

# *Visualization of Business Process Models*

**DISSERTATION**

der Mathematisch-Naturwissenschaftlichen Fakultät  
der Eberhard Karls Universität Tübingen  
zur Erlangung des Grades des  
Doktors der Naturwissenschaften  
(Dr. rer. nat.)

vorgelegt von  
**Dipl.-Inform. Philip Josef Effinger**  
aus Reutlingen

Tübingen  
2012

---

Tag der mündlichen Qualifikation: 17.04.2013  
Dekan: Prof. Dr. Wolfgang Rosenstiel  
1. Berichterstatter: Prof. Dr. Michael Kaufmann  
2. Berichterstatter: Prof. Dr. Herbert Klaeren

---

© Copyright by Philip Effinger 2013.  
All Rights Reserved.





---

To my mother.



# Acknowledgments

This work is the result of a journey on which many people were on my side to accompany and support me and my work. First and foremost, I would like to thank my supervisor Prof. Dr. Michael Kaufmann who allowed me to be part of his group and who provided the means that were necessary for my work, be it inspirations or additional funding. The funding of my research was provided by Deutsche Forschungsgemeinschaft (DFG) under grant KA812/15-1. This enabled me to concentrate on research without concerning about existential matters. Also, I was able to participate on numerous international workshops and international conferences which was essential for the progress on my research. For the commitment and effort as a reviewer of this work, I am grateful to Prof. Dr. Herbert Klaeren.

The initial spark to visualization was given to me by Dr. Katharina Zweig when pointing out this field of research on the search for a topic of my diploma thesis in 2008. The introduction to the field and reams of hours of tutoring and discussions were graciously due to Dr. Martin Siebenhaller who did a great job as a forerunner in visualization and automatic layout in the work group.

Of course, the colleagues that had to deal with me on a daily basis earn special thanks for motivating me in the many hours of analyzing and solving algorithmic challenges of visualizations, in alphabetical order, they are: Till Bruckdorfer, Andreas Gerasch, Markus Geyer, Stephan Kottler, Robert Krug, Christian Zielke. Also, I enjoyed the short breaks where we also tackled non–algorithmic related issues.

Aside from this thesis, I was involved in numerous publications which also relied on the contributions of many coauthors, namely: Benjamin Albrecht, Christian Bachmaier, Franz–Josef Brandenburg, Gero Decker, Carsten Gutwenger, Markus Held, Nicole Jogsch, Jyrki Katajainen, Karsten Klein, Sascha Meinert, Sandra Seiz, Johannes Spielmann, Miro Spönemann, Matthias Stegmaier, Tamara Wehrstein, Michael Wybrow. Thank you all for your fruitful cooperation and very enjoyable discussions and correspondence.

Warmest thanks to Kristina Abels for proofreading this thesis and detecting its shortcomings which could not be fully inhibited during the writing.

Last, I want to mention the guest researchers who visited our work group and who had quite an impact on me and my research (in order of appearance): Dr. Patrizio Angelini, Dr. Tamara Mchledlidze, Prof. Dr. Stephen G. Kobourov, Jawaherul Alam, Dr. Michael A. Bekos.

# Contents

<b>1. Introduction</b>	<b>1</b>
<b>I. Business Process Visualization in 2–dimensional Space</b>	<b>5</b>
<b>2. Preliminaries and Methods</b>	<b>7</b>
2.1. Preliminaries on Graphs and Graph Drawing . . . . .	7
2.1.1. Sugiyama Framework . . . . .	10
2.1.2. Topology–Shape–Metrics and Kandinsky model . . . . .	14
2.2. Business Process Model and Notation . . . . .	22
2.2.1. Objects for Process Flow Control . . . . .	23
2.2.2. Connecting Objects . . . . .	26
2.2.3. Artifacts . . . . .	27
2.2.4. Lanes/Pools . . . . .	28
2.3. Layout Aesthetics . . . . .	31
2.3.1. A User Study on Layout Aesthetics for BPMN . . . . .	33
2.3.2. Conclusion of User Study . . . . .	44
2.4. Static 2D–Layouts for BPMN . . . . .	45
<b>3. 2D–Visualizations of Business Process Models</b>	<b>49</b>
3.1. Sketch–Driven–Layout for BPMN . . . . .	49
3.1.1. Algorithm . . . . .	51
3.1.2. Application case: Divisions (Cuts) . . . . .	56
3.2. Pattern–based BPMN–Layout . . . . .	59
3.2.1. Motivation . . . . .	59
3.2.2. Layout Patterns . . . . .	60
3.2.3. Evaluation . . . . .	68

3.2.4. Summary . . . . .	73
3.3. A Layout Approach for BPEL–workflows . . . . .	73
3.3.1. Preliminaries . . . . .	74
3.3.2. Related Approaches . . . . .	76
3.3.3. Layout Algorithm for BPEL–workflows . . . . .	78
3.3.4. Summary . . . . .	90
<b>4. Summary for Part I</b>	<b>93</b>
<b>II. Business Process Visualization in 2.5D</b>	<b>95</b>
<b>5. Introduction to 2.5D–Visualizations</b>	<b>97</b>
5.1. Motivation . . . . .	97
5.2. Terminology and related work . . . . .	98
5.2.1. Related work on 2.5D/3D–graph–layout . . . . .	99
5.2.2. Related work on (Business Process) Visualization in 3D . . . . .	101
5.3. A 3D-Framework for 2.5D-Visualizations . . . . .	102
5.3.1. Criteria and Perspectives . . . . .	103
5.3.2. Implementation . . . . .	106
5.3.3. Navigation Support . . . . .	108
5.3.4. Summary . . . . .	109
<b>6. Visualization approaches in 2.5D</b>	<b>113</b>
6.1. Motivation . . . . .	113
6.2. Approach 1: Sequential Layer Sweep . . . . .	114
6.3. Approach 2: ILP formulation . . . . .	120
6.4. Approach 3: Partition Supported 2.5D-Layering . . . . .	128
6.4.1. A 2.5D hierarchical drawing of directed graphs . . . . .	138
<b>7. Analysis and Benchmarks</b>	<b>143</b>
7.1. Data set . . . . .	143
7.2. Performance . . . . .	146
7.3. Layout Quality . . . . .	148
7.4. Discussion . . . . .	153
7.4.1. Sequential layer sweep (SLS) . . . . .	153
7.4.2. Integer linear programming (ILP) . . . . .	153
7.4.3. Partition supported layering (PSL) . . . . .	155
7.5. Conclusion and Summary . . . . .	156

<b>III. Epilog</b>	<b>159</b>
<b>8. Applications and Projects</b>	<b>161</b>
8.1. Contribution to GraphDrawing2011 Contest . . . . .	161
8.1.1. Preprocessing of the graph . . . . .	163
8.1.2. Layout implementation . . . . .	163
8.1.3. Filtering features . . . . .	163
8.1.4. Interactive application . . . . .	164
8.2. Business Process Modeling using Web2.0 . . . . .	165
8.2.1. Oryx – A Web2.0-based collaborative graphical editor . . . . .	165
8.2.2. The automatic layout algorithm and integration into Oryx . . . . .	168
8.3. Flight Navigator for Business Process Models . . . . .	168
8.3.1. Presentation of Flight Navigator . . . . .	169
8.3.2. Summary . . . . .	172
8.4. GraphArchive . . . . .	172
8.4.1. Features of the new GraphArchive . . . . .	173
8.4.2. Presentation of the new system . . . . .	180
8.4.3. Summary . . . . .	181
<b>9. Conclusion</b>	<b>185</b>
<b>Bibliography</b>	<b>195</b>





# Chapter 1

## Introduction

Graph drawing is not functional, but beautiful.

GIUSEPPE LIOTTA

Dagstuhl seminary on *Graph Drawing with Algorithm Engineering Methods*

The saying “*A picture is worth a thousand words*” is very popular among people working with diagrams. Although being an American invention (“one look is worth a thousand words”, commending the effectiveness of graphics in advertising, Printer’s Ink, December 1921), the phrase was falsely attributed to Chinese origin<sup>1</sup> which also seemed to be plausible and might have even enforced its credibility. In general, the phrase expresses the statement that a visualisation is a better description than a verbal description<sup>2</sup>. Therefore, visualization is considered a powerful tool to express data in a visual representation, e.g. image-guided anamneses, using magnetic resonance imaging (MRI) or X-rays, are typical application cases of visualization that are today immanent to our everyday life. In Figure 1, we visually depict the rules of a popular child’s game as an example of visualization. In this work, we apply the idea of visualization to business processes models.

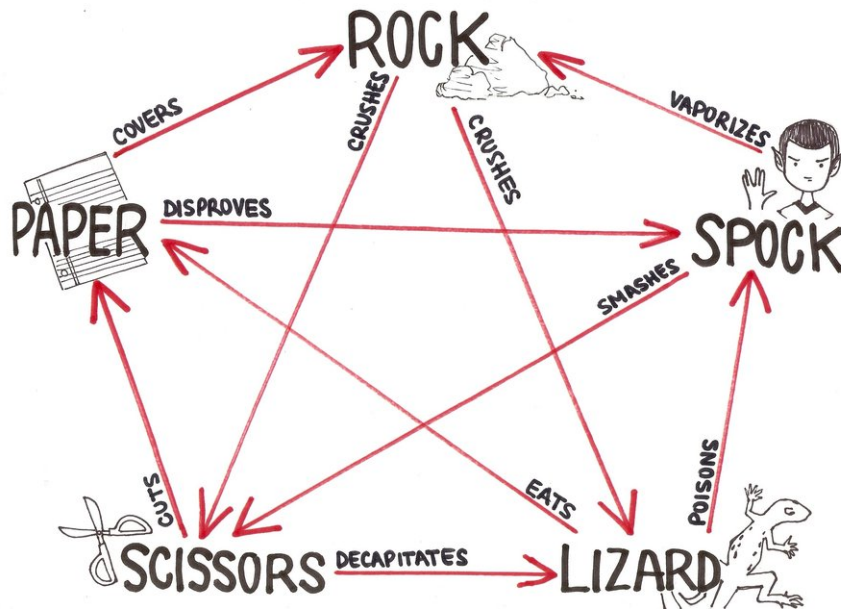
In the field of business process management (BPM), visualization quickly conquered the fundamental process of *modeling processes*. Where processes were specified in text files or later using structured text files, e.g. XML, *notation languages* took over this part of process management by introducing graphical devices to specify processes. An example for the growing importance of notation languages, for the field of software development, is the collection of different diagram styles in UML, the unified modeling language, which today, in version 2.3, offers 14 different diagram types<sup>3</sup>.

<sup>1</sup>see details on the origin of the idiom:

<http://www.phrases.org.uk/meanings/a-picture-is-worth-a-thousand-words.html>, 2012-09-30. Note that, from here on, we state for online references the date of last access.

<sup>2</sup>see [http://en.wiktionary.org/wiki/a\\_picture\\_paints\\_a\\_thousand\\_words](http://en.wiktionary.org/wiki/a_picture_paints_a_thousand_words), 2012-09-30.

<sup>3</sup>see <http://www.uml.org/>, 2012-09-30.



**Figure 1.1.:** Rules of child’s game *rock, paper, scissors* in a slightly extended version. The visual representation of the rules is backed by a graph of 5 nodes and 20 edges. The graph is complete and, therefore, called a  $K_5$ . The edges of the graph are directed. Note that the game is fair because each node has two outgoing and two incoming edges.

Source: <http://www.sodahead.com/user/profile/2457469/bazinga/>

For business processes, the notation language BPMN (business process model and notation), whose initiatives date back to 2002, is about to dominate the field of process modeling with the help of graphical notations. The newest release of BPMN 2.0 in January 2011 by the Object Management Group (OMG)<sup>4</sup> comprehends three types of diagrams to specify business processes. Also, this new release attempts to succeed BPEL (business process execution language) in terms process execution capabilities. For the modeling of processes, we consider in this work primarily process models in BPMN or BPEL, which both are de-facto notation standards for business process models and, therefore, represent a large majority of existing process models. From an algorithmic point of view, visualization of business process models is a challenging functional application of graph drawing techniques which have to fulfill aesthetic requirements of a business process model. Thus, we rely on *graphs* as mathematical constructs that are a combination of a set of objects (called nodes or vertices) and a set of binary relations between these objects (called edges). Many structures, e.g. entity-relationships, and theoretic and practical problems can be abstracted by projecting

<sup>4</sup>see <http://www.omg.org>, 2012-09-30.

them onto the structure of a graph. The field of *graph drawing* can be considered to be a sibling of graph theory and is dedicated to the design and implementation of layout algorithms for the computation of 'readable' drawings (diagrams) of graphs. Usually, as known from diagrams, graphs are embedded in a plane and rectangles are used for depicting nodes and lines are used for representing edges.

On the mission of visualizing business process models, we also touch further fields of research, e.g. layout aesthetics or business process management. Layout aesthetics attempt to formalize the readability of a diagram in order to eventually ease the burden of interpreting visual data for a user. Often, this field employs conceptions of user studies to support or reject hypotheses on effects of layout aesthetics on a user's cognition.

The focus of this work is on the development of algorithms for the computation of visualization of business process models. We also employ earlier findings on layout algorithms or graph drawing. We introduce new algorithms for the computation of layouts for business process models and we present novel approaches for the extension of powerful layout frameworks towards support of business process models. We also provide analysis of new algorithms with respect to performance and with respect to support of layout aesthetics. To endorse layout aesthetics for BPMN, we present the results of a user study conducted in order to obtain clues of user preferences for layout aesthetics in business process models.

The new algorithms are integrated into a software framework called *BPMN-Layouter* which provides an interactive modeling environment for BPMN and contains all layout algorithms developed for BPMN in this work. The layout algorithms compute visualizations in two-dimensional space and in three-dimensional space. Therefore, we also present an extension to *BPMN-Layouter*, called *3D-Navigator*, that allows to navigate freely in three-dimensional space and integrates our layout algorithms for this display concept. In the following, we describe the structure of this work.

## Outline

This work consists of three parts. The first part focuses on visualizations for business process models in two-dimensional space (*2D*). The second part comprehends the presentation of three novel approaches for computing visualizations for business process model in three-dimensional space (*3D*) while delimiting the display space by using the concept of two-and-a-half dimensions. The last part gives insights into projects that evolved to be part of my research and that all resulted in publications. The topics of these projects involve applications of visualizations and development of applications for graphs.

We begin in Chapter 2 with a thorough introduction into the field of graph drawing and define necessary notions. We also briefly recapture common algorithmic methodologies for the computation of layouts for general graphs. The results of a study on layout aesthetics of business process models are presented and are considered a starting point for constraints that should be fulfilled when designing algorithms for the visualization of business process models. Also, a previous work on a layout approach for BPMN is rehashed because it is referred to often in the remainder of this work.

The first new layout approach of this work is presented in Section 3 when we extend the Kandinsky model to support swimlanes in BPMN. The approach allows an interactive layout computation for BPMN diagrams where a new layout is based on the embedding of the input. The concept follows the idea of *preserving a user's mental map*.

Then, we introduce three novel layout patterns that represent semantics in BPMN. The patterns aim at extending the common set of layout aesthetics, which are mostly applicable only structurally or syntactically, with a set of formal rules that enable a layout algorithm to consider semantics in a business process model. All three patterns are fully integrated into the previously presented interactive layout approach. Also, we provide an analysis of the patterns and their effect on common layout aesthetics.

For business process models that employ the business process execution language (BPEL), an approach for visualization of these process models is given in Section 3.3. We define shapes and formats of BPEL elements because BPEL originally is based on XML-files only. The approach highlights hierarchical and nested structures in a process model by deriving execution paths from the models and preserving them during the computation of the layout.

In the second part, we expand the display space to three dimensions. We introduce the concept of two-and-a-half dimensions and present our self-developed software environment for the presentation of the process models in two-and-a-half dimensions, see Chapter 5.

In Chapter 6, we introduce three novel approaches for the computations of visualizations for business process models in two-and-a-half dimensions. For all three approaches, we give detailed algorithmic descriptions, followed by an evaluation and analysis on real-work business process models in Chapter 7. There, we also discuss the strengths and weaknesses of the approaches.

In the third and last part, we present different applications of visualizations and research projects that were undertaken in parallel to the work of the first two parts. The applications and projects resulted in various publications and are partially based on findings of the previous parts. The descriptions of these entertaining projects eventually lead to the conclusion at the end of this work.

**Part I.**

**Business Process Visualization  
in 2–dimensional Space**



## Preliminaries and Methods

In the following, we introduce necessary notions and definitions to build a common ground of understanding in the wide fields of *Graph Drawing* and *Visualizations* for the remainder of this work. Also, we will provide a short summary in Section 2.2 of the language *Business Process Model and Notation* (BPMN) that will be used throughout this work as standard modeling language for the business process models that are to be visualized.

Furthermore, we elaborate in Section 2.3 on layout aesthetics for business process models, and generally, on layout aesthetics as criteria for layout approaches. The chapter concludes with the presentation of previous work on a layout approach for BPMN that is referred to various times in this work.

### 2.1. Preliminaries on Graphs and Graph Drawing

In the next paragraphs, we rehash the basics of graph theory that are required for the understanding of this work.

