Visualization of graph structures

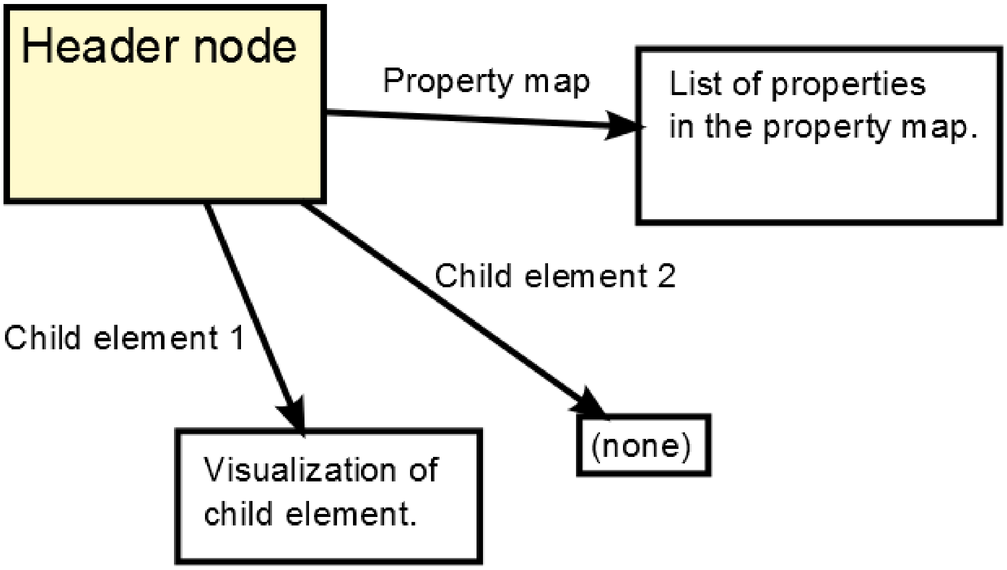
To demonstrate the possibilities of this framework, we represent the results of specialized transformations for three graph structures.

General structure

All visualizations follow the same general structure:

- A header node (colored in lemon) representing the currently selected element.
- A visualiation of the properMap in tabular form.
- A node for each child element. (e.g. visualisation of a graph, the bendPoints of a edge...)

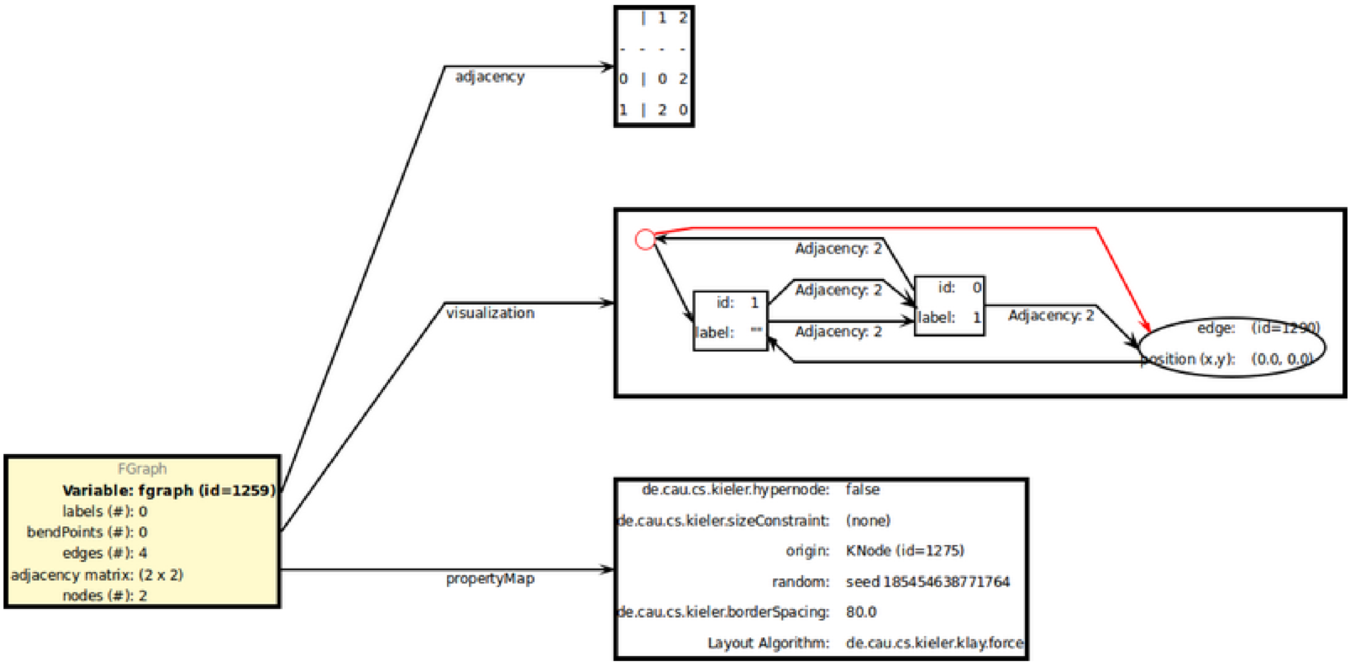
These main nodes all have a linewidth of 4 and are connected from the node by a directed and labeled edge. If one of the nodes is empty (e.g. there are no bendPoints on the selected edge), the node is still rendered and filled with the text "(empty)". Elements in these main nodes normally have a linewidth of 2.



