modeling object-oriented systems. Class diagrams mainly consist of classes and relationships between these classes. Although historically there were different notations for class diagrams today the Unified Modeling Language (UML) has emerged as standard for class diagrams. The UML [OMG 2001] defines a notation and a semantics for class diagrams. The UML was adopted by the Object Management Group (OMG) as a standard and virtually every tool which works with class diagrams follows the UML specification. Therefore we cover exclusively the UML notation in this work and will use the terms UML class diagram and class diagram interchangeable.

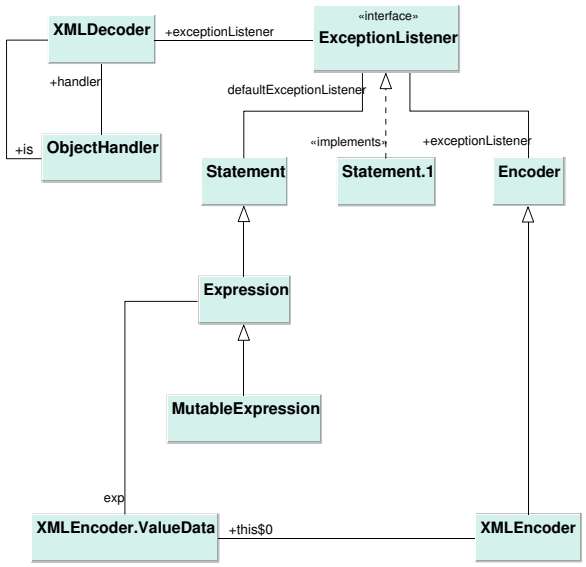


Figure 1: An UML class diagram

The UML specifies modeling elements and relations between these modeling elements. Relations between modeling elements can be visualized in one of the following three forms: paths, attachment, and containment. Mathematically we can view a class diagram as a graph with attached labels and clustering information. The graph encodes with its edges the first kind of visualization for relations, paths between the objects. Labels denote visualization by attachment. The containment structure of the diagram is denoted by the clustering of the graph. Figure 1 shows an example for a class diagram. Note that in a class diagram the classes themselves are represented by rectangles whose sizes are determined by the contained text.

Each graph drawing algorithm which can handle labels, clustering information and respects prescribed node sizes is therefore a potential layout algorithm for class diagrams. But not all graph drawing algorithms are equally suited for automatic layout of class diagrams. This is due to the fact that some special aesthetic criteria apply to class diagrams which are not necessarily covered by a graph drawing algorithm. The current automatic layout algorithms for class diagrams described in the literature are based on the *hierarchic graph drawing paradigm* and focus on the *direction of flow*

*Partially supported by DFG-Grant Ka812/8-1 and Ka812/8-2
†e-mail: eiglsper@informatik.uni-tuebingen.de
‡e-mail: mk@informatik.uni-tuebingen.de
§e-mail: siebenha@informatik.uni-tuebingen.de

ACM Symposium on Software Visualization, San Diego, CA

aesthetic criterion. Applied to class diagrams this aesthetic criterion says that all edges of some type should point in a common direction, i.e., all edges representing generalizations point upward. This approach has several drawbacks notably in the absence of flow information in the diagram. In this case the algorithm has no primary optimization goal and is in some sense "without orientation" which may have a dramatic effect on the result.

In this work we propose to use an alternative graph drawing paradigm, the topology-shape-metrics approach, as basis for a layout algorithm for class diagrams. This paradigm has been intensively studied in the last years in the area of graph drawing and has been used successfully in application areas like visualization of database schema. We show how this paradigm can be used for the visualization of class diagrams. Comparisons to existing algorithms show that the approach overcomes the above mentioned drawback and works well for all types of input.

The work is organized as follows: Aesthetic criteria for class diagrams are discussed in section 2. The hierarchic layout algorithms for class diagrams are reviewed and analyzed in section 3. The topology-shape-metrics approach for graph drawing upon which our algorithm is based is discussed in section 4. In section 5 we present our new approach for layouting class diagrams. We discuss the implementation of our algorithm in section 6. In section 7 we present a conclusion of the results. Appendix A contains an example diagram layouted with our algorithm.

2 Aesthetics of Class Diagrams

In this section we will discuss aesthetic criteria for class diagrams. These aesthetic criteria will guide us in the design of the automatic layout algorithm. Since class diagrams are drawings of graphs, we will first discuss aesthetics of drawings of graphs and then treat the special requirements of class diagrams.

There is no mathematical definition of aesthetics for a drawing of a graph, it can be defined informally that a drawing of a graph is more aesthetic than another drawing if it is "nicer" or "more readable". To mathematically describe aesthetics the concept of *aesthetic criterion* is used. An aesthetic criterion measures one isolated mathematically defined property of the drawing and defines rules for the values of this property. Examples for aesthetic criteria are [Di Battista et al. 1999]:

- minimize number of edge crossings CROSSING
- minimize number of bends BEND
- minimize number of node and edge overlap OVERLAP
- maximize number of orthogonal edges ORTHOGONAL
- maximize angular resolution RESOLUTION
- minimize edge length EDGE LENGTH
- minimize area AREA
- maximize rectangular aspect-ratio ASPECT RATIO
- maximize number of edges respecting flow FLOW
- maximize symmetry SYMMETRY

Some of the above criteria are contradicting, e.g., area and crossing minimization [Di Battista et al. 1999]. Therefore finding an aesthetic drawing of a graph can be seen as solving an multi-objective optimization problem, the objective function being a set of aesthetic criteria. Which aesthetic criteria apply to a given drawing depends on the semantics of the graph and user preference. Aesthetic criteria more specific to class diagrams are discussed in [Eichelberger 2002]:

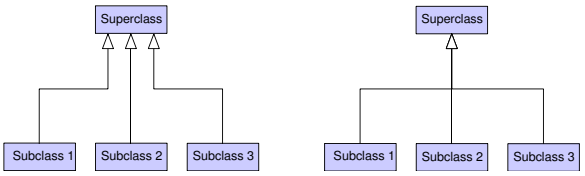


Figure 2: Generalization with distinct paths (left), hyperedge notation (right).

- use hyperedge notation for generalization relation HYPEREDGE,
- center n-ary association CENTER,
- place comment nodes and association classes near to the related model elements PROXIMITY.