A *graph*  $G$  is a ordered pair  $G = (V, E)$ , where  $V$  is the set of *nodes* and  $E \subseteq V \times V$ , pairs of nodes, is the set of *edges*. Nodes are synonymous with *vertices*. For a pair  $e = (u, w) \in E$ ,  $u$  and  $w$  are the endpoints of the edge  $e$ . The set of edges  $E$  can be distinguished by *directed edges*  $E_D$ , where the pairs in  $E_D$  are ordered, and *undirected edges*  $E_U$ . For an edge  $e = (u, w) \in E_U$ , it holds  $(u, w) = (w, u)$ ; pairs in  $E_U$  are not ordered. For an edge  $e_d = (v, x) \in E_D$ ,  $v$  is called *source* and  $x$  is called *target*.

If a graph  $G = (V, E)$  contains only directed edges, i.e.,  $E = E_D$ , we call  $G$  a *directed graph*, or short *digraph*. If a graph  $G$  contains directed and undirected edges, we call it a *mixed graph*. An edge  $e = (u, u) \in E$  with identical endpoints is called *self-loop*. If there are multiple edges between a pair of nodes, these edges are called *multi-edges*.

A *simple graph* is a graph without self-loops and multi-edges.

An edge  $e$  is *incident* to a node  $v$ , if  $v$  is an endpoint of  $e$ . A node  $v$  is *adjacent* to a node  $u$ , if  $(u, v) \in E$ ;  $v$  is a *neighbour* of  $u$  if  $(v, u) \in E$  or  $(u, v) \in E$ . An edge  $e_1$  is *adjacent* to a edge  $e_2$ , if  $e_1$  and  $e_2$  share an endpoint.

A subgraph  $G' = (V', E')$  of  $G = (V, E)$  is a graph with  $V' \subseteq V$  and  $E' \subseteq E$ . The subgraph  $G'$  is an *induced subgraph* if  $G'$  contains all edges  $(u, v) \in E' \subseteq E$  with  $u, v \in V'$ .

The *degree*  $\delta(v)$  of a node  $v$  denotes the number of incident edges to  $v$ . In a digraph, we distinguish the degree of a node: the *out-degree* of a node  $u$  is defined by  $\delta_{out}(u) = |\{v : (u, v) \in E\}|$ , the *in-degree* of a node  $v$  is defined by  $\delta_{in}(v) = |\{u : (u, v) \in E\}|$ . For the degrees, it holds  $\delta(v) = \delta_{in}(v) + \delta_{out}(v)$ . A  $k$ -*graph* is a graph with nodes of a maximum degree of  $k$ , or  $\max_{v \in V} \delta(v) \leq k$ . Self-loops contribute an increase of 2 to the degree of an incident node.

A *path*  $p$  of length  $n$  in a graph is a sequence of nodes  $(v_0, \dots, v_n)$ ,  $v_i \in V$ , with start node  $v_0$  and end node  $v_n$ , where  $(v_i, v_{i+1}) \in E$ ,  $\forall 0 \leq i < n$ . If the nodes in path  $p$  are pairwise distinct,  $v_i \neq v_j \forall 0 \leq i, j \leq n$  with  $i \neq j$ , we call  $p$  a *simple path*. A path  $p$  is a *cycle* if  $v_0 = v_n$ . A graph is *acyclic* if, for every pair of nodes  $(u, v) \in V \times V$ , there is no simple, cyclic path from  $u$  to  $v$ . If there is a path in  $G$  for every pair of nodes  $(u, v)$ ,  $u \neq v$ , then,  $G$  is a *connected graph*.

A *tree*  $T$  is a connected, acyclic and undirected graph. A *spanning tree* for a graph  $G$  is a tree which is a subgraph of  $G$  and which contains exactly  $|V| - 1$  edges. For a tree  $T$  and a weight function  $\omega : E \rightarrow \mathbb{N}$ ,  $T$  is a *minimum spanning tree* (MST) if the sum of weights  $\sum_{e \in T} \omega(e)$  is minimum. A *rooted tree* has exactly one root  $r$ . The nodes  $U \subseteq V$ , that are visited on a path from  $r$  to  $v$  for a node  $v \in V$ ,  $v \neq r$ , are called *ancestors*. A root  $r$  has no ancestor.

A *subtree*  $T'$  of tree  $T$  is a tree which is induced by choosing a node  $v \in V$  of  $T$  as root for  $T'$ . A node  $u$  is a *parent* of  $v$  if  $u$  is an ancestor of  $v$  and  $(u, v) \in E$ ,  $v$  is a *child* of  $u$ . A node with a parent but no child is called a *leaf*. A root is the only node without a parent.

A *topological ordering* of a directed acyclic graph  $G = (V, E)$  is a linear ordering  $\pi : V \rightarrow \mathbb{N}$  of nodes  $V$  such that  $\pi(v) < \pi(w) \forall (v, w) \in E$ .

A *partition* of a graph  $G = (V, E)$  is a mapping  $p : V \rightarrow \mathbb{N} \times \mathbb{N}$  of the nodes  $V$  to the coordinates of a cell in a two-dimensional grid. The grid consists of rows and columns; the width of the partition is defined by  $width(p) = \max\{i \mid p(n) = (i, j), \forall n \in V\}$ , and the height is given by  $height(p) = \max\{j \mid p(n) = (i, j), \forall n \in V\}$ . In an embedding with a given partition  $p(V)$ , each node  $v \in V$  is placed within the cell assigned by  $p(v)$ .

A digraph  $G = (V, E)$  with associated costs  $c : E \rightarrow \mathbb{N}$  and capacities  $u : E \rightarrow \mathbb{N}$  is called a *flow network*. We associate with each node  $i \in V$  a number  $b(i)$  which denotes its supply or demand depending on whether  $b(i) > 0$  or  $b(i) < 0$ . The *minimum cost*



*flow* for a flow network  $G$  can be stated as follows:

$$\text{minimize } z(x) = \sum_{(i,j) \in E} c(i,j) \cdot x(i,j)$$

subject to

$$\begin{aligned} \sum_{j:(i,j) \in E} x(i,j) - \sum_{j:(j,i) \in E} x(j,i) &= b(i) \quad \forall i \in V, \\ 0 \leq x(i,j) &\leq u(i,j) \quad \forall (i,j) \in E. \end{aligned}$$

The function  $x : E \rightarrow \mathbb{N}$  is called a *flow*. A flow  $f$  is *feasible* if it satisfies all of the constraints above. In order to be feasible, a minimum cost flow must satisfy the *mass balance constraints*:

$$\sum_{v \in V} b(v) = 0.$$

We refer to (Ahuja, Magnanti, and Orlin 1993) for a more thorough introduction into network flows. In the following, we introduce necessary notions that stem from the field of Graph Drawing.

**Definition 1 (Drawing of a graph).**

A mapping  $M$  of a graph  $G = (V, E)$  to the plane in  $\mathbb{R}^2$  is a drawing  $\Gamma$ , if

- $V$  is mapped onto distinct points in  $\mathbb{R}^2$ .
- $E$  is mapped on open Jordan–curves. A Jordan–curve is a planar curve that is topologically equivalent to the unit circle. The curve of an edge  $(v, w)$  connects the points that represent vertices  $v$  and  $w$ .

If no two curves intersect in a drawing  $\Gamma$  of a graph  $G$ , then,  $\Gamma$  is called *planar*. From the existence of a planar drawing for a graph, we can state:

**Definition 2 (Planarity).**

A graph  $G = (V, E)$  is *planar* if there exists a planar drawing  $\Gamma$  for  $G$ . □

A planar drawing of a graph partitions the plane into regions called *faces*. There is exactly one unbounded region which is called the *outer face*. An *embedding* of a graph is given by the clockwise cyclic ordering of the edges which are incident around each vertex. An embedding is called *planar embedding* 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 vertex  $v_f$  for each face  $f$  of  $G$  and an edge  $(v_f, v_g)$  for each edge of  $G$  separating two faces  $f$  and  $g$ . Hence, the size of the dual graph is linear. Furthermore, the dual graph is always planar. If the faces connected by an edge  $e$  in  $D_G$  are identical, we call  $e$  a *bridge*.

In a *box drawing*, the nodes are mapped to boxes (rectangles) instead of points. A point drawing is called an *orthogonal drawing* if the curve of each edge is represented by an alternating sequence of horizontal and vertical line segments. The chain of alternating segments is connected by *bends*.

If all nodes and bends along the edges have integer coordinates, the drawing is called an *orthogonal grid drawing*. Note that a graph has an orthogonal grid drawing if and only if it is a 4-graph. A drawing is called an *orthogonal box drawing* if it is an orthogonal drawing and each vertex is mapped to a box. In an orthogonal box grid drawing, the center of the boxes and the bends have integer coordinates.

The *crossing number*  $cr(G)$  of a graph  $G = (V, E)$  is the minimum number of edge crossings in any drawing of  $G$  in the plane  $\mathbb{R}^2$ . Computing the crossing number of non-planar graphs is NP-hard (Garey and Johnson 1979; Garey and Johnson 1983). Note that there are variants on how to count crossings in a graph, e.g. odd crossing number  $ocr(G)$  (Pelsmajer, Schaefer, and Stefankovic 2007), the smallest number of pairs of edges that cross an odd number of times in any drawing of  $G$ , or monotone crossing number  $mcr(G)$  (Pach and Tóth 2011), the smallest number of crossing points in a drawing of  $G$  in the plane, where every edge is represented by an  $x$ -monotone curve, that is, by a connected continuous arc with the property that every vertical line intersects it in at most one point. An upper bound on the number of crossings is  $cr(G) = O(|E|^2)$ . If every pair of edges crosses at most once, then the number of crossings is  $O(|E|^2)$ .

### 2.1.1. Sugiyama Framework

The Sugiyama framework (Sugiyama, Tagawa, and Toda 1981) aims at the computation of a layered drawing for any digraph. It is the most common approach for producing layered drawings of directed graphs. Almost all graph drawing libraries support the framework and, therefore, underline its powerfulness and generality. It consists of four steps:

1. Cycle Removal
2. Layer Assignment
3. Crossing Reduction
4. Horizontal Coordinate Assignment

The framework is now recaptured because we refer to the steps frequently in the remainder of this work. An excellent description of the framework can also be found in (Siebenhaller 2009) or, in more detail, in (Kaufmann and Wagner 2001). Together with the algorithms for each of the steps, the complete framework is also often referenced in abbreviated fashion as *Sugiyama's algorithm*.

The algorithmic framework of Sugiyama has also been extended, e.g. to handle radial layouts (Bachmaier, F.-J. Brandenburg, Brunner, et al. 2008), where the graphs are arranged in concentric circles around a starting node, and to three-dimensional layered drawings of graphs (Hong and Nikolov 2005).

### Step 1: Cycle Removal

If the connected, directed input graph  $G$  contains cycles, this step temporarily reverses edges to make the graph acyclic. Therefore, we compute a set of edges  $R \subset E$  that, when all edges in  $R$  are reversed, renders  $G$  acyclic and we call  $R$  a *feedback set*. The task is to choose  $|R|$  as small as possible. This problem is known as the *feedback arc set problem*, which is defined as the set of edges  $R$  with minimum cardinality, where it holds that  $G = (V, E \setminus R)$  is acyclic. The problem is NP-hard (Garey and Johnson 1979; Karp 1972).

The greedy heuristic described in (Eades, Lin, and Smyth 1993) determines a feedback arc set  $R$  of a simple digraph  $G = (V, E)$  in linear time such that  $|E \setminus R| \geq |E|/2 + |V|/6$ . Note that reversing all edges of a minimal feedback arc set guarantees that  $G$  is acyclic. For heuristics which do not necessarily return a minimal feedback arc set  $R$ , we can proceed as follows: We calculate a topological ordering  $\pi$  of the graph  $G = (V, E \setminus R)$  and reverse all edges  $(v, w) \in R$  for which holds  $\pi(v) > \pi(w)$ . This can be done in linear time and guarantees that  $G$  is acyclic.

### Step 2: Layer Assignment:

In the layer assignment, nodes  $V$  of an acyclic digraph  $G = (V, E)$  are assigned to layers  $l_1, \dots, l_k$ ,  $k$  denotes the number of layers, which might not be known beforehand. We call  $l_1, \dots, l_k$  a *partition* of  $V$  with  $l_i \subset V$ ,  $1 \leq i \leq k$  and  $\bigcup_{i=1}^k l_i = V$ . The partition is called a *layering* of  $G$  if for each edge  $(v, w) \in E$  with  $l(v) = l_i$  and  $l(w) = l_j$  holds  $i < j$ . Then, the *span* of an edge  $(u, w)$  is  $j - i$ .

In a layered drawing, all nodes  $v \in l_i$  are drawn on a horizontal line; thus, the layer assignment step assigns each vertex  $v \in V$  a  $y$ -coordinate. We call a layering *proper* if  $span(e) = 1$  for all edges  $e \in E$ . For edges  $e = (u, w)$  with  $span(e) > 1$  with the endpoints  $u$  and  $v$  on layers  $l_i$  and  $l_j$ , we replace  $e$  by a chain of dummy nodes  $d_{i+1}, \dots, d_{j-1}$  where vertex  $d_h$ ,  $i + 1 \leq h \leq j - 1$ , is placed on layer  $l_h$ . The nodes are connected by edges  $(u, d_{i+1})$ ,  $(d_{j-1}, v)$  as well as edges  $(d_h, d_{h+1})$  for each  $i + 1 \leq h < j - 1$ . We call

this replacement *normalization* and the result is a *normalized graph*  $G_N = (V_N, E_N)$ . With this construction, the next phase starts with a proper layering. Note that with  $O(|E|)$  edges of span  $O(|E|)$ , the number of dummy nodes is quadratic. However, the number of dummy nodes inserted in this step can be reduced (Eiglsperger, Siebenhaller, and Kaufmann 2005).

A simple layering approach is the *longest path layering*. It first places all nodes  $v \in V$  with in-degree  $\delta_{in}(v) = 0$  in layer  $l_1$ . For each remaining node  $v$ , we compute the length  $d$  of the longest path from  $v$  to a node in layer  $l_1$  and place  $v$  in layer  $l_{d+1}$ . Since  $G$  is acyclic the layering can be computed in  $O(|V|)$  using a topological ordering of the nodes. Furthermore, the layering produces a minimum number of layers.

Another approach introduced in (Gansner et al. 1993) is called *simplex layering*. Here, a layer assignment is calculated such that the total edge length, and thus the number of inserted dummy nodes, is minimized. The layering problem is formulated as a integer linear program and solved by applying the network simplex method.

Further algorithms for the layer assignment phase are also presented in (Healy and Nikolov 2001; Sander 1999).

### Step 3: Crossing Minimization:

The third step of the Sugiyama framework aims at the reduction of crossings between edges in the layered, normalized digraph. Since the number of crossings depends on the position of the nodes within each layer, this step tries to find an ordering of the nodes such that the number of crossings is minimum. The problem of finding an optimal solution is NP-complete, even if there are only two layers (Garey and Johnson 1983). Therefore, heuristic algorithms are used to reduce the number of crossings (Eades and Wormald 1994; Jünger and Mutzel 1997).

A very popular approach is the *layer-by-layer-sweep* (Di Battista, Eades, et al. 1999) where two layers  $l_i$  and  $l_{i+1}$ ,  $0 \leq i < k$  are considered at a time. The ordering of layer  $l_i$  is kept fixed while the positions of nodes in  $l_{i+1}$  are reordered such that crossings of edges between  $l_i$  and  $l_{i+1}$  are reduced. This problem is called *two-layer crossing problem* and is NP-hard (Eades and Wormald 1994). Heuristics employ two strategies:

1. fast computations of the number of crossings:

Counting the numbers is performed very often in order to decide if the number of crossings improves when interchanging node positions. For a two-layered graph  $G_l = (l_1 \cup l_2, E_l \subseteq l_1 \times l_2)$ , the so-called *bilayer cross counting problem* can be solved in  $O(|E_l| + c)$  (Sander 1999) where  $c$  denotes the number of crossings, and improvements are presented in (Barth, Mutzel, and Jünger 2004) with a time complexity of  $O(|E_l| \log(|l_1| + |l_2|))$ .

2. fast computations of improved orderings in the non-fixed layer  $l_{i+1}$ :

A common method is the *barycenter* method where the  $x$ -coordinate of each node  $v \in l_{i+1}$  is computed by the average of the  $x$ -coordinates of its neighbours. This method can be computed in linear time. A variant of this method uses the median instead of the average. Both approaches give an optimal solution for the case that the solution has no crossings (Di Battista, Eades, et al. 1999). In other cases, the heuristic variants cannot guarantee optimal solutions.

In (Jünger and Mutzel 1997), the two-layer crossing problem is formulated as an integer linear program which guarantees optimal solution. However, due to the high number of constraints ( $O(|l_{i+1}|^3)$ ), the approach is preferred in application cases with digraphs of small or medium size.

#### Step 4: Computation of Horizontal Coordinates:

In the last step of the Sugiyama framework, the nodes are assigned a horizontal coordinate in their corresponding layer. Note that each dummy node, in the normalized graph  $G_N$  with a layer ordering for reduced crossings from the last step, can cause a bend in the resulting layout. Thus, it is the goal of this step to arrange the nodes such that the edges run “as vertical as possible”, or in other words, with the least possible number of bends.

The optimization problem for keeping the edges as straight as possible may result in exponential width of the drawing and the main disadvantage is that since this problem has a quadratic objective function, it can only be solved to optimality for small instances (Kaufmann and Wagner 2001).