Visualization of FGraph

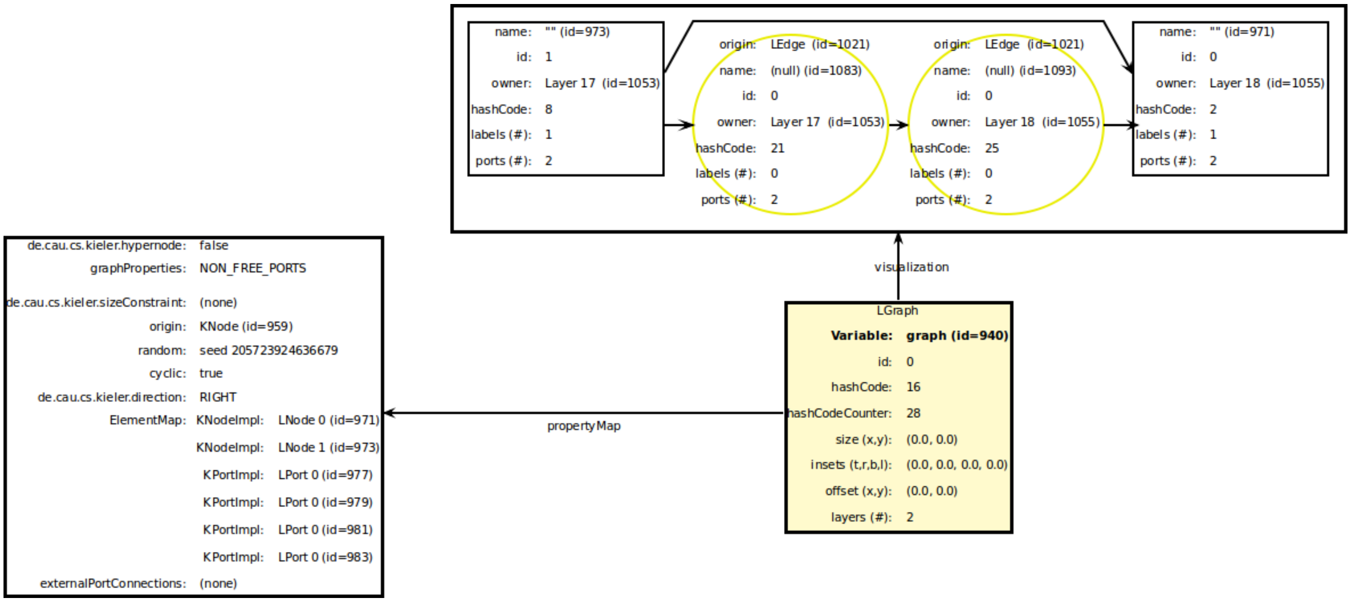
There are visualizations for

- FGraph
- FNode
- FEdge
- FBendpoints
- FLabel


| 1 2
- - - -
0 | 0 2
1 | 2 0
adjacency
visualization
propertyMap
Adjacency: 2
Adjacency: 2
Adjacency: 2
Adjacency: 2
id: 1
label: ""
id: 0
label: 1
edge: (id=1290)
position (x,y): (0.0, 0.0)
FGraph
Variable: fgraph (id=1259)
labels (#): 0
bendPoints (#): 0
edges (#): 4
adjacency matrix: (2 x 2)
nodes (#): 2
de.cau.cs.kieler.hypernode: false
de.cau.cs.kieler.sizeConstraint: (none)
origin: KNode (id=1275)
random: seed 185454638771764
de.cau.cs.kieler.borderSpacing: 80.0
Layout Algorithm: de.cau.cs.kieler.klay.force

Visualization of LGraph

- LGraph
- LLayer
- LNode
- LEdge
- LPorts
- LLabel



```
    name:  "" (id=973)
      id:  1
   owner:  Layer 17  (id=1053)
hashCode:  8
labels (#):  1
 ports (#):  2
```

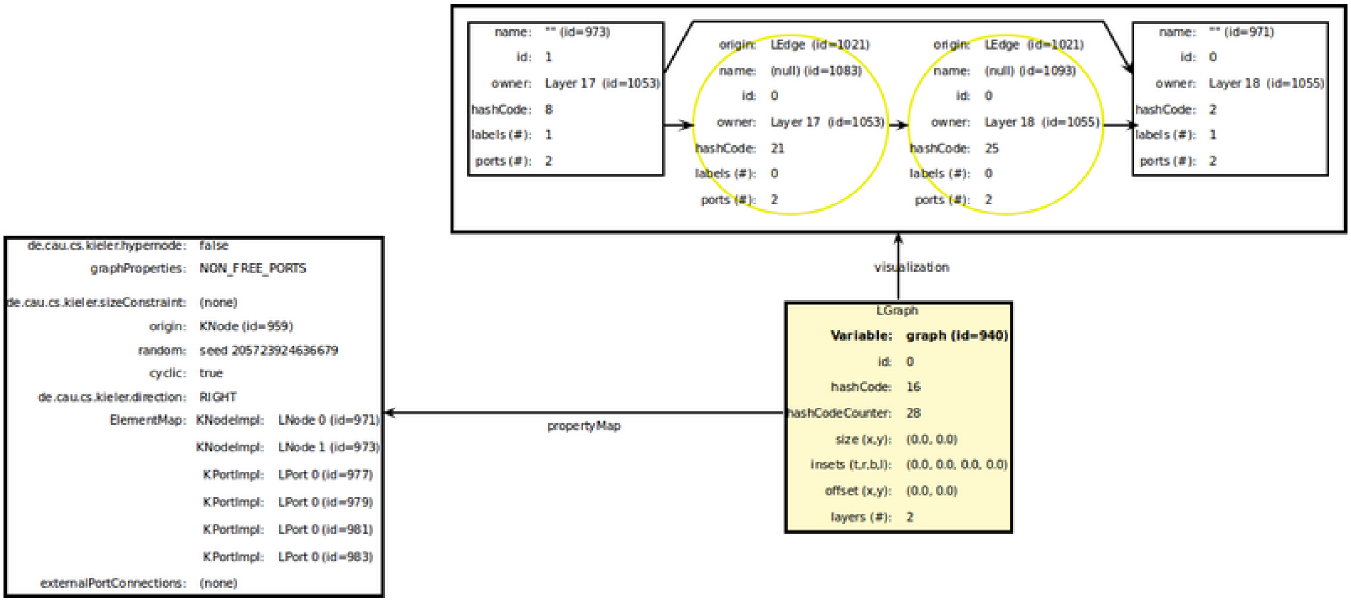
```
  origin:  LEdge  (id=1021)
    name:  (null) (id=1083)
      id:  0
   owner:  Layer 17  (id=1053)
hashCode:  21
labels (#):  0
 ports (#):  2
```

```
  origin:  LEdge  (id=1021)
    name:  (null) (id=1093)
      id:  0
   owner:  Layer 18  (id=1055)