There are few empirical studies about aesthetic criteria, and how they affect the readability of a drawing. Purchase performed a series of experiments [Purchase 1997; Purchase et al. 1997; Purchase et al. 2001] over the impact of different aesthetic criteria for the readability of drawings. The first two experiments [Purchase 1997; Purchase et al. 1997] were performed for graphs without semantics, the experiment [Purchase et al. 2001] was focused on class diagrams. In the experiments students had to answer questions about the graphs, resp. class diagrams in a limited amount of time. The hypothesis is that if a given aesthetic criterion influences the readability in some sense, this should be reflected by the quality of the answers.

For graphs without semantics the aesthetic criteria BEND, CROSSING, RESOLUTION, ORTHOGONAL and SYMMETRY have been studied. In the experiments aesthetic criterion CROSSING was found to be by far the most important, aesthetic criteria BEND and SYMMETRY having lesser importance and ORTHOGONAL and RESOLUTION with no significant effect. For class diagrams, the situation is less clear. In [Purchase et al. 2001] BEND and FLOW are evaluated to *decrease* the readability of a class diagram, ORTHOGONAL having no influence on the readability.

Another aspect of aesthetics of class diagrams is user preference. User preference is how an average user of class diagrams ranks aesthetic criteria. Note that user preference is only linked indirectly to the readability of a diagram, i.e., a user may prefer a certain visualization although it decreases readability, but in general it is assumed that a a diagram is more readable for a user when it satisfies his personal preferences. In an experiment on user preferences for class diagrams [Purchase et al. 2000] CROSSING ranked highest followed by BEND and HYPEREDGE.

As a conclusion we can say that it is not clear which is the most important aesthetic criterion for class diagrams. The aesthetic criteria CROSSING and BEND seem to play an important role. Although the importance for aesthetic criteria FLOW, HYPEREDGE and ORTHOGONAL is not backed by the above studies, partially because they have not been investigated, we think they play an important role for class diagrams and cannot be neglected. Because the situation is unclear we find it important for an automatic layout algorithm to be flexible enough to let the user choose the aesthetic criterion to optimize.

3 The Hierarchical Approach

In this section we discuss the hierarchical approach for the automatic layout of class diagrams. We first review the hierarchical

approach for drawing graphs and then discuss the extensions for class diagrams. Finally we analyze this approach and reveal some shortcomings of it.

3.1 The Algorithmic Framework

The hierarchical approach [Sugiyama et al. 1981], also called Sugiyama approach, is an algorithmic framework for drawing directed acyclic graphs. It consists of three phases: *layer assignment*, *crossing minimization* and *node placement*. In the layer assignment phase each node v in the graph is assigned a layer $l(v)$, such that all edges extend from a lower layer to a higher layer, i.e., $l(v) < l(w)$ for $(v,w) \in E$. In the second phase a permutation of each layer is computed, such that the number of crossings is minimized. The layering of the graph is usually normalized before this phase, what means that edges spanning more than one layer are split in a path by introducing dummy nodes, with each edge in the path spanning a single layer. In the third phase the coordinates for the nodes are determined. All nodes in a layer get the same y-coordinate and the x coordinates are assigned according to the permutation calculated in the second phase. Coordinates of dummy nodes become bends of the original edge. Usually one tries to minimize the number of bends, area and the length of the edges in this phase.

The hierarchic approach optimizes the FLOW aesthetic criterion in the first place, and the CROSSING criterion in the second place. Criteria BEND,AREA and EDGE LENGTH are considered with lower priority. There are variants of the algorithm leading to orthogonal drawings and generally the algorithm generates no overlaps. Therefore OVERLAP is fulfilled and ORTHOGONAL can be fulfilled if desired.

3.2 Class Diagrams

In general in class diagrams not all edges need to be drawn according to aesthetic criterion FLOW, often only generalization edges are drawn according to FLOW. Especially some edges may be undirected, e.g. symmetric associations or links to association classes and notes. Therefore the hierarchical approach cannot be applied directly to class diagrams, modifications are needed to handle these observations.

The work of Seemann [1997] was the first description of an enhanced version of the hierarchical approach which distinguishes between association and generalization edges. The algorithm works as follows: Nodes not adjacent to generalization edges are removed temporarily from the graph. Then the algorithm executes the first two phases of the hierarchical approach on the subgraph induced by the generalization edges. Then the removed nodes are inserted iteratively in the layers. As a last step node positions are calculated and the edges routed. Generalization edges are drawn as direct lines while association edges are drawn orthogonal. This algorithm is generalized by SugiBib [Eichelberger 2000; Eichelberger 2002] to handle additional constraints like aesthetic criteria HYPEREDGE, CENTER and PROXIMITY and to our knowledge it is the most sophisticated hierarchic layout algorithm for class diagrams at the moment. Furthermore it supports clustering and the edges denoting hierarchy are no longer restricted to generalizations. It rather divides the edges into hierarchic and non-hierarchic edges. Usually generalization edges are hierarchic and all others are not.

3.3 Discussion

The reason to use the hierarchical approach for the visualization of class diagrams is based on two assumptions:

1. There is a large set of hierarchical edges in the graph defining a deep hierarchy, and

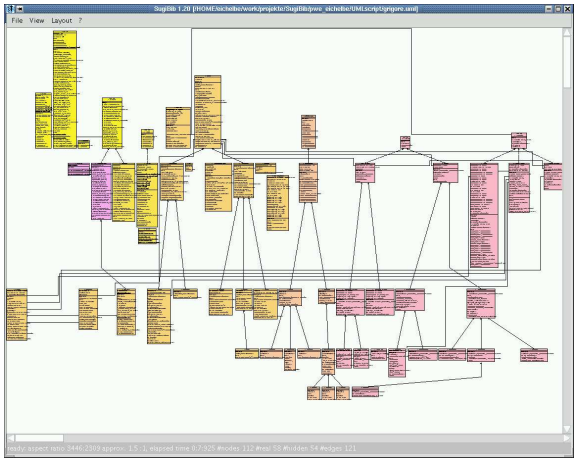


Figure 3: Example Layout of SugiBib, taken from `http://www2.informatik.uni-wuerzburg.de/staff/eichelberger/SugiBib/current.html`.

2. the users most important aesthetic criterion for the diagram is aesthetic criterion FLOW for the hierarchical edges.

These assumptions are often violated in practice. Class diagrams may contain no hierarchical structure at all, for example when the diagram contains only associations. Note that the extensive usage of inheritance in software engineering is discouraged. Gamma [1995] notes:

> Favor object composition over class inheritance.