In (Gansner et al. 1993), it is proposed to model the optimization step as an integer linear program for a normalized layered graph  $G = (V, E)$ . The linear program corresponds to a layer assignment for a subgraph  $G_a = (V_a, E_a) = (V, \{(u, w) : u, w \in V \wedge u, w \text{ consecutive in } L_i, 1 \leq i \leq k\})$ , the compaction graph, with the following objective:

$$\min \sum_{(u,w) \in E} \Omega(u, w) \omega(u, w) \cdot |x(w) - x(u)|$$

subject to

$$x(u) - x(w) \geq \rho(u, w), \forall u, w \in V_a.$$

where  $\omega$  is a measure for the importance of an edge and  $\Omega$  denotes an internal weight for straightening long edges;  $\rho$  denotes the minimal distance between two objects. Optimality in  $G_a$  implies optimality in  $G$  and a layering for  $G_a$  gives a solution for  $G$  (Kaufmann and Wagner 2001).

Using heuristics can reduce the complexity to a linear-time algorithm (Brandes and Köpf 2001; Sander 1999). There, the *linear segments model* is applied, where each

edge is drawn as a polyline with at most three segments. The first and the last segments are always proper (endpoints lie on adjacent layers) and the middle segment is drawn vertically. (Brandes and Köpf 2001) use a *longest path*-based heuristic which runs in linear time complexity to the size of the compaction graph  $G_d$ . The method of (Brandes and Köpf 2001) is described as follows: The algorithm consists of three basic steps. The first two steps are carried out four times. Then, the results of these four runs are merged and balanced. In the first step, referred to as vertical alignment, they try to align each vertex with either its median upper or its median lower neighbor, and, then, alignment conflicts are resolved either in a leftmost or a rightmost fashion. Thus, one vertical alignment is obtained for each combination of upward and downward alignment with leftmost and rightmost conflict resolution. For the resolution, the approach distinguishes between three types of crossings in a layered graph: type 2 conflicts correspond to a pair of crossing inner segments; an inner segment is an edge between two dummy vertices. Type 1 conflicts arise when a non-inner segment crosses an inner segment. The remaining type 0 conflict corresponds to a pair of non-inner segments that either cross or share a vertex. In the second step, called horizontal compaction, aligned vertices are constrained to obtain the same horizontal coordinate. A maximal set of vertically aligned vertices is called a *block*. From the blocks, a partition of the block graph into *classes* is computed such that classes are compounds of blocks in adjacent layers that are as large as possible. Using a longest path-approach on each class, all vertices are placed as close as possible to the next vertex in the preferred horizontal direction of the alignment. Finally, the four assignments obtained are combined to balance their biases.

### 2.1.2. Topology–Shape–Metrics and Kandinsky model

The Topology–Shape–Metrics (TSM) is an approach which produces orthogonal grid drawings. It was first presented in (Tamassia 1987) and in (Tamassia, Di Battista, and Batini 1988), originally known as the GIOTTO approach. TSM is considered a convincing solution for the problem of producing satisfiable orthogonal layouts compared to other approaches (Di Battista et al. 1997). The name itself is introduced in (Di Battista, Eades, et al. 1999). The approach aims at orthogonal layouts with few crossings and bends. Various refinements and applications of the approach were published (Eglsperger 2003; Fößmeier and Kaufmann 1995; Mutzel and Klau 1998; Siebenhaller 2009; Siebenhaller and Kaufmann 2005).

The approach consists of the three steps:

1. Planarization (Topology): the topology of a drawing is found by computing a planar embedding and stored in a *planar representation*. Temporary dummy vertices for the representation of crossings are inserted for non-planar graphs.

The number of dummy vertices, i.e., the number of crossings, is subject to minimization.

2. Orthogonalization (Shape): the shape is determined using an *orthogonal representation*: for each edge, edge bends are computed by determining a list of angles which represent the route and contained bends.
3. Compaction (Metrics): the final positions of nodes and bends are determined. This step has the aim of using the minimum possible area.

In the following, we will briefly recapture the three steps.

### Planarization

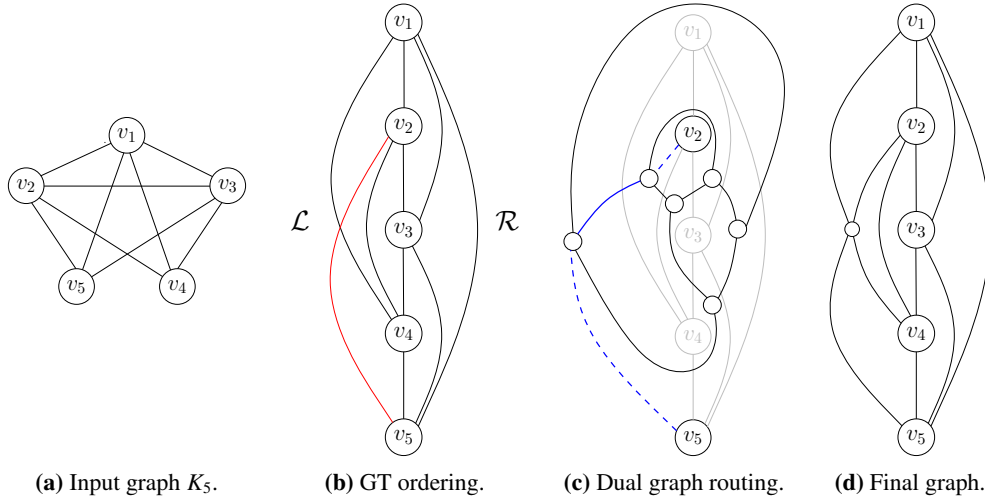
Remember that the problem of minimizing crossings is NP-complete (Garey and Johnson 1983). The planar representation determines the topology as follows: For each edge  $e \in E$  with endpoints  $v$  and  $w$ , the two possible orientations  $\langle v, w \rangle$  and  $\langle w, v \rangle$  are called *darts*. A planar representation  $\mathcal{P}$  encodes the planar embedding as follows (Siebenhaller 2009): For each face  $f \in F$ , it contains a cyclic ordered list  $\mathcal{P}(f)$  which contains the darts in clockwise order around  $f$ . The first list of the planar representation always determines the outer face.

To find a planar embedding of a graph, the following technique is very popular (Eiglsperger 2003): (1) compute an embedded planar subgraph, (2) insert the remaining edges sequentially and (3) reduce the number of crossings by rerouting of edges.

The computation of a maximal planar subgraph is NP-hard. Therefore, the planar subgraph  $G^* = (V, E^*)$  of  $G$  has as many edges  $E^* \subseteq E$  as possible but is not required to be maximal. To compute  $G^*$ , the heuristic of (Goldschmidt and Takvorian 1994), Goldschmidt and Takvorian (GT) is a favoured approach (Resende and Ribeiro 1997). At first, a node order  $\Pi(V)$  is determined and nodes are placed on a virtual chain according to  $\Pi(V)$ . Then, GT partitions  $E$  into three sets: (a) edges left of the chain ( $L$ ), (b) edges right to the chain ( $R$ ) and (c) remaining edges ( $B$ ). Of course,  $L \cap R = L \cap B = B \cap R = \emptyset$ . The partitioning is such that in both,  $L$  and  $R$ , no two edges cross with respect to  $\Pi(V)$  and  $|L \cup R|$  is large, or maximum at best. In (Goldschmidt and Takvorian 1994), it is shown that if the node order calculated in the first phase corresponds to a Hamiltonian cycle in a maximum planar subgraph of  $G$ , then the number of edges of the planar subgraph obtained by the GT heuristic is at least three quarters of the number of edges of a maximum planar subgraph. In (Siebenhaller 2009), the following heuristic to compute a GT ordering is given which exploits the Hamiltonian property: the first node  $v_1$  in the ordering  $\Pi$  is a node with minimum degree in  $G$ . Let  $v_1, \dots, v_i$  denote the first  $i$  nodes of the ordering and  $G_i$  the subgraph of  $G$  induced by the nodes of  $V' = V \setminus \{v_1, \dots, v_i\}$ . The  $i+1$ -th node  $v_{i+1}$  is a vertex of  $V'$  which is adjacent to  $v_i$  in  $G$  and has minimum degree in  $G_i$ . If there is no such node adjacent to  $v_i$ ,

vertex  $v_{i+1}$  is a node of minimum degree in  $G_i$ . This algorithm computes an ordering in  $O(|V|^2)$  time. In a randomized variant (Resende and Ribeiro 1997), the algorithm is called multiple times and, finally, the result with the largest resulting set  $|L \cup R|$ , .i.e., the largest planar subgraph, is chosen.

After the computation of a GT ordering, the remaining edges  $B$  are inserted into the planar subgraph  $G^*$ . This is achieved by inserting the edges in  $B$  one by one using a *shortest-path* approach in the dual graph  $G_D^*$  of  $G^*$ . We will also refer to this technique as *shortest-path routing* in the remainder of this work. For every edge  $e = (u, w) \in B$ , two nodes  $u'$  and  $w'$  are added to the dual graph  $G_D^*$  which represent  $u$  and  $w$ . Also, we add edges  $(u', f_u)$  from  $u'$  to nodes  $f_u$  for each face  $f$  of  $G$  which contains a dart incident to  $u$  in  $\mathcal{P}(f)$ . We perform analogously for  $w'$ . Then, the shortest path  $u' \xrightarrow{G_D^*} w'$  is computed, and we obtain a list of nodes in  $G_D^*$ ,  $u' = v_1, \dots, v_k = w'$ , where  $v_2, \dots, v_{k-1}$  correspond to faces in  $G^*$  that are traversed by the computed routing for  $e$ . Edges of  $G^*$  that are crossed by two adjacent path nodes  $(v_i, v_{i+1})$ ,  $2 \leq i \leq k-2$ , are now subdivided by a inserted dummy node which represents a crossing. The shortest path can be computed in linear time using a breadth-first-search. Therefore, the insertion of a single edge  $e$  can be computed in linear time.



**Figure 2.1.:** Example for a GT ordering for  $K_5$ . The red edge cannot be inserted into set  $\mathcal{L}$  or set  $\mathcal{R}$  without crossings. In the dual graph, the shortest path is computed and the edge is inserted. The new dummy node represents the crossing.



### Orthogonalization

In the orthogonalization phase, the shape of the orthogonal drawing is determined. The shape of an orthogonal drawing is encoded by the *orthogonal representation*  $Q$ . It extends a planar representation  $\mathcal{P}$  by adding information about bends and angles of edges. For each element in  $\mathcal{P}(f)$ ,  $f \in F$ , an ordered list of darts, we add a tuple  $(\langle v, w \rangle, s, a)$ . The first entry  $\langle v, w \rangle$  denotes the dart, and  $s$  is a bit string where the  $k$ -th bit of  $s$  represents the  $k$ -th bend when walking along the dart from  $v$  to  $w$ . A “1” represents a bend whose angle is  $270^\circ$  inside of  $f$  and a “0” a bend whose angle is  $90^\circ$ . If the dart has no bend,  $s$  is set to the empty string  $\epsilon$ . The angle between a dart and its cyclic predecessor in list  $\mathcal{P}(f)$  is specified by  $a$ , where  $a$  is a multiple of  $90^\circ$  and  $a \in \{1, 2, 3, 4\}$ .

In (Tamassia 1987), *valid* orthogonal representation for planar 4-graphs are characterized with the following properties:

Let  $G = (V, E, F)$  be a plane 4-graph with a fixed embedding and an orthogonal shape  $Q$ . Then,  $Q$  is valid if the following statements hold:

- Let  $(\langle v, w \rangle, s_1, a_1) \in Q(f_i)$  and  $(\langle w, v \rangle, s_2, a_2) \in Q(f_j)$ ,  $f_i, f_j \in F$  denote two distinct ordered lists whose darts represent the same edge. Then, the bit string  $s_1$  is equal to the reversed and flipped bit string  $s_2$ .
- Let  $L_v$  denote the set of list elements with dart  $\langle v, w \rangle$ ,  $w \in V$ . Then, for each  $v \in V$ , we have

$$\sum_{(\langle v, w \rangle, s, a) \in L_v} a = 4.$$

- Let  $\#_0$  ( $\#_1$ ) denote the function that states the number of 0’s (1’s) in a bit string. Furthermore, let  $\delta(f)$  denote the number of darts defining a face  $f$ . Since each face  $f \in F$  is a rectilinear polygon we have:

$$\sum_{(\langle v, w \rangle, s, a) \in Q(f)} a - \#_0(s) + \#_1(s) = \begin{cases} 2\delta(f) + 4 & \text{if } f \in F \text{ is the outer face} \\ 2\delta(f) - 4 & \text{otherwise.} \end{cases}$$

The number of bends of an orthogonal drawing is given by

$$\#bends = \frac{1}{2} \sum_{f \in F} \sum_{(\langle v, w \rangle, s, a) \in Q(f)} |s|.$$

Note that with a valid orthogonal representation, we can handle planar 4-graphs. With the optimized min-cost-flow algorithm that is presented in (Garg and Tamassia 1996a), the time complexity for the computation of a bend-minimum orthogonal representation of a plane 4-graph  $G = (V, E)$  is  $O(|V|^{7/4} \log |V|)$ .

For planar graphs of higher degree, we cannot draw nodes as points without producing

edge overlaps since there are only 4 different orthogonal directions. But since nodes are usually drawn as boxes, this is not an issue. Therefore, two edges that are incident to the same side of a node can share an angle of  $0^\circ$ . An orthogonal representation  $Q$  that allows the angle values  $a$  to become 0 is called a *quasi-orthogonal representation*. A quasi-orthogonal representation is called valid if there exists a corresponding planar orthogonal box drawing. As shown in (Fößmeier 1997), the above characterization of orthogonal representations also holds for quasi-orthogonal representations. An approach that allows us to handle the size of the boxes and supports quasi-orthogonal representations is the *Kandinsky model*.

**Kandinsky model** The Kandinsky model was introduced in (Fößmeier and Kaufmann 1995) and (Fößmeier 1997) and was further extended in (Di Battista, Didimo, et al. 1999) to allow for nodes of arbitrary size. Further results enabled the model to handle prescribed angles (Brandes, Kaufmann, et al. 2002), special edge shapes (for UML diagrams) (Eiglsperger, Gutwenger, et al. 2004) and port/side-constraints and clusters/partitions (Siebenhaller 2009).

The Kandinsky model requires an embedded planar graph  $G = (V, E, F)$  and, using  $F$ , a specific circular order of edges around nodes. Also, Kandinsky model assumes *grid drawings* where the center of nodes and bends of edges are put on integer coordinates. Kandinsky model allows drawings with nodes that have a degree larger than 4. This is achieved by attaching more than one edge per node side. The edges are aligned on fine grid lines that are adjacent to the nodes which are placed on a coarser grid. Therefore, for each coarse grid line, a set of  $2\kappa - 1$  fine lines is assigned in parallel, where  $\kappa \geq \max_{v \in V} \delta(v)$  to guarantee that straight-line edges are attached at the center of the node side, using the  $\kappa$ -th fine line.

Drawings with a Kandinsky model obey the bend-or-end-property and the non-empty-face-property (Eiglsperger 2003).

**Definition 3 (Bend-Or-End Property).**

*An orthogonal representation  $Q$  satisfies the bend-or-end property if for every pair  $\langle w, v \rangle, \langle v, u \rangle$  of darts following each other in a cyclic ordered list  $Q(f)$ ,  $f \in F$ , holds: either the last bend of  $\langle w, v \rangle$  or the first bend of  $\langle v, u \rangle$  is  $270^\circ$  inside of  $f$ .  $\square$*

The bend-or-end property determines that there is at most one straight-line edge on each node side. Bends that are caused by the bend-or-end property are called *vertex-bends*, other bends are called *face-bends*.

**Definition 4 (Non-Empty-Face Property).**

Let  $f \in F$  denote a triangular face with  $Q(f) = \{\langle w, v \rangle, \langle v, u \rangle, \langle u, w \rangle\}$ .

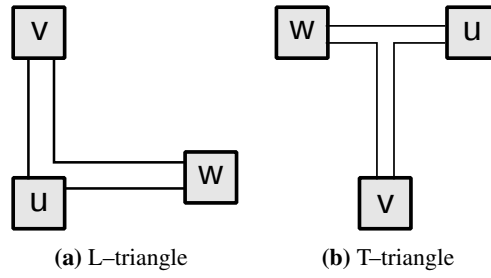
Then,  $f$  is called:

**L-triangle** if  $Q(f) = \{(\langle w, v \rangle, 1, 0); (\langle v, u \rangle, \epsilon, 0); (\langle u, w \rangle, \epsilon, 1)\}$  and

**T-triangle** if  $Q(f) = (\langle u, w \rangle, 1, 0); (\langle v, u \rangle, 1, 0); (\langle u, w \rangle, \epsilon, 0)$ .

$Q$  satisfies the non-empty-face property if it does not contain L- or T-triangles.  $\square$

Examples for the prohibited L- and T-triangles are depicted in Figures 2.2(a) and 2.2(b).



**Figure 2.2.:** Example for an L-triangle (a) and a T-triangle (b). Both triangles are not allowed in the Kandinsky model.

With the two properties, we can characterize representations in the Kandinsky model:

**Definition 5 (Kandinsky shape).**

A quasi-orthogonal representation  $Q$  is said to be of Kandinsky shape if it satisfies both, the bend-or-end property and the non-empty-face property.  $\square$

We will now present the network flow formulation for computing a Kandinsky shape of an embedded planar graph. The formulation is originally presented in (Eiglsperger 2003) and extends Tamassia's approach (Tamassia 1987). The network flow formulation will be reused and extended in later sections of this work.