hashCode:  25
labels (#):  0
 ports (#):  2
```

```
    name:  "" (id=971)
      id:  0
   owner:  Layer 18  (id=1055)
hashCode:  2
labels (#):  1
 ports (#):  2
```

```
         Variable:  graph (id=940)
               id:  0
         hashCode:  16
  hashCodeCounter:  28
       size (x,y):  (0.0, 0.0)
 insets (t,r,b,l):  (0.0, 0.0, 0.0, 0.0)
     offset (x,y):  (0.0, 0.0)
       layers (#):  2
```

```
     de.cau.cs.kieler.hypernode:  false
                graphProperties:  NON_FREE_PORTS
de.cau.cs.kieler.sizeConstraint:  (none)
                         origin:  KNode (id=959)
                         random:  seed 205723924636679
                         cyclic:  true
     de.cau.cs.kieler.direction:  RIGHT
                     ElementMap:  KNodeImpl:   LNode 0 (id=971)
                                  KNodeImpl:   LNode 1 (id=973)
                                  KPortImpl:   LPort 0 (id=977)
                                  KPortImpl:   LPort 0 (id=979)
                                  KPortImpl:   LPort 0 (id=981)
                                  KPortImpl:   LPort 0 (id=983)
        externalPortConnections:  (none)
```