Even diagrams with a lot of hierarchical edges may be a problem for the hierarchical approach, since these hierarchies are usually not deep. Software with deep inheritance hierarchies tend to be difficult to understand and hard to maintain. Booch [1994] reports that as a rule of thumb inheritance hierarchies in object-oriented software systems have a maximum depth of 7 ± 2. Empirical studies show that the value is even smaller, see, e.g. [Chidamber and Kemerer 1994], values less than three seem to be more realistic. This has as a consequence that only few layers are allocated by the above algorithms and when the size of the diagrams grow, the diagrams get wider but not higher resulting in diagrams with bad aspect ratio violating aesthetic criterion ASPECT RATIO. This can be avoided by using advanced layering algorithms which take the aspect ratio of the diagram into account. It is not clear how such a layering algorithm would look like, there are some approaches solving this problem, i.e., [Coffman and Graham 1972], but it is not evident how they integrate in the above layout algorithm for class diagrams.

Even if there are enough hierarchic edges in the diagram, the user might not want to optimize aesthetic criterion FLOW, therefore assumption 2 is often violated. As noted in section 2 aesthetic criterion CROSSING might influence the readability more than aesthetic criterion FLOW, i.e. the user might prefer a drawing with less crossings over a drawing with inheritance edges pointing upward. The hierarchic layout algorithms lack this flexibility by design.

A more technical argument is that for upward directed drawings the hierarchic graph drawing algorithm may produce more crossings than the upward planarization approach we suggest in section 5. In a study comparing the two algorithms [Eiglsperger and Kaufmann 2002] the upward planarization approach produced less crossings, especially for graphs with limited height. Note that we compared in this study the graph drawing algorithms and not the algorithms for drawing class diagrams, so these results have to be

interpreted with care, but nevertheless they suggest that the upward planarization approach might be superior to the hierarchical approach when minimizing the number of crossings.

4 The Topology-Shape-Metrics Approach

In this section we will review the topology-shape-metrics approach for graph drawing. We analyze results related to this approach which can be used for the automatic layout of class diagrams.

4.1 The Algorithmic Framework

The topology-shape-metrics approach origins from the seminal paper of Tamassia [1987]. The name topology-shape-metrics approach was introduced in [Di Battista et al. 1999]. Like the hierarchical approach the topology-shape-metrics approach is divided into several steps:

Planarization This step determines the topology of the drawing, which is described by a *planar embedding*. A graph is planar, if it has a drawing in the plane without edge crossings. Such a drawing divides the plane into regions, called *faces*. A planar embedding is a combinatorial description of the faces and contains for each face the sequence of edges contouring it. A planar embedding implicitly defines a cyclic ordering of the edges around a vertex. For non-planar graphs dummy nodes are inserted which represent crossings to make the resulting graph planar. Usually algorithms try to minimize the number of crossings.

Orthogonalization This step determines the angles and the bends in the drawing. Only multiples of 90^o are assigned as angles which ensures that the drawing is orthogonal. Usually algorithms try to minimize the number of bends in this step.

Compaction In this step the final coordinates are assigned to the nodes and to the edge bends. The dummy nodes introduced in the planarization step are removed. In this phase the main goal is to minimize the sum of the length of all edges and/or the area of the drawing.

4.2 State of the Art

As we have seen in section 1, automatic layout algorithms for class diagrams need to consider a lot of information beyond the pure graph structure like labels and clustering information. Also we have seen in section 2 that some special aesthetic criteria apply to class diagrams. The first algorithms based on the topology-shape-metrics approach did not have the complexity to handle all these issues. However the development in recent years filled this gap and we are now able to present an automatic layout algorithm for class diagrams based on the topology-shape-metrics approach. In the following we will list the relevant work which enables the automatic layout of class diagrams.

Node size: To keep node sizes fixed, the second and the third phase must meet special requirements. The only methods that guarantee an assigned node size rely on the `Kandinsky` model, also called podevsnef model. In this model the shape must conform to certain rules, the most important being that only one edge per side of the node has no bend. The `Kandinsky`-algorithm [Fößmeier and Kaufmann 1996] calculates bend-minimum drawings in the `Kandinsky` model. An alternative approach for a more restrictive model is described in [Bertolazzi et al. 2000]. In the third phase the assigned node size must be realized, the algorithms from [Battista et al. 1999; Eiglsperger and Kaufmann 2001b] can perform this task.

Direction: If we do not consider flow information in the first phase the subsequent phases may not be able to produce a drawing respecting the flow aesthetic criterion, therefore there may be edges in the drawing which do not point upward. All we can do then is to try to minimize the number of these edges. This is the approach taken in quasi-upward drawings as described in [Bertolazzi et al. 2002]. In quasi-upward drawings not all edges point upward, but upward directed edges emit from the upper half of the source node and connect to the lower half of the target node. In [Bertolazzi et al. 2002] a topology-shape-metrics approach for quasi-upward drawings with a minimal number of non-upward pointing edges is described. If we want to avoid non-upward pointing edges, the result of the first phase must meet some special requirements. In [Eiglsperger and Kaufmann 2001a; Eiglsperger and Kaufmann 2002] a heuristics for the first phase is presented which produces a *mixed upward embedding* of an input graph. Mixed upward embeddings have the property that there is a drawing of the input graph which is respecting the flow aesthetic criterion. An appropriate orthogonalization step can take this mixed upward embedding and calculate angles as well as bends such that all hierarchic edges point upward. An algorithm based on integer linear programming is presented in [Eiglsperger et al. 2000].

Labeling: Usually graph drawing systems perform labeling as a separate task after the drawings have been calculated and use a separate map labeling technique, i.e., [Wagner and Wolff 1998]. Labeling can be integrated into the compaction step which usually yields better results than using a map labeling algorithm. Label placement using integer linear programming has been described in [Klau and Mutzel 1999; Binucci et al. 2002], a heuristics for edge label placement has been presented in [Binucci et al. 2001].

Clustering: In all three steps clustering has to be considered. The concept of *c-planarity* [Feng et al. 1995] extends planarity to clustered graphs. A c-planarization algorithm together with a sketch for an orthogonalization algorithm is presented in [Battista et al. 2001]. In [Lütke-Hüttemann 1999] details for an orthogonalization algorithm can be found.

4.3 Analysis of the State of the Art

If we want to use the techniques listed above to integrate them into one algorithm for the automatic layout of class diagrams we face some problems. The first problem is that some solutions to problems are conflicting and are not compatible with each other. The second problem is that the performance of some of the above algorithms is not satisfactory. Since the automatic layout algorithm is usually used in a graphical user interface, it is essential that the running time supports interactivity. And finally not all requirements for automatic layout algorithms of class diagrams are covered by the above publications. We will now discuss these problems in detail.