Let  $G = (V, E, F)$  be an embedded planar graph with planar representation  $\mathcal{P}$ . We use  $\mathcal{P}$  to construct a network  $\mathcal{N}(\mathcal{P}) = (N, A)$  whose minimum cost flow induces a bend-minimum orthogonal representation  $Q$ . In the network, let  $c : A \rightarrow \mathbb{N}$  denote the cost function,  $u : A \rightarrow \mathbb{N}$  the capacity function and  $b : N \rightarrow \mathbb{Z}$  the supply/demand function. The set of nodes  $N$  is defined as

$$N = N_V \cup N_F \cup N_H$$

with

- $N_V$  (vertex nodes) contains a node  $n_v$  for each node  $v \in V$  with supply  $b(n_v) = 4 - \delta(v)$ .
- $N_F$  (face nodes) contains a node  $n_f$  for each face  $f \in F$  with supply/demand

$$b(n_f) = \begin{cases} -\delta(f) - 4 & \text{if } f \in F \text{ is the outer face} \\ -\delta(f) + 4 & \text{otherwise.} \end{cases}$$

- $N_H$  (helper nodes) contains a node  $n_h$  for each dart in  $\mathcal{P}$  with supply  $b(n_h) = 0$ .

The set of arcs  $A$  is given by

$$A = A_{VF} \cup A_{FF} \cup A_{HV} \cup A_{FH}$$

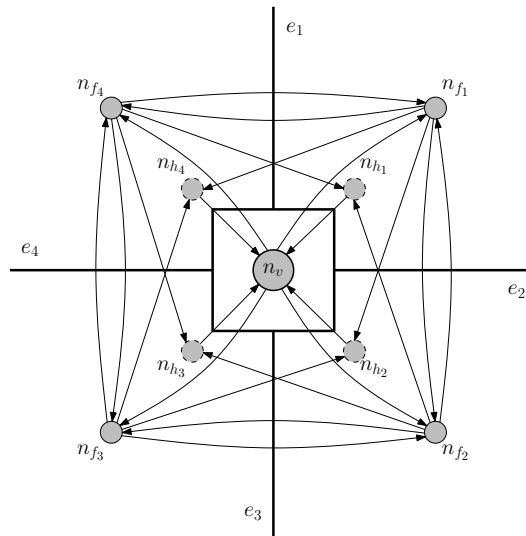
with

- $A_{VF}$  connects each node  $v$  with its adjacent faces.  $A_{VF}$  contains edges  $e_{\langle v,w \rangle}^V = (n_v, n_f)$  for each dart  $\langle v,w \rangle$  of  $\mathcal{P}(f)$ ,  $f \in F$ , starting at a node  $v \in V$ . Edges of  $A_{VF}$  have cost 0 and capacity  $\infty$ .
- $A_{FF}$  connects two faces which share an edge.  $A_{FF}$  contains edges  $e_{\langle v,w \rangle}^F = (n_f, n_g)$  for each dart  $\langle v,w \rangle$  of  $\mathcal{P}(f)$ ,  $f \in F$ , that separates  $f$  from face  $g \in F$ ,  $f \neq g$ . Edges of  $A_{FF}$  have cost 1 and capacity  $\infty$ .
- $A_{HV}$  connects the helper nodes with vertex nodes. For each face  $f_i$  and its helper node  $n_{h_i}$ ,  $0 \leq i \leq k$  around  $v$  we insert an edge  $(u_{h_i}, n_v)$  to  $A_{HV}$  with cost 0 and capacity 1.
- $A_{FH}$  connects face nodes and helper nodes in the adjacent faces. For each face  $f_i$  and its helper node  $n_{h_i}$ ,  $0 \leq i \leq k-1$  around  $v$  we insert edges  $(n_{f_{(i-1) \bmod k}}, n_{h_i})$  and  $(n_{f_{(i+1) \bmod k}}, n_{h_i})$ ,  $0 \leq i \leq k-1$  with cost 1 and capacity 1.

The network model around a node  $v$  is depicted in Figure 2.3. Note that we now have prevent the case, that an edge has two vertex–bends by adjacent faces. This happens if there is flow on edge  $(n_{f_i}, n_{h_{i+1}})$  as well as on edge  $(n_{f_{i+1}}, n_{h_i})$ , this would induce two vertex–bends at one endpoint of edge  $e_i$ , which is not allowed. Therefore, we create *devices*, a partition of the edges  $A$ . The set of devices  $D = \{d_0, \dots, d_k\}$ ,  $d_i \subseteq A$ ,  $0 \leq i \leq k$ , and a capacity function  $u' : D \rightarrow \mathbb{N}$  allow for extending the network to an *edge partition minimum cost flow problem* (Eiglsperger 2003) by inserting restrictions of the form

$$\sum_{e \in d} f(e) \leq u'(d), \forall d \in D.$$

Using the restrictions, we can now produce a valid Kandinsky shape by setting edges  $(n_{f_{(i-1) \bmod k}}, n_{h_i})$  and  $(n_{f_{(i+1) \bmod k}}, n_{h_i})$  for a node  $n_{h_i}$  in one device with capacity  $u'(d) = 1$ . In (Eiglsperger 2003), it is shown that the edge partition minimum cost flow problem is NP–hard, but it is also given a 2–approximation on a relaxation of the problem which runs in time  $O(|V|^{7/4} \sqrt{\log |V|})$ .



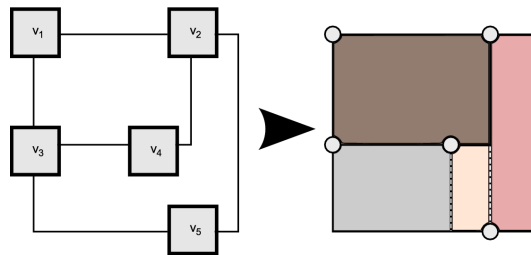
**Figure 2.3.:** Network flow model of a node  $v$  for the Kandinsky model.

### Compaction

In the compaction phase, nodes and bends are assigned to a grid drawing according to the orthogonal presentation  $\mathcal{Q}$  from the orthogonalization phase. The lengths of the vertical and horizontal edge segments between end points and bends are computed such that the area consumption of the grid drawing is small. Overlapping of nodes is not allowed and edges only cross in crossing dummy nodes of the presentation  $\mathcal{Q}$ .

Orthogonal compaction for general graphs is NP-hard (Patrignani 2001) and, in (Bannister and Eppstein 2011), a fixed-parameter tractable variant and a lower bound on the optimal solution in polynomial time of  $\Omega(n^{1/4-\epsilon})$ ,  $n = |V|$ , is given. In (Binucci and Didimo 2005), an experimental study on orthogonal compaction approaches shows that, for larger graphs, the solution that rely on ILP models are not feasible in terms of running time but, for smaller and high-density graphs, the exact algorithm outperform the flow-based heuristic. The study prefers an approach that relies on the *turn-regularity* (Bridgeman, Battista, et al. 2000) of 4-planar graphs. There, a planar orthogonal drawing of  $\mathcal{Q}$  with minimum area can be computed in  $O(n)$  time, and a planar orthogonal drawing of  $\mathcal{Q}$  with minimum area and minimum total edge length within that area can be computed in  $O(n^{7/4} \log n)$  time.

A popular approach for compaction is the *rectangular decomposition* (Tamassia 1987) where the faces are transformed to rectangles by introducing dummy edges and dummy nodes for bends and crossings, see Figure 2.4. Then, the angles of a face are searched for patterns to iteratively reduce the consumed area of the faces. Of course, the decomposition of a graph with representation  $\mathcal{Q}$  is not unique, but the resulting compaction depends highly on the decisions of the decomposition of faces into rectangles.



**Figure 2.4.:** Rectangular decomposition. Faces are transformed to rectangles by inserting dummy nodes and edges.

In (Eiglsperger 2003; Eiglsperger and Kaufmann 2002), the following approach allowed for efficient compaction of drawings in Kandinsky model with prescribed node size: first, a coarse but valid compaction is calculated. Then, this low quality compaction is improved by a post-processing algorithm in the second step. The first step is computed using a fast linear-time heuristic. The approach exploits the findings of a study (Klau, Klein, and Mutzel 2000) which states that the results of different constructive compaction heuristics are very similar after applying a flow-based one-dimensional compaction algorithm as a post-processing step.

## 2.2. Business Process Model and Notation

For the notation of business process models, we use *Business Process Model and Notation* (BPMN) (White 2004a) in most parts throughout this work. The selection of BPMN among other notation languages, e.g. EPK, UML activity diagrams, etc., is due to the standardized way to create graphical models of processes which contain all necessary information for a subsequent implementation (White 2004b, 2005) of the process model. More important, BPMN is a standard for business process modeling published by the *Object Management Group* (OMG)<sup>1</sup>. Also, it is widely used by many software vendors offering process modeling tools, often incorporated with larger solutions of business process management systems (BPMS).

Current version of BPMN passed OMG standardization process for version 2.0 in January 2011. When research for this work started at the end of 2008, version 1.2 was about to be passed in January 2009. New features of BPMN 2.0 in comparison to version 1.2 are<sup>2</sup>:

- Aligning BPMN with the business process definition meta model BPDM to form a single consistent language.

<sup>1</sup>see the website of OMG: <http://www.omg.org>, 2012-09-30.

<sup>2</sup>see description of BPMN 2.0 at [http://en.wikipedia.org/wiki/Business\\_Process\\_Model\\_and\\_Notation#BPMN\\_2.0](http://en.wikipedia.org/wiki/Business_Process_Model_and_Notation#BPMN_2.0), 2012-09-30.

- Enabling the exchange of business process models and their diagram layouts among process modeling tools to preserve semantic integrity.
- Expand BPMN to allow model orchestrations and choreographies as stand-alone or integrated models.
- Serialize BPMN and provide XML schemes for model transformation and to extend BPMN towards business modeling and executive decision support.

Since none of these features added more basic BPMN elements to the process models of BPMN 1.2, i.e., elements were only updated or described in more details to resolve ambiguities, it is fully reasonable to support 1.2 and statements on BPMN 1.2 models in this work are valid without exceptions for BPMN 2.0. The supported BPMN 1.2 models are called *collaboration diagrams* in BPMN 2.0 and are to be distinguished from conversation diagrams and choreography diagrams which were newly introduced in BPMN 2.0 and are yet to be adopted and integrated into (commercial) BPMS software. Since the new diagram types (conversation diagrams and choreography diagrams) are not fully supported by commercial tools yet and the future support is (to this day) questionable, we focused in this work on the established collaboration diagrams with support of all general elements in BPMN 2.0.

In the following, we present a short summary of modeling elements of BPMN. An extensive and comprehensive introduction can be found in (Allweyer 2010).

### 2.2.1. Objects for Process Flow Control

Sequence flow in a BPMN model is controlled by *flow objects*. They can be considered vertices in a graph. The largest group among flow objects are *events*. Events can trigger actions during a process or initiate/terminate a (sub-)process. Therefore, there are *start events*, *intermediate events* and *end events*. Depending on the specific event type, details are depicted by symbols.

#### 1. Events:



- a) Start events initiate the process sequence flow. They have a circle shape and, in our tool *BPMN-Layouter*, are assigned a green color to emphasize the initiation of 'start' as it is popular for dashboards. As a vertex  $v$ , a start event has in-degree  $\delta_{in}(v) = 0$ .



- b) Intermediate events control the sequence flow within the process. Intermediate events can trigger timer or await other events (conditions, messages, etc.) before allowing the sequence flow to continue. The shape of intermediate events is a circle surrounded by two parallel lines on the circle border. Our tool assigns intermediate events a yellow color fill.



- c) End events terminate sequence flows in process models and represent ending points of sequence flow or sequence flow branches. Note that there are end events to indicate abnormal termination of a process denoted by symbols in the circle shape of the end event, e.g. for throwing errors or triggering compensation actions. Vertices representing end events have always out-degree  $\delta_{out} = 0$ .

The variants of events are, i.e., external inputs, data transfers or internal timer or errors occur. The Tables 2.1 and 2.2 depict symbols, unique names and descriptions of BPMN events.

## 2. Activities:
















*Activities* are the actual transactions or tasks that must be processed and passed before sequence flow continues on outgoing connecting elements. Activities can be repeated (multiple instances), depending on logical conditions and termination criteria (activity loop). The shape of activities is given by yellow rectangles with rounded corners. The activities are listed in Table 2.3.

*Subprocesses* allow to encapsulate a process model in another process models. A subprocess is a decomposable activity. It can be collapsed to hide details. A subprocess contains a valid BPMN diagram. Note that a subprocess is independent from the process model it is embedded to. From the surrounding parent process, there are no direct connections to elements inside a subprocess. A subprocess is initiated regularly by a start event that is contained in the valid inner BPMN diagram. For visualizations, we treat subprocess as regular vertices in the surrounding process models. If subprocesses exist, we resolve the nested process model by performing a bottom-up-approach: compute visualization for subprocesses first, then, compute visualizations for the surrounding (sub-) processes. Since process models and the nested subprocesses form a tree structure and do not contain cycles, this is always feasible. In Table 2.3, the shape of a collapsed subprocesses, as it is used for the treatment as a regular vertex, is given. When a subprocess is expanded, it is assigned the shape of a task (yellow rectangle) with adapted size such that it contains the BPMN elements of the inner BPMN model.






## 3. Gateways:

*Gateways* are able to process multiple sequence flows. They operate as logical gates and support simple and complex logic operations, e.g. combinations of *AND*- and *OR*-conjunctions of ingoing sequence flows. Also, outgoing flows are activated depending on conditions, e.g. for a complex gateway with multiple outgoing flows, it is possible to activate none, one, selected or all outgoing branches. The gateways in BPMN are listed in Table 2.4.



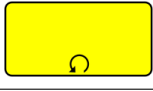
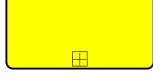


Symbol	Unique name	Description
	start_event	(Default) Start Event
	intermediate_event	(Default) Intermediate Event
	end_event	(Default) End Event
	message_start_event	Message Start Event
	message_intermediate_event	Message Intermediate Event
	message_end_event	Message End Event
	link_start_event	Link Start Event
	link_intermediate_event	Link Intermediate Event
	link_end_event	Link End Event
	rule_start_event	Rule Start Event
	rule_intermediate_event	Rule Intermediate Event
	cancel_intermediate_event	Cancel Intermediate Event
	cancel_end_event	Cancel End Event
	compensation_intermediate_event	Compensation Intermediate Event
	compensation_end_event	Compensation End Event

**Table 2.1.:** Table of BPMN events, Part 1.

Symbol	Unique name	Description
	error_intermediate_event	Error Intermediate Event
	error_end_event	Error End Event
	timer_intermediate_event	Timer Intermediate Event
	timer_end_event	Timer End Event
	terminate_end_event	Terminate End Event






**Table 2.2.:** Table of BPMN events, Part 2.

Symbol	Unique name	Description
	activity_task	Task
	activity_multiple_instance	Multiple Instance
	activity_loop	Loop
	subprocess_collapsed	Subprocess

**Table 2.3.:** Table of BPMN activities.

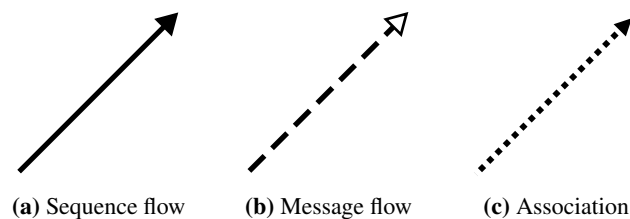
### 2.2.2. Connecting Objects

*Connecting objects* represent connections between elements and are considered edges in an underlying graph. BPMN offers three basic variants of connection objects: sequence flows, message flows and associations to attach information data. In Figures 2.5(a) – 2.5(c), the styles of the connecting objects are depicted. Sequence flow defines the execution order of activities. Message flow symbolizes information flow across organizational boundaries. Message flow can be attached to pools (see below),

Symbol	Unique name	Description
	gateway_fork_join	Fork/Join
	gateway_inclusive	Inclusive Decision/Merge (OR)
	gateway_exclusive_data	Exclusive Decision/Merge (XOR) (data-based)
	gateway_exclusive_event	Exclusive Decision/Merge (XOR) (event-based)
	gateway_complex	Complex Decision/Merge

**Table 2.4.:** Table of BPMN gateways.

activities or message events. Associations indicate information flow, e.g. a input or output data object which is described next.



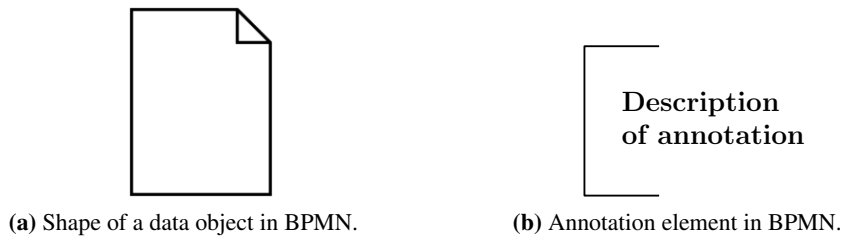
**Figure 2.5.:** Connecting objects in BPMN.

### 2.2.3. Artifacts

Artifacts are elements in BPMN for representation of data or documentation that cannot be assigned to any of the element groups mentioned above.

- Data object:

A *data object* is a placeholder for data that is input to the process or output from the process, e.g. forms (input) or reports (output). It is always attached to a element of BPMN which is the sender or recipient of the data. Therefore, a data object is not autonomous as a single element in a BPMN process model. The simple shape of a data object is depicted in Figure 2.6 (a).