visualization
propertyMap
LGraph

```
    name:  "" (id=973)
      id:  1
   owner:  Layer 17  (id=1053)
hashCode:  8
labels (#):  1
 ports (#):  2
```

```
  origin:  LEdge  (id=1021)
    name:  (null) (id=1083)
      id:  0
   owner:  Layer 17  (id=1053)
hashCode:  21
labels (#):  0
 ports (#):  2
```

```
  origin:  LEdge  (id=1021)
    name:  (null) (id=1093)
      id:  0
   owner:  Layer 18  (id=1055)
hashCode:  25
labels (#):  0
 ports (#):  2
```

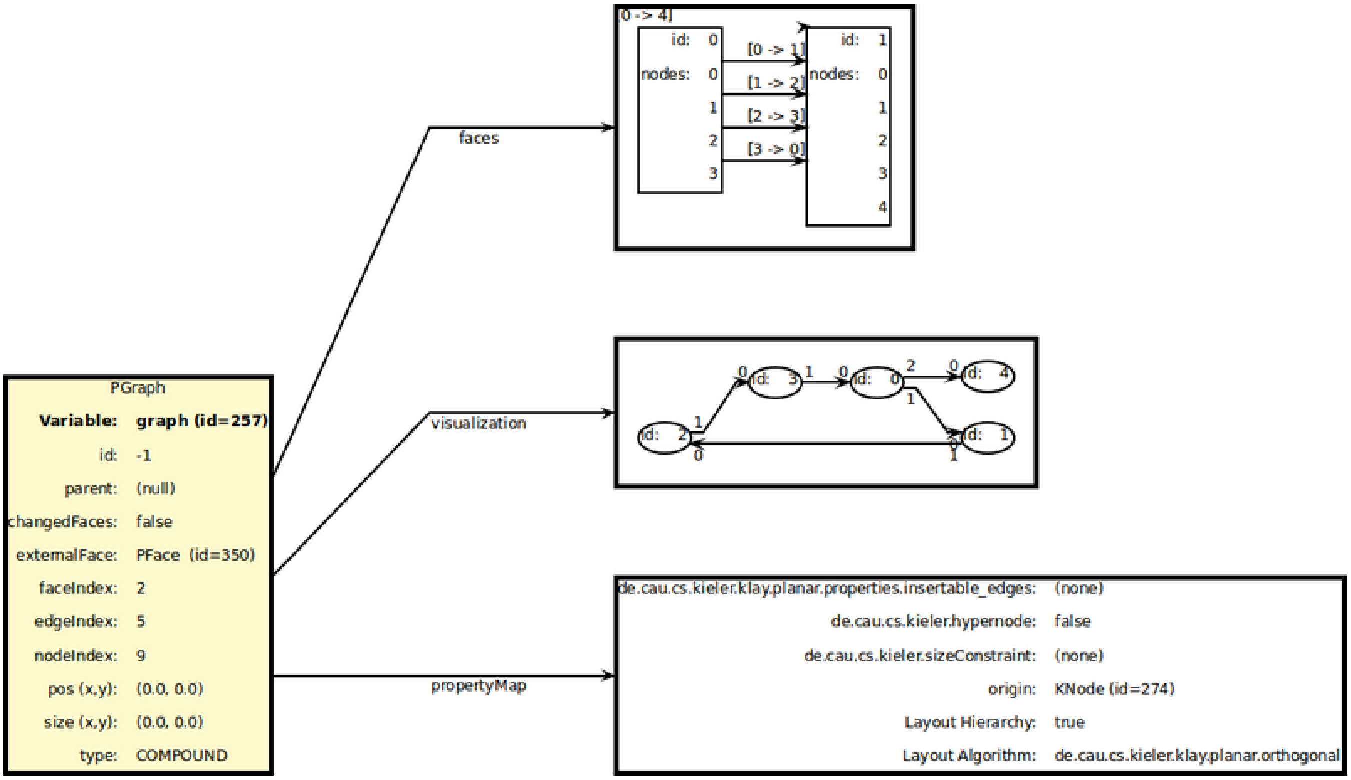
```
    name:  "" (id=971)
      id:  0
   owner:  Layer 18  (id=1055)
hashCode:  2
labels (#):  1
 ports (#):  2
```

```
         Variable:  graph (id=940)
               id:  0
         hashCode:  16
  hashCodeCounter:  28
       size (x,y):  (0.0, 0.0)
 insets (t,r,b,l):  (0.0, 0.0, 0.0, 0.0)
     offset (x,y):  (0.0, 0.0)
       layers (#):  2
```

```
     de.cau.cs.kieler.hypernode:  false
                graphProperties:  NON_FREE_PORTS
de.cau.cs.kieler.sizeConstraint:  (none)
                         origin:  KNode (id=959)
                         random:  seed 205723924636679
                         cyclic:  true
     de.cau.cs.kieler.direction:  RIGHT
                     ElementMap:  KNodeImpl:   LNode 0 (id=971)
                                  KNodeImpl:   LNode 1 (id=973)
                                  KPortImpl:   LPort 0 (id=977)
                                  KPortImpl:   LPort 0 (id=979)