The solutions proposed for handling clustering [Battista et al. 2001] and direction [Eiglsperger and Kaufmann 2002] are not compatible. The reason for this is that both approaches extend planarity in different directions, c-planarity for clustering and mixed upward planarity for direction. We can define *mixed upward c-planarity* for a clustered graph by merging the two definitions but currently there are no algorithms which can construct a mixed upward c-planarization. Therefore currently the user can either focus on hierarchy or clustering but not on both. We believe that deriving an algorithm for mixed upward c-planarization from the algorithm for mixed upward planarization is not too difficult and the challenges are rather technical ones.

Some of the above methods are based on integer linear programming (ILP), a very powerful combinatorial optimization method. ILP has been applied successfully to a number of graph drawing problems. Since with ILP usually optimal solutions are produced it is often used to evaluate heuristics, e.g., see [Klau et al. 2001]. The drawback of integer linear programming is that it is NP-complete and therefore all known algorithms have exponential worst case running time which is not suitable for interactive applications. It requires furthermore sophisticated solvers whose restrictions, e.g., supported platforms, licensing fees, etc., are prohibitive for our setting. The only orthogonalization algorithm which can handle aesthetic criterion FLOW is based on ILP.

Issues for class diagrams not covered yet by publications are the special aesthetic criteria for class diagrams: HYPEREDGE, CENTER and PROXIMITY.

5 Our Algorithm

In this section we present our new algorithm for the automatic layout of class diagrams. An alternative concept for the automatic layout of class diagrams based on the topology-shape-metrics approach is presented in [Gutwenger et al. 2002], but no details of the algorithm are revealed.

We first define the input of the algorithm and then present the complete algorithm. The algorithm follows a description of the parts of it which are not covered by previous work. Finally we analyze our algorithm and compare it to the existing algorithms.

5.1 Interface to the Application

Our algorithm assumes as input a *class diagram graph* which consists of a graph $G = (V,E)$, a set of labels L, a function $S : V \cup L \to \mathbb{N}^2$ denoting the size of the nodes, resp. labels, in the drawing. To each node, edge, or label a type is assigned denoting its semantics in the class diagram:

node types	`class, interface, misc`
edge types	`dependency, generalization, association, misc`
label types	`multiplicity, role, name, misc`

The type `misc` is used as generic type for objects for which none of the other types applies. In a transformation step the class diagram graph is transformed into a suitable input for the layout algorithm. The input for the layout algorithm is:

- A graph $G = (V,E)$,
- a subset $D \subseteq E$ denoting the directed edges,
- a subset $H \subseteq E$ denoting the edges which are considered to be part of an hyperedge,
- a set of labels L together with a function $T : L \to \{\texttt{source}, \texttt{center}, \texttt{target}\}$ denoting the preferred position of the label along the edge,
- a function $S : V \cup L \to \mathbb{N}^2$ denoting the size of the nodes, resp. labels, in the drawing.

The transformation is parametrized to give the user the choice between different styles. One possible style is to direct all generalization edges and draw them as hyperedges. Another possibility is to direct the associations and to keep the generalizations undirected. The algorithm distinguishes between interfaces and classes. Class and interface inheritance can be given different styles. This is especially useful for class diagrams for Java programs. In the transformation step to each label a preferred placement is assigned. Labels of type `multiplicity` and `role` are always assigned to one end of an association. They receive therefore either the preferred placement `source` or `target`. All other labels receive the preferred placement `center`.

The types and the transformation rules mentioned above should be rather seen as an example than as a definition. Through the use of the intermediate transformation step we make it easy to integrate further model elements by enhancing the type table and the transformation. The transformation step itself influences the main aesthetic criterion for the diagram. If no edges are directed, and no hyperedges are defined, we try to minimize the number crossings. Otherwise the crossings minimization plays only a secondary role behind the flow aesthetic criterion, and/or the hyperedge criterion.

5.2 The Complete Algorithm

We present now our automatic layout algorithm for class diagrams. The heart of our algorithm is a new orthogonalization method, which considers the FLOW and CENTER aesthetic criteria.

We assume that the directed graph induced by the predicate H is acyclic. This assumption is justified, since the generalization relationship between classes is acyclic by definition and the hyperedge notation is restricted to generalization edges in the UML [OMG 2001]. If the directed graph induced by D and H contains cycles we use a similar strategy like the hierarchical graph drawing algorithm: we draw only a subset of the edges, which induces an acyclic subgraph, upward. We need only to remove edges from D, since the directed graph induced by H is acyclic.

1. **Preprocessing**
   (a) In a first preprocessing step the graph is divided into its connected components. Each connected component is processed separately by our algorithm. Therefore we assume that in the remainder - except for the last step - that the input graph is connected.
   (b) Remove edges from D until the edges in $D \cup H$ induce an acyclic subgraph of G.
   (c) In a further preprocessing step the edges in H are substituted by stars, see 5.3 for details.
   (d) If the edges in D do not induce a connected subgraph some edges are added temporarily to D to make this subgraph connected by using a minimum spanning tree algorithm.
2. **Planarization**
   The algorithm described in [Eiglsperger and Kaufmann 2001a] is used to calculate a mixed upward planarization of the input graph. All edges in $D \cup H$ are considered as directed. This algorithm is able to consider edge weights, we assign the highest weight to edges in H , followed by edges in D.
3. **Orthogonalization**
   The orthogonalization uses the algorithm described in 5.5.
4. **Compaction**
   (a) Edge labels with preferred placement `center` are introduced into the drawing by subdividing edges as described in 5.4.
   (b) The algorithm described in [Eiglsperger and Kaufmann 2001b] is used to calculate an initial drawing.

(c) Visibility post-processing known from VLSI layout, see e.g., [Lengauer 1990], is used to reduce area and edge lengths of the drawing. Either path based or flow based visibility postprocessing is used. Empirical tests on 4-graphs [Klau et al. 2001] have shown that using a heuristics for compaction together with flow based visibility post-processing yield nearly optimal solutions.

5. **Postprocessing**

(a) All dummy nodes are removed from the graph including crossings, label nodes, hyperedge nodes, and artificial bends.

(b) Edge labels with preferred placement `source` or `target` are placed by a map labeling algorithm.

(c) The layouted connected components of the graph are arranged by a floor planning algorithm, see e.g. [Freiwalds et al. 2001] for an overview of algorithms.