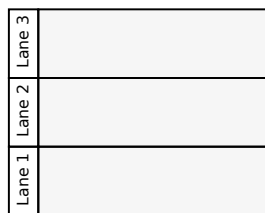
**Figure 2.6.:** Artifacts in BPMN.

- Annotations: *Annotations* offer the possibility to add comments to flow objects as well as connecting objects. Annotations are attached to the corresponding object by an association. In a graph model, annotations can be handled as common vertices. The shape of an annotation is given in 2.6 (b).

#### 2.2.4. Lanes/Pools

*Lanes* (or swimlanes) and *pools* are structural elements that render the hierarchic structure of a BPMN process models. BPMN elements are assigned to lanes. Lanes often reflect the organizational structure of a company, then, e.g. a lane represents a certain department of this company. Also, lanes can reflect roles and responsibilities of a position, e.g. marketing manager or CIO of a company. Multiple lanes can be aggregated to a pool in order to introduce a hierarchy level between roles/responsibilities. BPMN elements are not assigned to pools directly but to lanes that are subordinated.

In a model, lanes are denoted by a rectangle; elements assigned to a lane are contained in the rectangle corresponding to the very same lane. A pool is drawn as a rectangle surrounding the lanes contained in its hierarchy. In Figure 2.7, a pool with three lanes is depicted. Throughout this work, lanes or swimlanes are used synonymous.



**Figure 2.7.:** Hierarchic structure of a pool with three lanes in BPMN.

In Figure 2.8, a real-world process model of BPMN is depicted. Closing the summary of BPMN elements, we define the graph underlying a BPMN process model as it will be used in the visualization approaches in the remainder of this work.

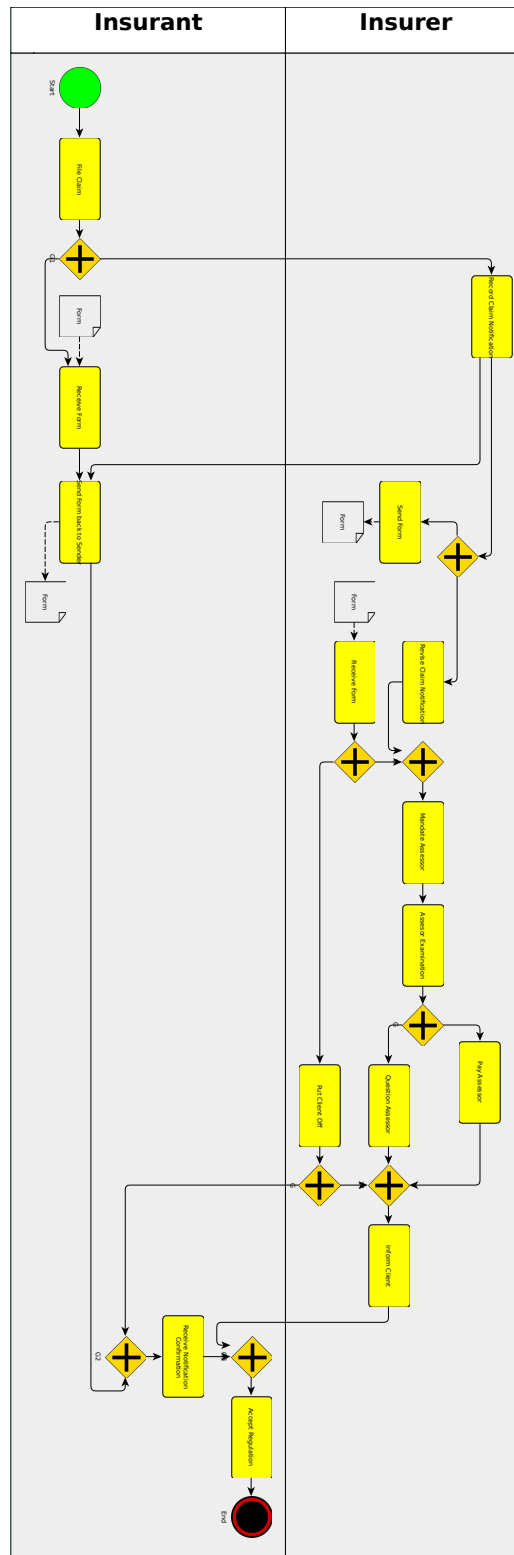
**Definition 6 (BPMN–Graph).**

A *BPMN–Graph* is a connected graph  $G = (V, E)$  with an embedding into the 2D–plane and the following additional information:

- A mapping  $\text{vertex\_type} : V \rightarrow T$ , where  $T$  denotes the set of possible types of a BPMN–element for a vertex  $v \in V$ . For a node  $n \in V$ , the shape of  $n$  is prescribed by the vertex type  $\text{vertex\_type}(n)$ .
- A mapping  $\text{edge\_type} : E \rightarrow C$ , where  $C$  denotes the set of connecting objects in BPMN, available for an edge  $e \in E$ . Values of  $C$  are: sequence flow, message flow and association.
- A mapping  $\text{swimlane} : V \rightarrow S$ , where  $S$  denotes the set of swimlanes. A vertex  $v \in V$  is assigned to exactly one swimlane  $s \in S$ . A swimlane  $s \in S$  is identified by a unique string.  $\square$

Admissible values for vertex types  $T$  are:

Events	Gateways
start_event	gateway_fork_join
intermediate_event	gateway_inclusive
end_event	gateway_exclusive_data
message_start_event	gateway_exclusive_event
message_intermediate_event	gateway_complex
message_end_event	
link_start_event	Activities
link_intermediate_event	activity_task
link_end_event	activity_multiple_instance
rule_start_event	activity_loop
rule_intermediate_event	
cancel_intermediate_event	
cancel_end_event	
compensation_intermediate_event	
compensation_end_event	
error_intermediate_event	
error_end_event	
timer_intermediate_event	
timer_end_event	
terminate_end_event	



**Figure 2.8.:** Example of a BPMN process model. The process represents the sequence flow of a notification of claim in an insurance company.

## 2.3. Layout Aesthetics

An *aesthetics* of a layout measures a graphical property of a drawing. The word *aesthetics* derives from the Greek *αισθητικός* (*aisthetikos*), meaning 'esthetic, sensitive, sentient', which in turn was derived from *αισθάνομαι* (*aisthanomai*), meaning "I perceive, feel, sense".

According to (Di Battista, Eades, et al. 1999), aesthetics 'specify graphic properties of the drawing that we would like to apply, *as much as possible*'. Aesthetics are to distinguish from *drawing conventions*. A drawing convention is 'a basic rule that the drawing *must* satisfy to be admissible' (Di Battista, Eades, et al. 1999). For instance, a drawing convention can define in a drawing what style is to use for drawing the edges, while an aesthetics prescribes that the number of crossings should be low. In general, aesthetics are often expressed as optimization problem that should be resolved such that readability of the drawing is increased.

In the remainder of this section, we introduce the concept of layout aesthetics, and the distinction from drawing conventions, and present the results of a user study<sup>3</sup> that is performed to obtain conclusions for layout aesthetics which are valid for business process models in BPMN.

Unfortunately, up to now there are neither research work nor empirical studies for aesthetics of BPMN diagrams. However, research was performed in the area of aesthetics of UML diagrams (Purchase, Allder, and Carrington 2001) or general explorations for diagram aesthetics (Purchase 1997; Purchase, Cohen, and James 1995), but none of them is applicable to BPMN diagrams without limitations. Since the underlying structure of BPMN diagrams are graphs, we will first discuss aesthetics that apply to abstract graphs (graphs without special semantics).

The formalization of layout aesthetics is reached by expressing them as optimization problems, as in (Siebenhaller 2009; Siebenhaller and Kaufmann 2006a,b). For BPMN as a graph-based notation, we consider the following layout aesthetics as necessary:

- Minimize the number of crossings of connecting elements (CROSSING).
- Minimize the area of the drawing (AREA).
- Minimize the sum of the lengths of the edges (EDGE\_LENGTH).
- Minimize the number of bends of edges (BEND).
- Minimize the number of overlapping nodes (OVERLAP).
- Maximize the number of orthogonally drawn edges (ORTHOGONAL).

In (Di Battista, Eades, et al. 1999), an overview on common graph drawing aesthetic

<sup>3</sup>The study was joint work with Sandra Seiz and Nicole Jogsch, née Ferstl. Parts of the result of the user study are published in (Effinger, Jogsch, and Seiz 2010), (Effinger, Seiz, and Jogsch 2011) and (Seiz et al. 2010).

criteria is given. Also, in (Schrepfer et al. 2009), a subset of these criteria are confirmed to be important for BPMN.

In contrast, drawing conventions aim primarily at the style and orientation of edges. Remember that drawing conventions are compulsory if applied for a drawing. A list of drawing conventions (Di Battista, Eades, et al. 1999) is given by:

- Polyline drawing: an edge is drawn as a chain of segments.
- Straight-line drawing: an edge is drawn as a single straight line segment.
- Orthogonal drawing: an edge is drawn as a chain of vertical and horizontal segments.
- Grid drawing: nodes, crossings and bends are placed on integer coordinates.
- Planar drawing: no two (or more) edges cross.
- Upward drawing: given an acyclic digraph, each edge is drawn as a curve monotonically non-decreasing in the vertical direction, e.g., in a drawing oriented left-to-right, 'upward' edges point rightward.

While fulfilling multiple aesthetics, layout algorithms solve a multi-objective optimization problem. However, for more complex notation languages, as BPMN is, the task to tackle the whole set in a single algorithm is very complex which is underlined by the low number of tools supporting BPMN layout and the poor quality of layout results, see (Effinger, Siebenhaller, and Kaufmann 2009a; Seiz et al. 2010) for an overview.

Also, BPMN has specific requirements towards layout aesthetics since it provides notation semantics within its graphical representations. The following requirements represent aesthetics that consider the specific requirements of BPMN which can be derived inspecting the standardization document:

- Nodes have different sizes (ELEMENT\_SIZE).
- Partitions must be considered, e.g. pools and swimlanes (PARTITION).
- Labeling of pools, swimlanes and elements must be feasible (LABEL).
- Maximize the number of edges respecting workflow direction (FLOW).

These principles are also mentioned in informal collections of drawing principles for special diagram types, e.g. for UML activity diagrams (Ambler 2005) or network diagrams (W. Huang, Hong, and Eades 2007). Summing up the relevant aesthetics, we state the following list of standard layout aesthetics for the layout of BPMN diagrams:

FLOW, PARTITION, OVERLAP, LABEL, ELEMENT\_SIZE, CROSSING, EDGE\_LENGTH, ORTHOGONAL, AREA, BEND.



The list of design standards is also conform on conventions within the BPMN community (Mendling, H. A. Reijers, and Cardoso 2007; Silver 2011) and corresponds to a superset of supported aesthetics in existing BPMN tools. Note that aesthetics SYMMETRY, which aims at maximizing the number of symmetrical structures, does not seem appropriate for BPMN diagrams because of the lack of a symmetric structure. A precedence of the aesthetics is only partly available if we consider user studies, e.g. (Purchase, Cohen, and James 1997). However, for this larger set of aesthetics, an order of preference has to be confirmed by a user study.

In the user study that is presented in the following, we developed a test design for a catalogue with statements that cover the different objectives of layout aesthetics. We face this catalogue with the users' personal appreciation of different layouts in order to confirm or reject our conjectures.

### 2.3.1. A User Study on Layout Aesthetics for BPMN

#### Software Tools

In this section, we give a summary of our test design and test method. For a more detailed description, we refer to (Seiz et al. 2010). For the comparison of layout results for BPMN models, we have to choose among existing solutions of tools that support modeling with BPMN and provide automatic layout features. After a market study, we obtained a list with 54 software packages that support BPMN according to the vendors. However, support of BPMN solely is not sufficient for our study, therefore, we developed the following criteria for tool selection:

1. **BPMN support:** The tool must support the standardized version 1.2 of BPMN, including all elements given in the standard.
2. **Automatic layout support:** The tool must provide the user with a feature that calculates an automatic layout of a given BPMN model. Thus, the user has the possibility to acquaint himself with the principles of layout aesthetics. The layout feature must also consider basic BPMN aesthetics, e.g. swimlanes must be respected and edges must be drawn orthogonally.
3. **Evaluation license availability:** For our study, we depend on evaluation licenses provided by the software vendors.

These criteria were compulsory. We could not consider tools or vendors that could not meet one or more of the criteria. During the study preparation, the most challenging criteria was the support of automatic layout. Also, software that supposedly supports BPMN provided repeatedly only a subset of BPMN. This prevented us from preparing our modeling test case on those tools.

After applying our criteria filters on the list of software tools, we obtained a set of five tools: three commercial tools which fulfilled the above mentioned criteria plus two variants of *BPMN-Layouter* (one variant of an earlier alpha-version and the current version which was trimmed to higher usability) that fulfilled all criteria and that were considered for our user study. In (Seiz et al. 2010), a complete list with all software vendors, which were considered, and details on the selection process are given.

### Test Method

In the following, we briefly describe the procedure instructions given to probands and the surrounding setup for the experiment of the study. The setup was successfully confirmed to be well-chosen with the help of a pre-test prior to the experiment.

The probands of the study were chosen among students with majors in economics and/or computer science. Their skills (education and/or experience) in process modeling spread from very low to very high in order to represent inhomogeneous but, because of their major subject, potential future users of modeling tools.

The software tools were installed on PCs and probands were randomly assigned to one tool. For assuring a basic common understanding on BPMN and business process modeling, probands initially were asked to insert a simple extension into an existing process using the assigned modeling tool. The set of BPMN elements needed for the implementation of the extension was given as support. The process for the modeling task was a book order instance. The process can be inspected in detail in Figure 2.14 on page 43. After the modeling part, probands were requested to prepare a presentable version of the new process. They were advised to use the automatic layout feature(s) of the corresponding tool. At this point, probands were supposed to be familiar with BPMN modeling and also with the automatic layout results and features of the respective tool. For the evaluation, each proband obtained five different layout versions of the new process, one from each evaluated tool. In the accompanying questionnaire, probands were asked to give rankings among the tools for each statement contained in the questionnaire. For comparison reasons, we chose to ask for rankings with strictly relative order among the tools. Thus, two tools cannot be ranked equally and we obtain more exact responses. The statements of the questionnaire correspond to the layout aesthetics stated in the beginning of Section 2.3. All layout aesthetics are represented by a statement. For the representation as statements, we group the statements into categories that aimed at the same attribute of a graph/diagram. The following list gives the wording of the statements as they appeared in the questionnaire:

1. Category 1: „Connection elements“ (edges)
  - a) Edges are drawn orthogonally and are inserted in such a way that they appear as short as possible (ORTHOGONAL,EDGE\_LENGTH).

- b) Edges appear to be drawn with the lowest possible number of crossings (CROSSING).
  - c) Edges appear to be drawn with the lowest possible number of bends (BEND).
  - d) Edges are drawn such that they consider the reading direction (FLOW).
2. Category 2: „Area usage“
- a) The size of the swimlanes is chosen such that all elements have enough space (ELEMENT\_SIZE).
  - b) The diagram contains unused space that could be better exploited by rearranging the elements (AREA).
3. Category 3: „Elements“
- a) Elements are arranged such that they do not overlap (OVERLAP).
  - b) The size of the elements is chosen such that the description of the label is readable (LABEL).
  - c) The assignment of an element to its swimlane is easy to perceive (PARTITION).
4. Category 4: „Coloring“
- a) The choice of colors supports to obtain a detailed comprehension of the model.
  - b) The choice of colors supports to obtain a quick overview of the model.

After the ranking of tools for each statement, probands were asked to rank the categories in decreasing importance according to their personal judgement. Also, probands were encouraged to add additional categories and/or statements that, to their opinion, were not represented in our above catalogue of statements. These proband-defined categories were also included in the ranking of categories. Eventually, the probands are asked to give a ranking of all layout diagrams presented in the file to their personal preference, independently from the above given statements.

The total time per proband of our study depended on the skill level of the probands. The periods of time ranged from 55-75 minutes per proband for the task of modeling and responses to the two questionnaires that considered layout and usability questions. The total number of participants of the study was 39.

### **Evaluation**

In this section, we present the results of our study. The study data was collected while conducting the study at two universities, at Eberhard Karls Universität Tübingen and Humboldt Universität Berlin. The circumstances were kept comparable during both times, identical software tools and identical questionnaires were used.

Note that, generally on ordinal scales, the application of mean values is not possible. However, in our case, as stated in (Gehring and Weins 2009), the distance between the ordinal items can be considered to be equal from a proband's perspective. Therefore, the application of mean values for our evaluation is rendered possible and allows an thorough evaluation using descriptive statistics.

First, we analyze the categories of the questionnaire of Section 2.3.1. For the categories, we present the ranking of all users. For the analysis, we also examine the results of the proband group separately by gender, process modeling experience and prior education in modeling.

For the analysis, the total user group is filtered yielding to subset groups with the following attributes:

- Gender: male (m) and female (f)
- Experience: business process modeling experience is rated from 'none' (0) and 'low' (1) to 'high/very high' (2+).
- Education: number of lectures (or similar events) attended in the field of business process modeling; the group is split into ranges from none (0), one (1) to two or more (2+).

The division into subset groups is due to the focus on inhomogeneous user groups. Each subset group represents a set of business process modelers with distinct backgrounds, capabilities and skills. Thus, we analyze the corresponding preferences of these groups. Before that, we propose our conjectures for the analysis in the following section.