                                  KPortImpl:   LPort 0 (id=981)
                                  KPortImpl:   LPort 0 (id=983)
        externalPortConnections:  (none)
```

Visualization of PGraph

- PGraph
- PNode
- PEdge
- PFace
- PLabel



PGraph

Variable:	**graph (id=257)**
id:	-1
parent:	(null)
changedFaces:	false
externalFace:	PFace (id=350)
faceIndex:	2
edgeIndex:	5
nodeIndex:	9
pos (x,y):	(0.0, 0.0)
size (x,y):	(0.0, 0.0)
type:	COMPOUND

de.cau.cs.kieler.klay.planar.properties.insertable_edges:	(none)
de.cau.cs.kieler.hypernode:	false
de.cau.cs.kieler.sizeConstraint:	(none)
origin:	KNode (id=274)
Layout Hierarchy:	true
Layout Algorithm:	de.cau.cs.kieler.klay.planar.orthogonal

Class KTextIterableField.java

In the process of development a class for the creation of table layouts was generated. It provides the user with a very easy way to create table layouts. It is not necessary to specify the size of the table. The user simply adds elements to a arbitrary positions of the table. Also a table head can be added. This element will be centered above all other table elements. After fully filling the table, the table elements are added by iterating over the structure. Each element is returned as a KText.

Later we dropped the use of this class, as the usage of the GridLayout is recommended. As an example, the class is still used in following transformations:

- FEdgeTransformation
- FGraphTransformation
- FLabelTransformation