5.3 Hyperedge Substitution

Hyperedges are substituted by stars. All incoming edges of a node u are removed from u and connected to a *hyperedge node* d, which itself connects to u. Let $u \in V$ and $H(u) = \bigcup_{e=(v,u)\in E} e \in H$. For $H(u) = \{(v_1,u),\ldots,(v_k,u)\}$, we create a new node d, we substitute the edges (v_i,u) with (v_i,d), $1 \le i \le k$, and add the edge (d,u). See figure 4 for an example.

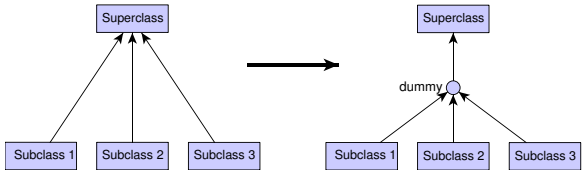


Figure 4: Substitution of edges forming a hyperedge by a star.

5.4 Label Placement

Edge labels with preferred placement `source` or `target` are ignored and placed by a map labeling algorithm after the layout. This approach works for class diagrams since these labels are fairly small, i.e., multiplicity labels have the form "1..n" or "$*$". Edge labels with preferred placement `center` are handled differently. Before the compaction phase, the edge, to which the label is attached, is a path of segments. We choose a segment in the middle and split it in two parts by introducing a dummy node. The dummy node has the size of the label. Then after the compaction the label is placed on the position of the dummy node and the dummy node is removed from the graph. See figure 5 for an example.

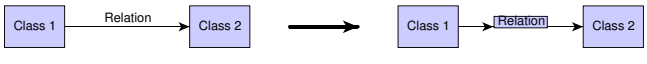


Figure 5: Labels with preferred placement `center` are treated like nodes in the compaction.

5.5 Orthogonalization

As already stated in Sec. 4, the orthogonalization algorithm calculates the angles and bends in the drawing. The input for the algorithm is in our case a mixed upward planarization of the class diagram graph. We assume that the dummy vertices in the planarization representing crossings are marked accordingly. We assume furthermore that the subgraph induced by the directed edges is connected, which is ensured by step 1d.

Traditionally orthogonalization algorithms try to minimize the number of bends in the drawing, and therefore optimize aesthetic criterion BEND. The orthogonalization, which we present in this section, tries to optimize the FLOW and CENTER aesthetic criteria in the first place, the aesthetic criterion BEND is only the second objective. We achieve this by performing the orthogonalization in two phases. In the first phase we calculate the orthogonalization for directed edges and hyperedges. In the second phase the remaining edges are orthogonalized. We will now discuss these two phases.

As already stated, the first phase works on the subgraph induced by the directed and hyperedges. The output is a string over the alphabet $\{\leftarrow,\uparrow,\rightarrow\}$ for each edge in the subgraph. This string denotes the segments of the edge and their orientation. We call this string the *shape* of the edge. The shapes are assigned in a way that all incoming edges connect to the same side of the node and all outgoing edges connect to the same side. The sides for the incoming edges and outgoing edges are opposite to each other.

The algorithm works as follows: For each edge a tail- and a headshape is calculated. The two shapes are concatenated yielding the shape for the edge. The algorithm picks one node at a time and assigns the headshape to the incoming edges of the node and the tailshape to the outgoing edges. According to the bend-or-end property of the `Kandinsky` model only one incoming edge and one outgoing edge may not bend. We choose this non-bending edge according to the following rules: If there are edges which are part of a hyperedge, choose the median edge according to the embedding of these edges. If there are no edges which are part of a hyperedge, choose the median edge according to the embedding if the number of edges is even, choose no non-bending edge if the number is odd. The assignment of the shapes is illustrated in Fig. 6. The tail- and headshapes can always be concatenated since the assignment of the shapes has the invariant that each tailshape ends with a $\uparrow$ and each headshape starts with a $\uparrow$. For the assignment of shapes at crossings we provide two alternative configurations which are symmetric. In both cases one edge bends while the other remains straight. We choose the configuration such that the edge with the lower weight bends and the edge with higher weight does not. Since edges in H have higher weight than edges in D, hyperedges do only bend at edge crossings if the other part is also an hyperedge. The algorithm is summarized in Algorithm 1, *Shape Calculation*.

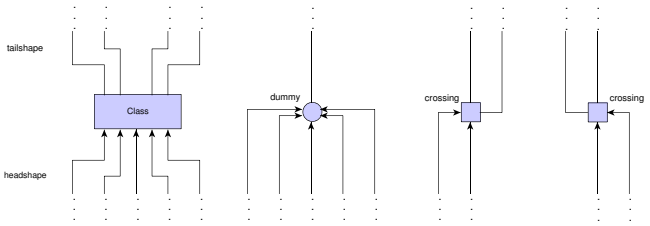


Figure 6: Shapes assigned to directed edges and hyperedges.

Theorem 1 *Algorithm* Shape Calculation *is correct, i.e., there is a drawing of the directed subgraph G' induced by the edges $D \cup H$, in which the paths of the edges correspond to the calculated shape.*

Input: Upward planarization $G = (V,E)$
Output: A function $shape : E \longrightarrow \{\leftarrow, \uparrow, \rightarrow\}^*$
// Tailshape
for $v \in V$ **do**
  $l = \{e \in E | e = (v,w), e \in D \cup H\}$
  $m = median(l)$
  $shape(l_m) = " \uparrow "$
  if *v is crossing* **then**
    if $m = 0$ **then** $shape(l_1) = " \rightarrow \uparrow "$
    else $shape(l_0) = " \leftarrow \uparrow "$
  else
    for $0 \leq i < m$ **do** $shape(l_i) = " \uparrow \leftarrow \uparrow "$
    for $m < i < |l|$ **do** $shape(l_i) = " \uparrow \rightarrow \uparrow "$
  end
end
// Headshape
for $v \in V$ **do**
  $l = \{e \in E | e = (w,v), e \in D \cup H\}$
  $m = median(l)$
  if *v is crossing* **then**
    if $m = 0$ **then**
      $shape(l_1) += " \leftarrow "$
    end
    else $shape(l_0) += " \rightarrow "$
  else
    if *v is hyperedge node* **then**
      for $0 \leq i < m$ **do** $shape(l_i) += " \rightarrow "$
      for $m < i < |l|$ **do** $shape(l_i) += " \leftarrow "$
    else
      for $0 \leq i < m$ **do** $shape(l_i) += " \rightarrow \uparrow "$
      for $m < i < |edges|$ **do** $shape(l_i) += " \leftarrow \uparrow "$
    end
  end
end

Algorithm 1: *Shape Calculation*. In the algorithm l denotes a list of edges adjacent to a vertex v. The edges are ordered in l from left to right according to the mixed upward embedding. We denote with $|l|$ the size of the list and with l_i the i^{th} element in the list. The concatenation of two stings is denoted by $+=$.