### **Conjectures**

Our conjectures aim to confirm that the set of statements that we defined in Section 2.3.1 is convenient to support the aesthetics from Section 2.3 and correspond to the users' preferences.

For better coverage, we state our conjectures for the total group and the subset groups separately. The conjectures are also evaluated separately.

#### **Conjecture 1 (Layout Aesthetics for Total Group).**

*The user ratings of the total group correspond with respect to 1) the catalogue of category statements and 2) the users' personal appreciation (general ranking).*

If Conjecture 1 is supported, we can state that the statements of our catalogue of categories 1-4 are sufficient to fulfill the aesthetics' requirements of a BPMN diagram for all probands.

**Conjecture 2 (Layout Aesthetics for Subset Group 'Gender').**

*The user ratings of the subset groups 'Gender' (male/female) correspond with respect to 1) the catalogue of category statements and 2) the users' personal appreciation (general ranking).*

If Conjecture 2 is supported, we can confirm that the statements of our catalogue of categories 1-4 are sufficient to fulfill the aesthetics' requirements for the subset group of male, or female respectively, probands.

**Conjecture 3 (Layout Aesthetics for Subset Group 'Experience').**

*The user ratings of the subset groups 'Experience' (Experience (0), Experience (1), Experience (2+)) correspond with respect to 1) the catalogue of category statements and 2) the users' personal appreciation (general ranking).*

Conjecture 3 states the validity of the statements considering probands with different levels of modeling experience, from none (novices) to very high (experts).

**Conjecture 4 (Layout Aesthetics for Subset Group 'Education').**

*The user ratings of the subset groups 'Education' (Education (0), Education (1), Education (2+)) correspond with respect to 1) the catalogue of category statements and 2) the users' personal appreciation (general ranking).*

If Conjecture 4 is supported, we can confirm that the statements are sufficient to fulfill the aesthetics' requirements for the subset groups of probands with different educational background for business process modeling.

If the correspondence between the general ranking and the catalogue of statements is not given, that means that Conjectures 1- 4 cannot be supported, we state a conjecture for the case that, from the resulting values, we can derive a tendency to the better- and the less well-rated tools.

**Conjecture 5 (Layout Aesthetics and Tool Tendency).**

*From the user ratings of the Total Group/Subset groups 'Experience' / 'Education' / 'Gender', we can observe a match between the best- and the worst-rated tools with respect to 1) the catalogue of category statements and 2) the users' personal appreciation (general ranking).*

In addition to conjectures on whole subset groups, we state conjectures that claim the statements of our layout aesthetics adapt better the higher the users' experience or the better their education in business process modeling with BPMN. In other words, the differences between the general ranking and the catalogue of aesthetics' statements diminish inside the subset group 'Education', or 'Experience' respectively:

**Conjecture 6 (Increasing Benefit with Higher Experience).**

*The differences in the user rating of the subset groups 'Experience' diminish with increasing experience levels of probands with respect to 1) the catalogue of category statements and 2) the users' personal appreciation (general ranking).*

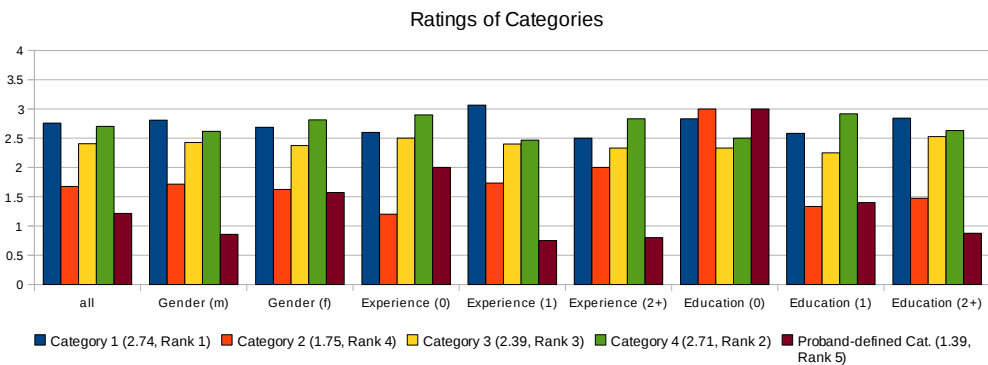
**Conjecture 7 (Increasing Benefit with Higher Education).**

*The differences in the user rating of the subset groups 'Education' diminish with increasing educational background of probands with respect to 1) the catalogue of category statements and 2) the users' personal appreciation (general ranking).*

Conjectures 6 and 7 represent our conjecture that our layout aesthetics are more efficient and exceedingly appropriate for BPMN modeling users that have a fundamental understanding of process modeling in general. Before analyzing our set of conjectures, we present the results of the ranking of categories.

**Results of Category Ranking**

The results of the category ranking show where subset groups depict aesthetics with the largest impact on the layout. The categories were ranked in a relative order from 1 to 4, respectively to 5 or 6 if proband-defined categories were given. The diagram in Figure 2.9 depicts the resulting values. After normalization, values can range from 0 to 4 while 4 is the highest (best) score.



**Figure 2.9.:** Ratings of the categories for total group (all) and all subset groups. Mean values and ranks for categories are given in parentheses.

The following interpretation of Figure 2.9 is done for each category.

For category 1 “Connection elements”, the values contain only two spikes, for group ‘Experience (1)’ (maximum) and group ‘Experience (2+)’ (minimum). However, the derivations of the spikes are diametrically opposed. Thus, no conclusion can be drawn

from these spikes. However, together with the highest mean value of all categories, the low number of derivations show that category 1 has a great effect on the acceptance of the users. Thus, we can state that users prefer layout models that correspond with statements 1.1 - 1.4.

Category 2 “Area usage” has the largest spikes in the ‘Education’ subset group. More precisely, it has an obvious peak in ‘Education (0)’. This peak renders category 2 to be the most important for ‘Education (0)’, in contrast to other subset groups. For subset groups ‘Education (1)’ and ‘Education (2+)’, the derivations are low. Thus, category 2 seems to lead to a most promising but eventually alleged effect on the layout for users with less education in process modeling. In total, category 2 ranks 4th and therefore last of all pre-defined categories.

Category 3 “Elements” is the category with the significantly least derivations in all subset groups. Thus, all users agree that element layout properties are indispensable, but no group rated this category 1st, or in other words, most preferred. Rank 3 for this category shows that it contributes important statements and may not be ignored when creating layout models.

Category 4 “Coloring” has only two minor spikes. In subset groups ‘Experience (1)’ and ‘Education (1)’, the spikes show two derivations to slightly higher, or lower respectively, values. However, in most subset groups, category 4 is rated most important or second most important after category 1. Thus, together with category 1, this category has strong effect on the layout.

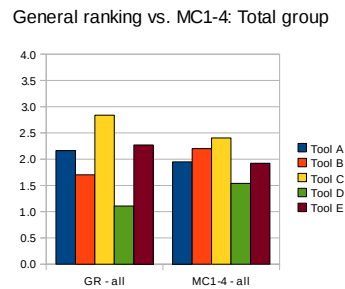
The proband-defined category shows the most diverging behaviour. While for subset group ‘Education (0)’, the category yields the highest value, it ranks last for 6 of 9 subset groups. Only subset group ‘Experience (0)’ describes a similar behaviour as ‘Education (0)’. The value of ‘Experience’ and ‘Education’ may already imply that the more experience or education modeling users are equipped with, the more the rating of models converges to a set of commonly accepted measurement attributes and aesthetic criteria (Conjectures 6 and 7).

### **Overall Ranking**

We will now face the results of the general ranking given by the probands with the aggregation over the results of the set of categories. Therefore, we calculate the mean values of the general ranking for both, the total group and each subset groups. This approach allows to consider the subset groups and their inhomogeneities. For obtaining the results, we compare data from the general ranking (GR) over all tool models with the aggregated data per category and its corresponding statements from Section 2.3.1 (mean of categories, MC1-4).

**By Total Group:**

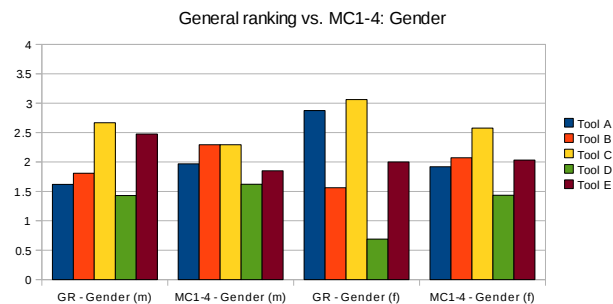
The contrast between the GR and MC1-4 is depicted in Figure 2.10 and becomes visible when taking a closer look at Tools A,B and E. Their ranking is not equal over both measurements. However, as Tool C always ranks 1st and Tool D ranks last, we can state that our methods can distinct between tools that suffice users' preferences and those that fail. Thus, we have to reject Conjecture 1, but the support of Conjecture 5 for the total group is successful.



**Figure 2.10.:** Comparison of general ranking (GR) and mean of categories (MC1-4) for total group.

**By Gender:**

The gender subset group contains the male and female subsets of probands. For the comparison between GR and MC1-4, depicted in Figure 2.11, we state that a link between GR and MC1-4 is not given. The ratings of both, male and female, differ for GR and MC1-4. However, in all subset groups, Tool C ranks 1st and Tool D ranks last which, as in the total group, allows us to distinct between best and imperfect tools. Therefore, Conjecture 2 must be rejected, but Conjecture 5 is supported for subset group 'Gender'.



**Figure 2.11.:** Comparison of general ranking (GR) and mean of categories (MC1-4) for subset groups of 'Gender'.

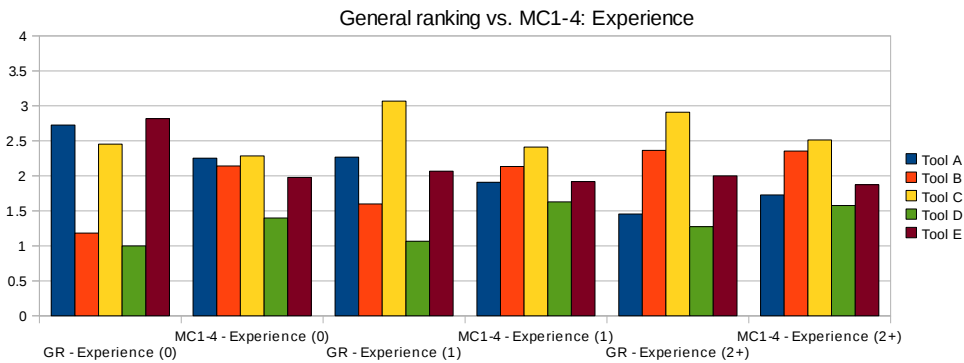
**By User experience:**

The subset groups for filter 'Experience' sum up to 6 groups, see Figure 2.12. For



'Experience (0)', the values and rankings of GR and MC1-4 differ, but considering users with at least little experience in group 'Experience (1)', we observe only a minor switch of ranks for Tool A and B. For users with high experience in group 'Experience (2+)', the values correspond and the ranking is identical.

Since the values of 'Experience (0)' do not correspond, we have to reject Conjecture 3. However, since the differences clearly diminish with higher experience of modeling users, Conjecture 6 is supported. Also, for both subset groups 'Experience (1)' and 'Experience (2+)', we can confirm that Conjecture 5 is supported since one can obviously inspect the best and the deficient tools chosen by the probands.



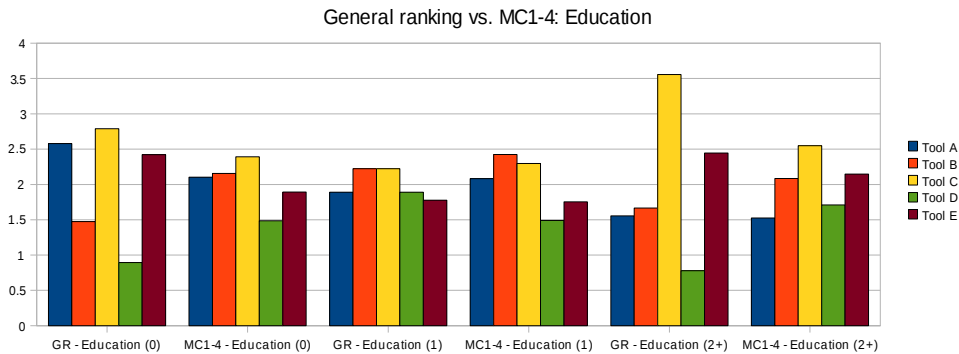
**Figure 2.12.:** Comparison of general ranking (GR) and mean of categories (MC1-4) for subset groups of 'Experience'.

#### By User Education:

The filter 'Education' also creates 6 subset groups. In Figure 2.13, the values are presented. We can state that for 'Education (0)', the values diverge, there are two swaps of tools' ranking (Tool A, B and E). However, for groups with higher modeling education, we only notice one swap: Tool D and E for 'Education (1)', Tool A and D for 'Education (2+)'. Moreover, the distance between values that cause swaps diminish the higher the education level, e.g. the distance between Tool A and D for 'Education (2+)' is almost vanished to 0.19 on a 0-to-4-scale.

Also, as for subset group 'Experience', we cannot state that the corresponding Conjecture 4 for group 'Education' is supported. On the other hand, we can confirm an improvement of the results the higher the educational background, therefore Conjecture 7 is supported. Moreover, Conjecture 5 is supported for 'Education (0)' and clearly for 'Education (2+)', too.

Before closing the evaluation section, we sum up the results and conjectures of this section. Conjectures 1, 2, 3 and 4 are rejected since the differences between the values of the general ranking (GR) and the mean of categories (MC1-4) diverge in the total



**Figure 2.13.:** Comparison of general ranking (GR) and mean of categories (MC1-4) for subset groups of education.

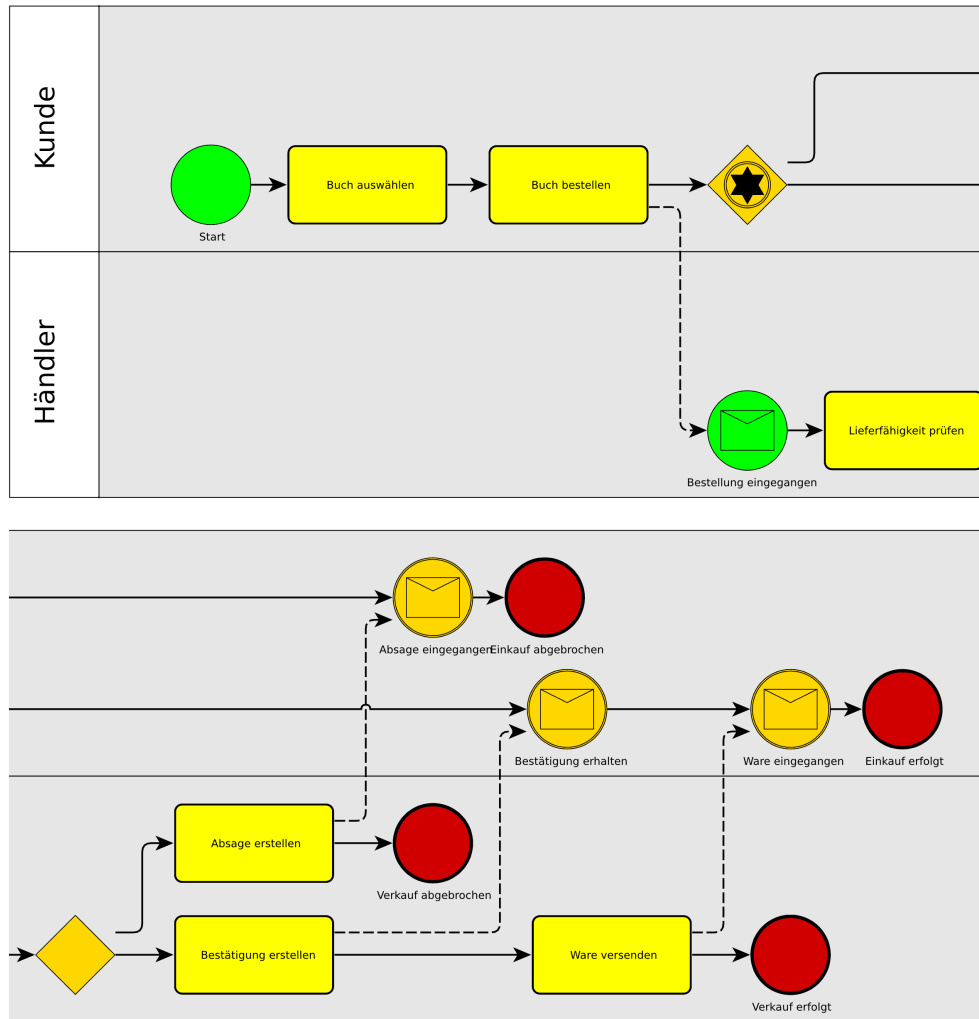
group and at least one subset group of the filters (gender, education, experience). However, we confirm Conjecture 5 for 7 of total 9 subset groups. The Conjecture states the correct prediction of the user preference tendency of a tool’s layout capabilities when using our catalogue of statements. Also, Conjectures 6 and 7 can be confirmed. They state that the values of subset groups ‘Experience’ and ‘Education’ correspond for GR and MC1-4 when considering the groups with average or higher practice experience, or at least basic education respectively, for the field of business process modeling.

The layout from Tool C that was favoured by the probands in the general ranking is depicted in Figure 2.14.