Proof: By definition of mixed upward planarization, G' is upward planar. For each upward planar graph there is an upward drawing where nodes are representated as horizontal straight lines and edges as vertical straight lines, called visibility representation, see, e.g., [Di Battista et al. 1999]. Replacing the horizontal straight line in the visibility representation by the configurations shown in Fig. 6 yields a drawing of G' with the correct shapes. □

After this assignment we perform *bend stretching* transformations to reduce the number of bends. A bend stretching transformation removes superfluous bends from an edge by assigning a new shape with less bends to it without changing the first and last direction in the shape. Table 1 shows all bend-stretching transformations. Figure 7 shows how the two bend stretching transformations 2 and 3 on the edge $(C1, C7)$ reduce the number of bends in the graph by 4.

The above algorithm runs in linear time with respect to the size of the planarization and the result has the following properties:

- on class-nodes the directed edges leave the source node at the top and enter the target node on the bottom,
- aesthetic criterion FLOW is satisfied,
- the number of bends per edge is at most $2c+4$ where c is the number of crossings of the edge.

shape	v is crossing	w is crossing	new shape
$\uparrow\rightarrow\uparrow\rightarrow\uparrow$			$\uparrow\rightarrow\uparrow$
$\uparrow\rightarrow\uparrow\rightarrow$		X	$\uparrow\rightarrow$
$\rightarrow\uparrow\rightarrow\uparrow$	X		$\rightarrow\uparrow$
$\rightarrow\uparrow\rightarrow$	X	X	$\rightarrow$
$\uparrow\leftarrow\uparrow\leftarrow\uparrow$			$\uparrow\leftarrow\uparrow$
$\uparrow\leftarrow\uparrow\leftarrow$		X	$\uparrow\leftarrow$
$\leftarrow\uparrow\leftarrow\uparrow$	X		$\leftarrow\uparrow$
$\leftarrow\uparrow\leftarrow$	X	X	$\leftarrow$

Table 1: Table of bend-stretching transformations that apply to an edge $e = (v,w)$.

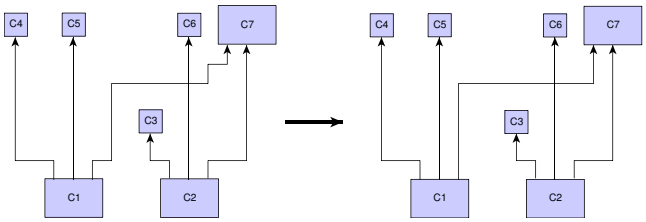


Figure 7: Example for a bend stretching transformation.

The first property follows directly from the assignment of the shapes. The same holds for aesthetic criterion FLOW since the edges are only assigned upward pointing shapes. Because we choose the non-bending edge to be the median, the superclass is centered in a topological sense: There is the same number of subclasses on the left side of the superclass as on the right side.

The second phase calculates exclusively the values for the undirected edges. It takes the shapes of the directed edges and edges which are part of an hyperedge as input and transform the shapes into bends and angles. The bends and angles defined by this transformation are not allowed to change. We use the network flow algorithm [Fößmeier and Kaufmann 1996] with modifications similar to those in [Brandes et al. 2002] to calculate the shape for the remaining edges while guaranteeing that the prescribed angles and bends are conserved. The algorithm has running time $O(n^2 \log n)$, where n is the number of vertices in the planarization, and minimizes the number of bends of the undirected edges. Note that the running time does not depend on the number of directed edges and edges which are part of an hyperedge.

5.6 Discussion

In this section we discuss the properties of the topology-shape-metrics algorithm and compare it to the properties of existing algorithms.

The main aesthetic criterion for the topology-shape-metrics algorithm is the number of crossings CROSSING, the second most important one is the number of bends BEND. The aesthetic criteria total edge length EDGE LENGTH and area AREA are secondary optimization goals. If wanted, the user can enforce the flow aesthetic criterion FLOW and the hyperedge aesthetic criterion HYPEREDGE for subsets of the edges. The algorithm generates no overlaps OVERLAP, all edges are orthogonal paths ORTHOGONAL, the angular resolution RESOLUTION is 90^o for this reason. The aesthetic criteria symmetry SYMMETRY and aspect ratio ASPECT RATIO are not considered explicitly. The same holds for the centering of n-ary association nodes CENTER, handling of comment nodes and association classes PROXIMITY. In [Eichelberger 2002] clustering is used to ensure these aesthetic criteria. As

soon as clustering support is integrated in our algorithm, we can use the same technique. However, choosing high weights for the edges involved in these aesthetic criteria might work for our approach equally well. Since comment nodes are usually 1-degree nodes, they are placed automatically near the opposite node by our algorithm. For this reason we believe that these aesthetic criteria are supported implicitly by our algorithm, and we do not need to support them explicitly.

The running time is dominated by the mixed upward planarization step and the solution of the network flow problem in the orthogonalization step. If n is the number of nodes in the graph, m the number of edges and c the number of crossings calculated in the planarization step, then the planarization has time complexity $O(nm^2 + (n+c)^2 m)$ and the solution of the network flow problem $O((n+c)^2 \log(n+c))$.

Compared to the algorithms based on the hierarchic approach, the topology-shape-metrics algorithm offers a greater flexibility concerning the aesthetic criteria since the hierarchic approach is built solely upon the flow aesthetic criterion. We have seen in section 2 that concentrating on the flow aesthetic criterion is at least questionable. The hierarchic approach has advantages if the number of crossings in the drawing is high or if the hierarchy of edges for which the flow aesthetic criterion applies is well formed, i.e., it has a sufficient depth and most nodes are contained in it. Currently the hierarchic approach has the advantage that it supports clustering. However, as noted in section 4, there are approaches for handling clustering in the topology-shape-metrics approach and they will be incorporated in future versions of our algorithm.

6 Implementation

We implemented the above algorithm in JAVA based on the yFiles library [Wiese et al. 2001]. We integrated it in the demo reverse engineering tool `JarInspector` which is available at: `http://www-pr.informatik.uni-tuebingen.de/c/forschung/uml/jarinspector.xml`. The program is not a full-featured reverse engineering tool, its aim is to show how automatic layout can be used in the scenario of reverse engineering, The program takes Java byte-code as input, either from a Java archive (jar) or from class files stored in a directory hierarchy, and generates an UML model from it, which contains the following information:

- Each class or interface defined in the byte-code,
- all generalization relationship between classes/interfaces,
- all package dependencies, and
- associations which are defined by a field of a class.

The content of the generated UML model can be browsed interactively by the user. For each selected view of the model, a diagram is generated automatically and layouted by our algorithm.

7 Conclusions

We reviewed in this work aesthetic criteria for UML class diagrams and state of the art of automatic layout algorithms for class diagrams. We showed that these algorithms, which are based on the hierarchical approach, have major drawbacks. We presented a new automatic layout algorithm for class diagrams based on the topology-shape-metrics approach which overcomes these drawbacks. We implemented the algorithm in JAVA and provide some example outputs of our algorithm. The implementation demonstrates the effectivity and efficiency of the algorithm in practice.

Directions of our future research are:

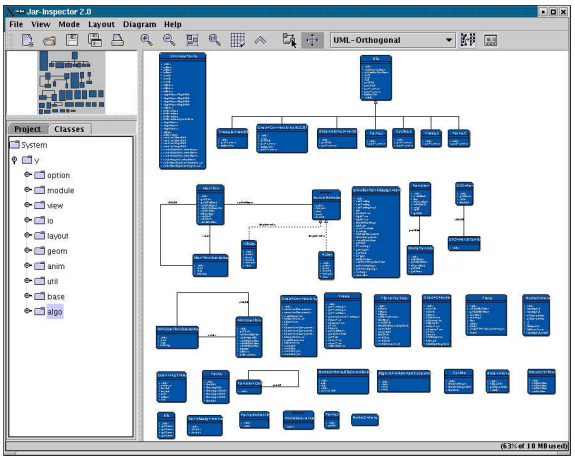


Figure 8: Screenshot of the `JarInspector` application.

- Merge the concepts of c-planarity and upward planarity,
- extend the work to other types of UML diagrams,

References

BATTISTA, G. D., 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, J. Kratochvil, Ed., vol. 1731 of *LNCS*, 297–310.

BATTISTA, G. D., DIDIMO, W., AND MARCANDALLI, A. 2001. Planarization of clustered graphs. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'2001)*, vol. 2275 of *LNCS*, 60–74.

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.

BERTOLAZZI, P., BATTISTA, G. D., AND DIDIMO, W. 2002. Quasi-upward planarity. *Algorithmica 32*, 3, 474–506.

BINUCCI, C., DIDIMO, W., LIOTTA, G., AND NONATO, M. 2001. Labeling heuristics for orthogonal drawings. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'2001)*, vol. 2275 of *LNCS*, 139–153.

BINUCCI, C., DIDIMO, W., LIOTTA, G., AND NONATO, M. 2002. Computing labeled orthogonal drawings. In *Proceedings of the 10th International Symposium on Graph Drawing (GD'2002)*, vol. 2528 of *LNCS*, 66–73.

BOOCH, G., RUMBAUGH, J., AND JACOBSON, I. 1999. *The Unified Modeling Language User Guide*. Addison-Wesley.

BOOCH, G. 1994. *Object Oriented Design and Analysis*. Addison-Wesley.

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'2002)*, Springer, vol. 2528, 1–12.

CHIDAMBER, S., AND KEMERER, C. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering 20*, 6, 467–493.

COFFMAN, E. G., AND GRAHAM, R. L. 1972. Optimal scheduling for two processor systems. *Acta Informatica 1*, 200–213.

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

EICHELBERGER, H. 2000. Automatisches Zeichnen von UML Klassendiagrammen mit dem Sugiyama-Algorithmus in Tagungsband des GI-Workshops Softwarevisualisierung 2000. Tech. Rep. A/01/2000, Universität des Saarlandes.

EICHELBERGER, H. 2002. Aesthetics of class diagrams. In *Proc. of the First IEEE International Workshop on Visualizing Software for Understanding and Analysis, Vissoft 2002*, 23–31.

EIGLSPERGER, M., AND KAUFMANN, M. 2001. An approach for mixed upward planarization. In *Proceedings of the 7th International Workshop on Algorithms and Datastructures (WADS 2001)*, Springer, 352–364.

EIGLSPERGER, M., AND KAUFMANN, M. 2001. Fast compaction for orthogonal drawings with vertices of prescribed size. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'2001)*, Springer, vol. 2265, 124–138.

EIGLSPERGER, M., AND KAUFMANN, M. 2002. An approach for mixed upward planarization. *Journal of Graph Algorithms and Applications*. submitted.

EIGLSPERGER, M., FOESSMEIER, U., AND .KAUFMANN, M. 2000. Orthogonal graph drawing with constraints. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, 3–11.

FENG, Q.-W., COHEN, R. F., AND EADES, P. 1995. Planarity for clustered graphs. In *Proceedings of the 3rd European Symposium on Algorithms (ESA'95)*, Springer, vol. 979 of *LNCS*, 213–226.

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.

FREIWALDS, K., DOGRUSOZ, U., AND KIKUSTS, P. 2001. Disconnected graph layout and the polyomino packing approach. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'2001)*, Springer, vol. 2265 of *LNCS*, 378–391.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns*. Addison-Wesley.

GUTWENGER, C., JÜNGER, M., KLEIN, K., KUPKE, J., LEIPERT, S., AND MUTZEL, P. 2002. Caesar automatic layout of uml class diagrams. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'2001)*, Springer, vol. 2265 of *LNCS*.

KLAU, G. W., AND MUTZEL, P. 1999. Combining graph labeling and compaction. In *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, Springer, J. Kratochvil, Ed., no. 1731 in LNCS, 27–37.

KLAU, G. W., KLEIN, K., AND MUTZEL, P. 2001. An experimental comparison of orthogonal compaction algorithms. In *Proceedings of the 8th International Symposium on Graph Drawing (GD'2000)*, no. 1984 in LNCS, 37–51.

LENGAUER, T. 1990. *Combinatorial Algorithms for Integrated Circuit Layout*. Applicable Theory in Computer Science. Wiley-Teubner.

LÜTKE-HÜTTEMANN, D. 1999. *Knickminimales Zeichnen 4-planarer Clustergraphen*. Master's thesis, Universität des Saarlands.