### Related work on Studies of Layout Aesthetics

Layout aesthetics are also known as secondary notation. The term of secondary notation is due to fundamental cognitive research of (Petre 2006, 1995). Secondary notation is an important part of the cognitive dimensions of notation framework developed by (Green and Blackwell 1998). It is also used in the study design of (Schrepfer et al. 2009) where differences in understandability of business process models between novices and experts are targeted. In (Ware, Purchase, et al. 2002), graph aesthetics are used for cognitive measurements.

The area of research on layout aesthetics is broader if considering other diagram types or graph classes, too. There are diagram types widely related to BPMN, e.g. UML-diagrams which were part of an analysis in (Sun and Wong 2005) where laws for diagram layout were formalized, e.g. the ‘law of proximity’. This would refer to our statements ‘ORTHOGONAL’ and ‘ELEMENT\_SIZE’. Aesthetics for UML-diagrams were also proposed in the context of automatic layout in (Eichelberger 2005). Suggestions of aesthetics are also given for Petri-Nets in (Jensen 1996) or for more general graphs in (Coleman and Parker 1996). In (Genero, Poels, and Piattini 2008), metrics



**Figure 2.14.:** Layout of Tool C scored highest in 7 of total 9 subset groups. For reading purposes, the image is for this paper manually cut in two halves of equal width; the upper first part is originally directly connected to the lower second part. As the study was conducted in German, the labels are given in German language.

are proposed and validated for entity-relationship (ER-) diagrams. Aesthetics, or 'effects', for social-network visualization are considered in (W. Huang, Hong, and Eades 2007) with the focus on edge crossings, or in our terminology, aesthetics 'CROSSING'.

The set of layout aesthetics is further enriched by research works and studies of (Purchase 1997; Purchase, Allder, and Carrington 2001; Purchase, Cohen, and James 1995)

where a subset of aesthetics is target of a ranking analysis. Further confirmation for aesthetics ranking is conducted by (Apfelbacher et al. 2006) that states that diagrams with short edges can be read more easily because the nodes' proximity is higher and the probability of crossings ('CROSSING') is lower. Also, (W. Huang, Hong, and Eades 2008) confirmed the importance of 'CROSSING'. Moreover, general modeling guidelines, e.g. (Apfelbacher et al. 2006), are available and provide a fundamental set of aesthetics.

User studies concerning aspects of layout aesthetics are done by (Purchase, Allder, and Carrington 2001) for UML, (Mendling, H. A. Reijers, and Cardoso 2007) for syntactical structures in process models and also (W. Huang, Eades, and Hong 2008) for general graphs. The cognitive complexity of integrating multiple diagrams with different notations is examined in (Hahn and J. Kim 1999). In (Agarwal, De, and Sinha 1999), object-oriented (OO) models are compared to process-oriented (PO) models with the objective of user comprehension.

### 2.3.2. Conclusion of User Study

In the user study, we analyzed secondary notation in terms of layout aesthetics and users' preferences of layout aesthetics for BPMN with consideration of inhomogeneous user groups. We proposed a catalogue of criteria which promises modeling results that are well-accepted by most users when being applied in algorithms.

The formalized catalogue is tested by the conduction of a user study. The results of the study were presented and interpreted. The data analysis of the study results was performed with respect to not only all participants at a time but also to subset groups according to modeling experience, modeling education and gender. We were able to show that our layout catalogue is most appropriate when applied for users with average or higher practice experience and users with at least basic knowledge in business process modeling. We also could show that our catalogue of statements is sufficient to predict the tendency of the users' judgement for the layout capabilities of a tool.

Concluding, our results can be used for designing powerful algorithms in modeling tools for BPMN that produce layout for BPMN diagrams that will be well-received by users. We pursue this goal when presenting visualization approaches in 2D in Chapter 3 and for two-and-a-half dimensions in Chapter 6.

For the remainder of this work, we refer to the following set of aesthetics as the *Standard Layout Aesthetics* for BPMN:

PARTITION, FLOW, OVERLAP, ELEMENT\_SIZE, EDGE\_LENGTH,  
CROSSING, ORTHOGONAL, LABEL, AREA, BEND.

The set of aesthetics is ordered according to the results of the category ranking. Note that aesthetics PARTITION is not part of the category ranking because it is immanent to valid BPMN models. We therefore added PARTITION as a required aesthetics at highest priority to this list.

## 2.4. Static 2D-Layouts for BPMN

In this section, we briefly present results from previous work (Effinger 2008; Effinger, Siebenhaller, and Kaufmann 2009b) that provide the computation of a 2D-layout for BPMN models. The work is based on results from (Siebenhaller and Kaufmann 2006a,b) and uses the TSM approach and the Sugiyama framework.

In (Siebenhaller and Kaufmann 2006a), the concept of *p-planarity* is introduced:

**Definition 7 (P-Planarity).**

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

The concept of *p-planarity* and the introduction of partitions allows for the support of swimlanes by assigning the nodes of a business process models to rows; nodes in the same swimlane are assigned to identical rows in the partition.

*P-planarity* is related to *c-planarity* (Siebenhaller and Kaufmann 2006a), or *clustered planarity* (Cornelsen and Wagner 2003) of compound, or clustered graphs: a compound graph  $G_C = (G, T)$  is induced by a graph  $G = (B \cup C, E_G)$  and a tree  $T = (B \cup C, E_T)$  which is called *inclusion tree*; the set  $B$  contains the base nodes and the set  $C$  of compound nodes defines the hierarchy in the clustered compound graph  $G_C$ . A directed path  $v \rightarrow^* w$  in  $T$  denotes that  $w$  is part of the compound node  $v$ . Base nodes are leafs in  $T$  and compound nodes are inner nodes of  $T$ . A *cluster drawing* of a compound graph  $G_C$  is a drawing where each compound nodes is depicted as a closed region, e.g. a rectangle. Note that compound nodes can be nested, thus, these regions can contained further regions. Then, a compound graph is *c-planar* if it has a planar and a cluster drawing at the same time. To characterize cluster drawings,  $\langle \alpha, \beta, \gamma \rangle$ -drawings are introduced in (Angelini et al. 2012), where  $\alpha$  is the number of edge-edge crossings,  $\beta$  denotes the number of edge-region crossings and  $\gamma$  is the number of region-region crossings. With this characterization, (Angelini et al. 2012) show that:

$p^{1,2}$	$p^{2,2}$	$p^{3,2}$	$p^{4,2}$
$p^{1,1}$	$p^{2,1}$	$p^{3,1}$	$p^{4,1}$

**Figure 2.15.:** Example for the matrix-like arrangement of the rectangles of cells in a partition.

1. minimizing  $\alpha$  in an  $\langle \alpha, 0, 0 \rangle$ -drawing is NP-complete even if the underlying graph is a matching. For a graph  $G = (V, E)$ , a matching  $M \subseteq E$  is a set of pairwise non-adjacent edges. Therefore, no two edges share a common endpoint.
2. minimizing  $\beta$  in a  $\langle 0, \beta, 0 \rangle$ -drawing is NP-complete, even if the compound graph is  $c$ -connected embedded and flat. If each compound node in  $C$  induces a connected subgraph of  $G$ , then  $G_C$  is  $c$ -connected.  $G_C = (G, T)$  is called *flat* if, in any path from the root to a leaf of  $T$ , there are at most three nodes. An *embedded* graph uniquely defines cyclic orders of edges incident to the same node. Here, the embedding is given and fixed.
3. minimizing  $\gamma$  in a  $\langle 0, 0, \gamma \rangle$ -drawing is NP-complete.

P-planarity delimits the construct of  $c$ -planarity. A graph  $G = (V, E)$  with a partition  $p$  can be formulated as a compound graph  $G_C = (G, T)$ . Then, tree  $T$  has height 3 and is induced by the partition  $p$  of  $G$ . The inner nodes of  $T$  are given by compound nodes for the rows and columns in depth 1 and compound nodes for each single partition cell of  $p$  in depth 2. The leaves of  $T$  are the base nodes  $V$  of  $G$ . Note that in a partition, region-region crossings are not possible because partition cells are denoted by rectangles in a matrix-like arrangement, as depicted in Figure 2.15.

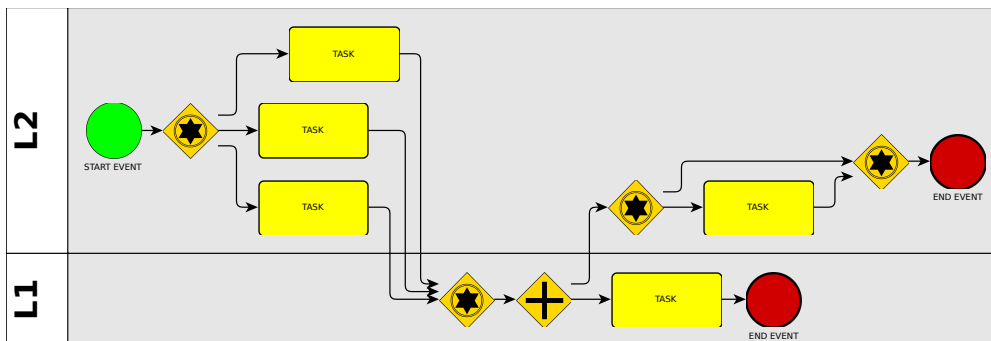
In (Siebenhaller 2009), the Sugiyama framework is extended to support p-planarity: for the partition cells, dummy nodes are inserted into the layers and, during the crossing minimization, a swap of two nodes is not allowed if it requires to cross such a dummy node. This extension of the Sugiyama framework is then integrated as a planarization phase of the TSM approach. In the orthogonalization phase, the cells are preserved by preserving the shape of the partition, i.e., border edges of the partition are not allowed to bend and are considered as fixed.

The overall algorithm for an input graph  $G = (V, E)$  and a partition  $p$  is as follows:

1. Pre-processing:
  - a) if  $G$  is not connected, insert temporarily connecting edges between connected components.

- b) add the partition graph  $G_P$  to  $G$ , where  $G_P$  represents the grid graph of  $p$ , see (Siebenhaller and Kaufmann 2006a).
2. Planarization:  
We apply the extended Sugiyama framework with support of p-planarity.
3. Orthogonalization
4. Compaction:  
We use the approach of (Eiglsperger 2003) that allows for drawings in Kandinsky model with prescribed node size.
5. Post-processing:
  - a) remove all temporary nodes and edges (e.g. for connectedness of  $G$ ).
  - b) add rectangles to the drawing that denote the partition cells, .i.e., one rectangle per row of the partition  $p$ , in order to highlight the swimlanes of the BPMN model.

An example of a layout produced by the approach is depicted in Figure 2.16. Note that this approach does not consider a previous embedding of  $G$  or a pre-existing layout of the BPMN model represented by  $G$ . The approach produces *static* layouts only. This issue will be tackled when computing layouts with respect to given sketches of the model, see Section 3.1.



**Figure 2.16.:** Example of a drawing with the 2D-approach for BPMN models that was presented in previous work (Effinger, Kaufmann, and Siebenhaller 2009).





## 2D–Visualizations of Business Process Models

In this chapter, we present approaches that compute 2D–visualizations with respect to the specifics of business process models. At first, we will extend an interactive layout approach in order to be able to comply with aesthetics PARTITION. Then, in Section 3.2, we will introduce semantics in visualizations of BPMN models. Therefore, we define three new layout patterns that apply to semantics of BPMN models and that are modeled such that the layout algorithm in use is able to handle the patterns when computing a BPMN layout. Finally, we will present a new approach for computing visualizations of BPEL models, see Section 3.3. BPEL is a text–based markup language to document executable business processes. A brief introduction to BPEL will be given before stepping into the details of the BPEL layout algorithm.

### 3.1. Sketch–Driven–Layout for BPMN

In the practice of modeling, changes must be done rather often in an existing model. The same applies for layouts; a layout can undergo changes since the underlying model changes. However, if the model changes, the layout should not be changed substantially. Thus, the goal should be, not to destroy the user’s mental map of a model. The *Sketch–Driven–Layout*–approach (SDL) addresses this challenge. The original idea of SDL stems from (Brandes, Kaufmann, et al. 2002). SDL considers an existing drawing and calculates a new layout targeting to fulfill given objectives. It is built on a Bayesian framework in order to measure the amount of changes in a graph. In the original approach, changes in edges’ bends and angles were minimized when performing a layout algorithm.

The benefits of SDL are obvious: Any time, a BPMN model is changed, an automatic layout for the model can be calculated without changing the mental map of the modeling user. This allows to integrate the layout step in the process of designing the model: When a connecting object or a flow object is added/removed, SDL is called and adapts the layout to fulfill the layout requirements. Using SDL for interactive layout enables a modeling tool to offer an automatic and interactive layout approach that supports the human designer during the design process of a model.

However, the original idea of SDL was limited, e.g. since PARTITION was not considered. Thus, we had to extend SDL such that it considers swimlanes beyond preserving the user's mental map. This could be solved by extending the Kandinsky network. In the remainder of this section<sup>1</sup>, we will present our approach of enabling SDL for BPMN.

Since the foundations of SDL, as presented in (Brandes, Kaufmann, et al. 2002), are based on the Kandinsky model (Föbmeier 1997; Föbmeier and Kaufmann 1995), we will briefly recapture this model in the following paragraphs; for a more detailed description of the Kandinsky model, see the introduction in Section 2.1.2.

Remember that a Kandinsky model requires an embedded planar graph  $G = (V, E, F)$  where  $F$  is the set of faces and, using  $F$ , a specific circular order of edges around nodes is given. Also, Kandinsky model operates on *grid drawings* where the center of nodes and bends of edges are put on integer coordinates. An orthogonal shape  $Q$  maps the set of faces  $F$  to ordered lists of tuples. For each face  $f$ , the tuples  $(e_i, a_i, b_i)$ ,  $1 \leq i \leq |Q(f)|$  with edge  $e_i \in E$  and  $a_i \in \{1, \dots, 4\}$ , where  $a_i$  represents the angle formed by multiples of  $90^\circ$ , give the shape along the face  $f$ ;  $b_i$  denotes the list of bends of edge  $e_i$ , given by a string. A *quasi-orthogonal shape* allows values  $a_i = 0$ , denoting that the succeeding edge  $e_{i+1}$  of an edge  $e_i$  in a tuple is adjacent to the same side at a node. A quasi-orthogonal shape  $q$  is *valid* if there is a planar orthogonal box drawing with quasi-orthogonal shape  $q$ . A planar orthogonal box drawing is a planar drawing where nodes are mapped to boxes and edges are mapped to sequences of vertical and horizontal segments.

Drawings with a Kandinsky model obey the bend-or-end-property ( $P1$ ) and the non-empty-face-property ( $P2$ ).  $P1$  states that, given two edges  $e_1$  and  $e_2$  which are adjacent to the same side of a node in a face  $f$  with  $e_2$  being the successor of  $e_1$  in the embedding, either  $e_1$  must have a last bend in  $f$  with  $270^\circ$  or  $e_2$  must have a first bend with  $270^\circ$  in  $f$ .  $P2$  prevents cases of degenerated triangles in the graph, see Chapter 2 for more details.

---

<sup>1</sup>Parts of this section were published in (Effinger, Siebenhaller, and Kaufmann 2009a) and (Effinger, Kaufmann, and Siebenhaller 2009).

Note that with the topology–shape–metrics (TSM) approach, a *Kandinsky drawing*, i.e., a drawing of a graph in valid Kandinsky model, with the minimum number of bends can be obtained using a minimum–cost–flow in a network (Eiglsperger 2003; Tamassia 1987).

Before describing the original SDL–approach, we introduce the *Bayesian paradigm* that is employed for difference metrics in SDL. In dynamic graph drawing, a series of graphs, which stems from a single graph and its modified successors over time, is to be visualized under the premise that changes between consecutive graphs, or frames in an animation, should be minimized in order not to destroy a user’s mental map (Eades, Sugiyama, et al. 1991). An example for dynamic graph drawing is the task of visualizing time–series graph in higher dimensions (Dwyer 2004). The Bayesian paradigm (Brandes and Wagner 1997) suggests to incorporate a difference metric as a penalty in the objective function of layout algorithms that are based on the optimization of objective functions. Difference metrics (Bridgeman and Tamassia 1998) describe the measurement of layout aesthetics between two drawings of a graph. The original SDL–approach uses difference metrics for measuring changes in angles and bends in drawings with orthogonal shapes.

### 3.1.1. Algorithm

In the following, we will describe the algorithm of SDL for BPMN models. The new approach is based on the original SDL–approach from (Brandes 1999; Brandes, Kaufmann, et al. 2002). Therefore, we will introduce the preliminaries of the original approach before presenting the extended algorithm for BPMN models.

For SDL, let  $\Sigma$  be an *admissible* drawing, called a *sketch*, of a graph  $G_\Sigma = (V_\Sigma, E_\Sigma)$ . A sketch is admissible if no edge and non–incident node overlap and no more than two edges cross in the same point. Then, the objective of SDL is to determine a orthogonal box drawing of  $G_\Sigma$  which fulfills the following properties:

- (i) the topology of  $G_\Sigma$  is preserved.
- (ii) the final drawing is in the Kandinsky model.
- (iii) angles in the final drawing deviate only little from angles in the sketch.
- (iv) the drawing contains few bends.