OMG, 2001. Unified Modeling Language v1.4. `http://www.omg.org/technology/documents/formal/uml.htm`.

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., ALLDER, J. A., AND CARRINGTON, D. 2000. User preference of graph layout aesthetics: A uml study. In *Proceedings of the 8th International Symposium on Graph Drawing (GD'2000)*, Springer, vol. 1984 of *LNCS*, 5–18.

PURCHASE, H. C., MCGILL, M., COLPOYS, L., AND CARRINGTON, D. 2001. Graph drawing aesthetics and the comprehension of uml class diagrams: an empirical study". In *Proceedings of the Australian Symposium on Information Visualization*, Australian Computer Society Inc., vol. 9.

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)*, vol. 1353 of *LNCS*, 248–261.

SEEMANN, J. 1997. Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. In *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, vol. 1353 of *LNCS*, 415–424.

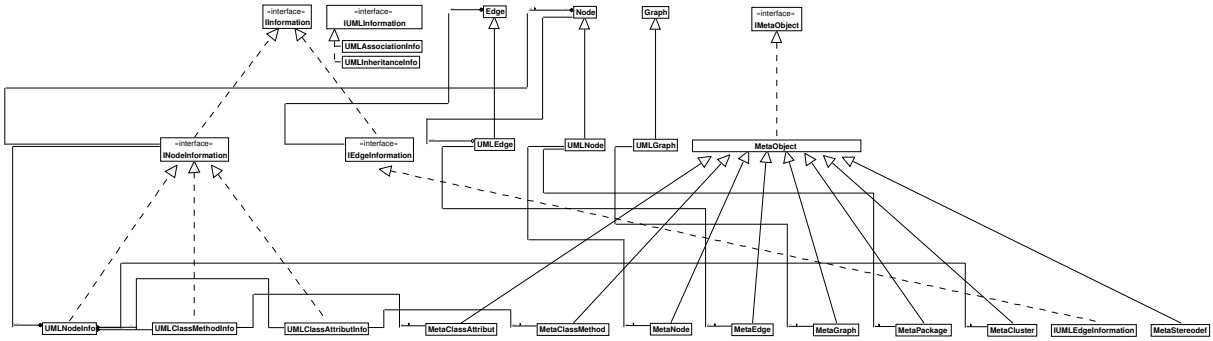
SUGIYAMA, K., TAGAWA, S., AND TODA, M. 1981. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics 11*, 2 (February), 109–125.

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

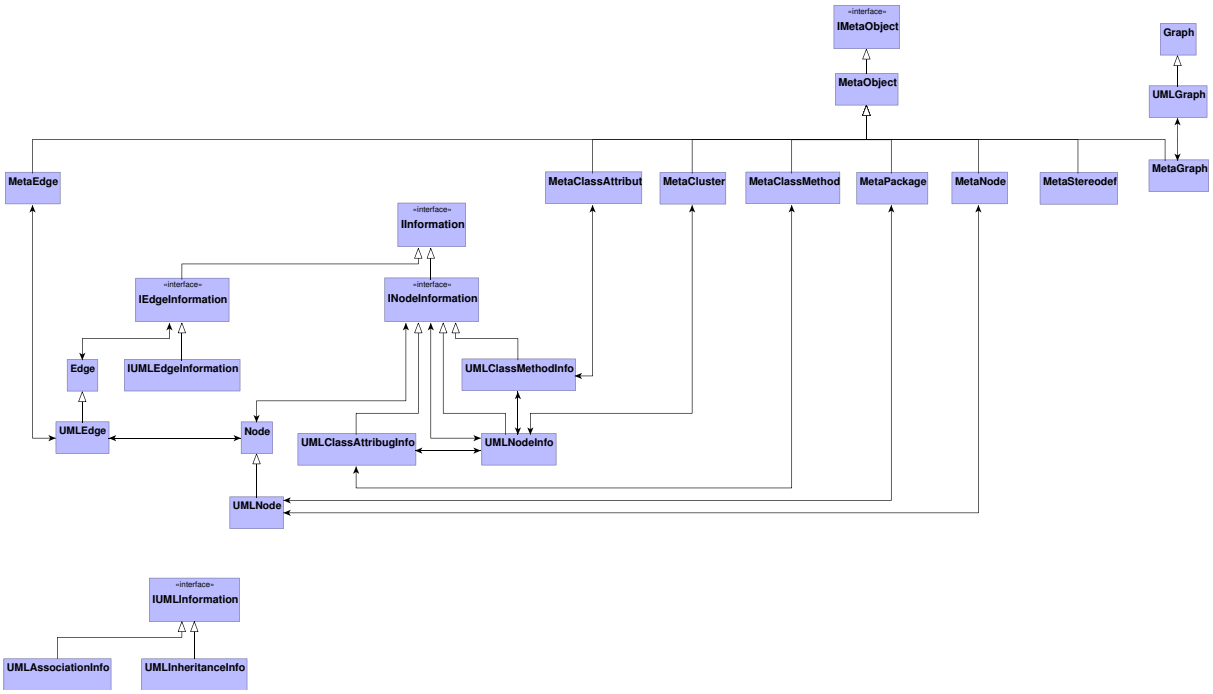
WAGNER, F., AND WOLFF, A. 1998. A combinatorial framework for map labeling. In *Proceedings of the 6th International Symposium on Graph Drawing (GD'98)*, Springer, no. 1547 in LNCS, 316–331.

WIESE, R., EIGLSPERGER, M., AND KAUFMANN, M. 2001. yfiles: Visualization and automatic layout of graphs. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, Springer, LNCS, 453–454.

A Example



(a)



(b)

Figure 9: Example UML class diagram from [Eichelberger 2000] drawn with the hierarchical approach (a) and with the topology-shape-metrics approach (b).

A New Approach for Visualizing UML Class Diagrams

Carsten Gutwenger
Research Center caesar

Michael Jünger
University of Cologne

Karsten Klein
Research Center caesar

Joachim Kupke
Research Center caesar

Sebastian Leipert
Research Center caesar

Petra Mutzel
Vienna University of Technology

This page shows three different UML class diagrams that have been automatically laid out using the GoVisual layout algorithm for UML class diagrams. Each layout has been computed within less than a second on a standard PC. The GoVisual Software is available as API and as plug-in for Borland® Together® ControlCenter™, Gentleware Poseidon for UML, and Microsoft® Visio® 2000/2002.