Property (iii) constitutes the stability of the drawing and Property (iv) ensures readability. Readability is measured in SDL by the number of bends and stability in the deviation of angles. Therefore, SDL is defined as a bi–criteria optimization problem on a quasi–orthogonal shape  $Q$ . For the number of bends  $B$  in a quasi–orthogonal shape  $Q$ , we know that

$$B(Q) = \frac{1}{2} \sum_{f \in F} \sum_{(e,a,b) \in Q(f)} |b|.$$

The stability, or deviation of angles, is given by the difference  $\Delta_A$  of the angles in  $Q$  and the angles of the shape  $S$  in the sketch

$$\Delta_A(Q, S) = \sum_{f \in F} \sum_{1 \leq i \leq |f|} |a(S, f, i) - a(Q, f, i)|,$$

where  $a(Q, f, i)$  denotes the value of the  $i$ -th angle of face  $f$  in shape  $Q$ . The difference  $\Delta_B$  in edge bends is given by

$$\Delta_B(Q, S) = \sum_{f \in F} \sum_{1 \leq i \leq |f|} \Delta(b(S, f, i) - b(Q, f, i)),$$

where  $b(Q, f, i)$  denotes the value of  $i$ -th bend of face  $f$  in shape  $Q$  and  $\Delta(s_1, s_2)$  denotes the edit-distance of two strings which allows insert and delete operations only.

Introducing weights  $\alpha, \beta, \gamma$  gives us the objective function for SDL

$$D(Q|S) = \alpha \cdot \Delta_A(Q, S) + \beta \cdot \Delta_B(Q, S) + \gamma \cdot (B(Q) - B(S)).$$

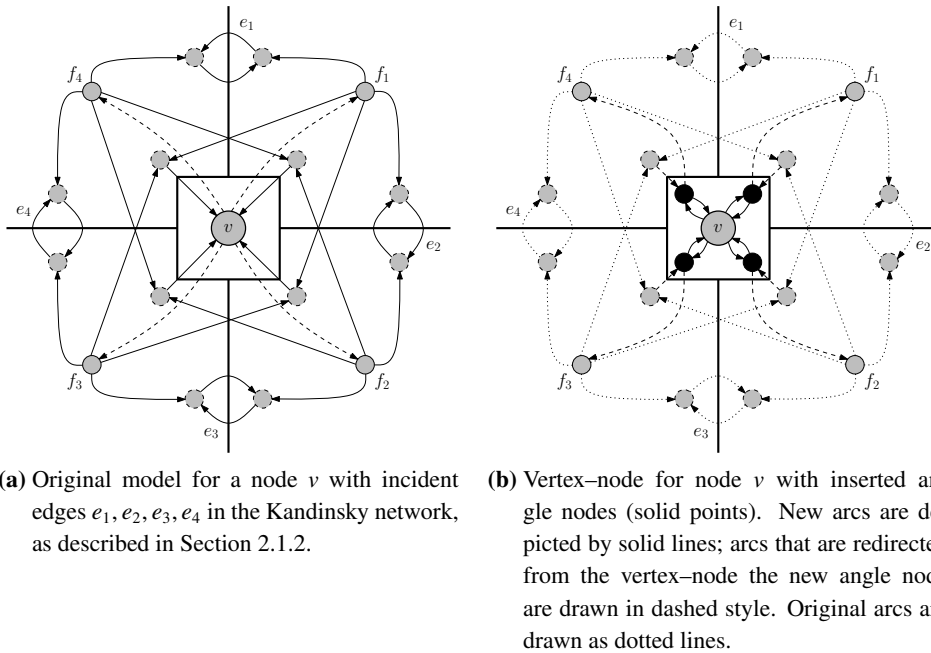
The weights  $\alpha, \beta, \gamma$  control the priority of difference of angle bends, edge bends and bend number. Formally, the problem statement for SDL is now:

**Problem 1 ((Brandes, Kaufmann, et al. 2002)).**

*Given a quasi-orthogonal shape  $S$  of a planar graph  $G$ , find a valid quasi-orthogonal shape  $Q$  of  $G$  in the Kandinsky model such that  $D(Q|S)$  is minimum.  $\square$*

In the following, we describe the modifications of the min-cost-flow network representation of Kandinsky to adapt the network to be able to solve Problem 1 which is a specification of the CONSTRAINED KANDINSKY BEND MINIMIZATION problem (Eiglsperger 2003). We assume familiarity with the min-cost-flow representation of the Kandinsky model as introduced in Section 2.1.2.

The first modification is performed on the vertex-nodes in the network to model angles around a nodes. For each angle between two adjacent edges  $e_1$  and  $e_2$ , that are consecutive in the circular order of edges around a node  $n$ , an *angle-node*  $a_{e_1, e_2}^n$  is inserted. We insert two arcs in the network connecting  $a_{e_1, e_2}^n$  and the vertex-node  $v$  of  $n$ ; one arc is directed to  $a_{e_1, e_2}^n$ , the other arc is directed in the opposite direction. Both arcs have unconstrained capacity and are assigned cost  $\alpha$ . Corresponding arcs in the original network that were connected to  $v$  are now connected to  $a_{e_1, e_2}^n$ . Therefore,  $v$  is only connected to angle nodes. For each angle node  $a_n$ , there is a *target angle*  $ta(a_n)$  in the shape  $S$  of the sketch. If  $ta(a_n) > 0$ , then  $a_n$  is connected to the source with an arc which is assigned cost 0 and capacity  $ta(a_n)$ , else,  $a_n$  is connected to the sink with an arc which is assigned cost 0 and capacity  $ta(a_n)$ . From the supply of  $v$ , we remove  $ta(a_n)$  for each angle node  $a_n$  that is connected to  $v$ . In Figure 3.1, the modification of a vertex-node is depicted.

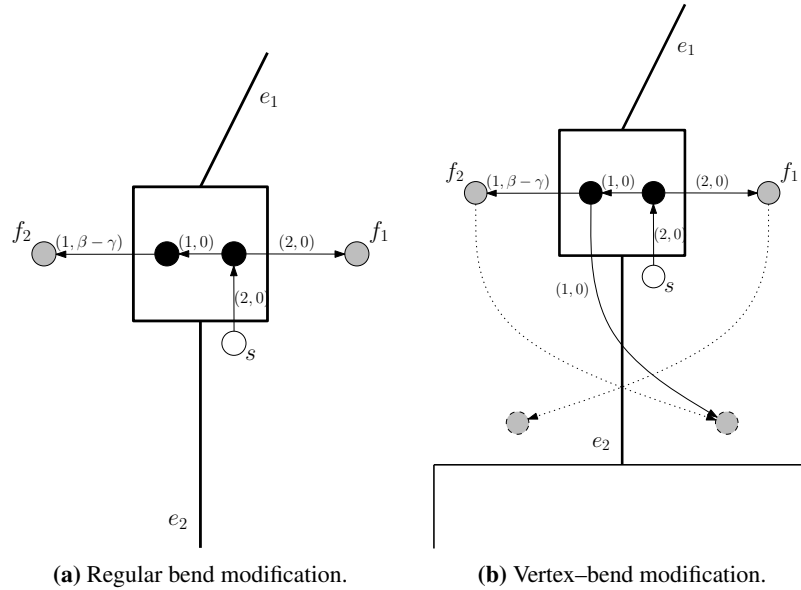


**Figure 3.1.:** Modification of a vertex-node in the Kandinsky network for SDL.

The second modification affects the modeling of bends in the Kandinsky network. A distinction has to be made between regular bends and vertex-bends. The modifications for both types of bends are depicted in Figure 3.2. For each bend, we add a bend-node to the model. For regular bends, both faces incident to the edge containing the bend are added a demand of 1, with cost 0 for the face which is on the concave side of the bend and  $\beta - \gamma$  for the face on the convex side, respectively, as shown in Figure 3.2(a). This bend model ensures that the bend is either contained in  $Q$  at cost 0 or removed at cost  $\beta - \gamma$ . For vertex-bends, we employ the same model with an extension: we add an additional arc connecting the bend-node to the face-node (of the concave face) of the vertex-node with capacity 1 and cost 0, see Figure 3.2(b). Also, for face-nodes of adjacent faces, we add arcs with cost of  $\beta + \gamma$  representing cost  $\beta + \gamma$  for each additional bend on this edge.

It is shown in (Brandes, Kaufmann, et al. 2002) that these modifications in the Kandinsky network allow an optimal solution for Problem 1 when solving the min-cost-flow network.

We will now describe how to modify the presented original SDL-approach in order to be able to handle a BPMN-graph  $G = (V, E)$ . Therefore, we use a construction that considers mapping *swimlane* and places nodes within the same swimlane in a surrounding *box*.

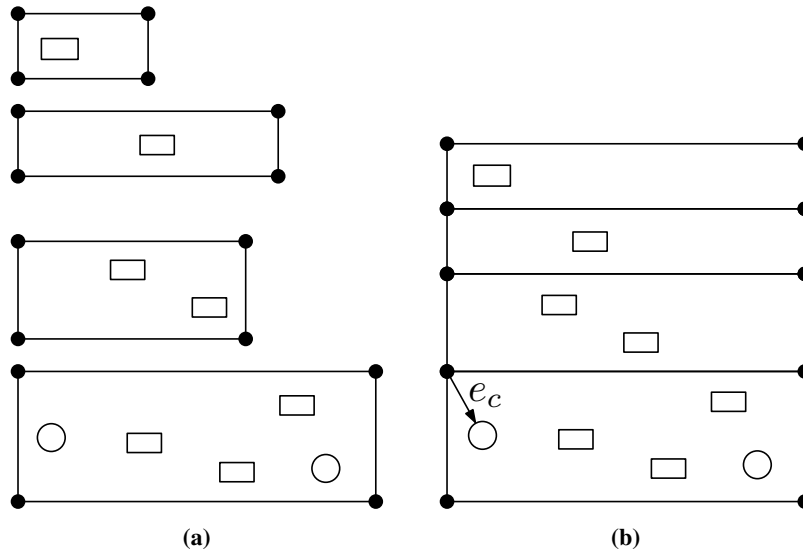


**Figure 3.2.:** Modifications for bends at regular and vertex-bends in the Kandinsky network for SDL. New arcs are drawn as solid lines and labeled with capacity and cost.

A box is constructed by inserting *structural edges*  $E_s$  which surround the box in two vertical and two horizontal segments. The segments are connected such that, together, they represent a rectangle. For each element of the kernel  $S$  of mapping *swimlane*, we construct one box. The nodes incident to edges of  $E_s$  are called *structural nodes*  $V_s$ . The boxes are then connected such that adjacent boxes share two nodes and one structural edge, see Figure 3.3 for the resulting graph  $G_s = (V \cup V_s, E \cup E_s)$ .

**Remark:** In the following, we assume that a user that gives the input sketch for SDL considers the principle of *swimlane*, that is: a new node  $n$  is added to the graph by drawing/dragging it into the box representing the swimlane of  $n$ . If the principle is not obeyed, we resolve this by translating nodes that are positioned in incorrect boxes in vertical direction to the corresponding box of their swimlane. Note that by translating nodes only vertically, we do not affect the (horizontal) flow of the BPMN-graph. Thus, *FLOW* is not violated.

Note that preserving the boxes of the swimlanes suffices to support *swimlane* in SDL. Therefore, we integrate the structural edges  $E_s$  into the Kandinsky network as follows: we remove the arcs from the network connecting the incident face-nodes of a structural edge  $e \in E_s$ . Remember that an additional bend for an edge  $e \in E$  has cost  $\beta + \gamma$  in the modified Kandinsky network. By not allowing bends on structural edges, we keep the boxes stable. Without bends, only parallel translation, vertically or horizontally, of structural nodes is possible in order to align boxes and box sizes. Since the SDL



**Figure 3.3.:** (a) Swimlanes and the corresponding boxes constructed of structural edges  $E_s$  and structural nodes  $V_s$  (solid circles). Nodes  $V$  are depicted as non-filled rectangles or circles (denoting events). Edges  $E$  are omitted. (b) The resulting graph  $G_s$ . Boxes are connected sharing structural edges and nodes. Also, connecting edge  $e_c$  is inserted. Again, edges  $E$  are omitted.

is topology preserving for a sketch  $\Sigma$ , see above (i), nodes remain in the assigned box and, therefore, mapping *swimlane* is further valid and supported.

Alternatively, the arcs between incident face-nodes could remain in the Kandinsky network, but with very high cost and, thus, rendering bends on structural edges unlikely. This alternative has the benefit that the min-cost-flow might be easier to solve due to more edges with high capacity that can be used in temporary solutions. However, the alternative does not guarantee that bends on edges  $E_s$  are prevented, although, bends are very unlikely if the bend penalty is set high enough. In Algorithm 1, the integration of our alternative SDL into the TSM framework is formulated. In (Siebenhaller 2009), the approach of adopting general constraints is extended further for implementing clusters or port- and side-constraints in the Kandinsky model of static layout approaches. However, (Siebenhaller 2009) does not apply the concept of SDL, neither on angles and bends, nor on cluster, partitions or port- and side-constraints. There, the idea of extending SDL, or interactive layout, is sketched as future work and is now realized for support of partitions in the present work.

The modification of the Kandinsky network is not yet complete because the sets  $V$  and  $V_s$  are not connected by any edge  $e \in E \cup E_s$ . Note that a BPMN-graph is always connected. Otherwise, there exists a node  $n \in V$  which can never be reached in any execution of the process. Then, the process is not a valid process. Therefore,

we assume  $G$  to be connected. Building a valid Kandinsky model in the orthogonalization phase within TSM, given two unconnected subgraphs as input would result in two unconnected layouts because TSM reduces crossings (in the planarization phase) by separating  $G_s$  into subgraphs  $G_{s1} = (V, E)$  and  $G_{s2} = (V_s, E_s)$ . Then, the boxes would not surround the nodes assigned to the corresponding swimlane and *swimlane* is not adhered in the layout. Therefore, we introduce a temporary *connecting edge*  $e_c$ . This edge connects a node  $n \in V$  with  $vertex\_type(n) = start\_event$  and a structural node  $n_s \in V_s$  of the box surrounding *swimlane*( $n$ ), see Figure 3.3. If  $n$  is not unique, a *start\_event* is chosen randomly. Since SDL is preserving the topology of the input sketch  $\Sigma$  and  $e_c$  renders  $G_s$  to be an admissible sketch for SDL,  $e_c$  ensures during a call of TSM with SDL that the boxes are stable. Therefore, mapping *swimlane* is adhered in the layout.

The connecting edge  $e_c$  has cost 0 for additional bends because it is removed after the solution of the min-cost-flow network and bends of  $e_c$  do not affect the final layout. Note that we choose  $e_c$  to connect a start node. Thus, we can ensure not to introduce additional crossings since *FLOW* requires start nodes to be aligned in the (left) beginning of swimlanes. By choosing  $n$  to be a start node of a BPMN-graph considering *FLOW*, no other node  $u \in V \cup V_s$  is placed between  $n$  and  $n_s$ . An example of the layout with SDL for BPMN in *BPMN-Layouter* with a small BPMN-graph  $G$  is depicted in Figure 3.4. There, the corresponding extended graph  $G_s$  with structural nodes, structural edges and connecting edge is also shown.

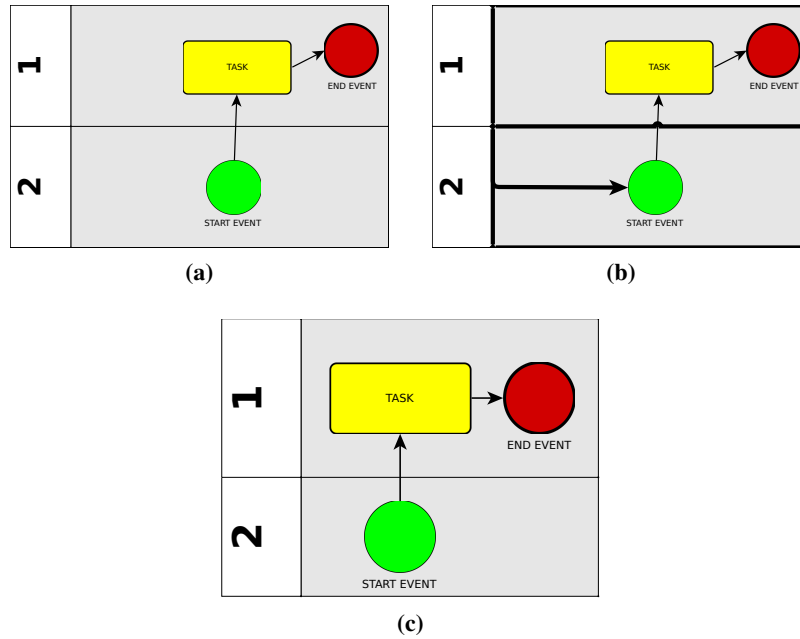
### 3.1.2. Application case: Divisions (Cuts)

We will now present an application case<sup>2</sup> for SDL for BPMN. The application aims at cutting large process models into submodels by subdividing the underlying embedding of the input diagram. In cases where process models become very complex and diagrams become large, it is desirable to divide the resulting diagram into smaller pieces, e.g. for printing a diagram on several sheets of standard size paper. The study in (H. Reijers and Mendling 2008) showed that finer grained submodels are preferred to complete models in large diagrams. In the following, we give an algorithm that divides BPMN-graphs automatically, subject to constraints, e.g. the size of sheets the BPMN-graph is to be printed on or the number of resulting pieces.

In the following definition, we introduce constraints for a division, e.g. constraints may comprise requirements concerning partitioning of a BPMN-graph or maximum area size for a BPMN-graph.

<sup>2</sup>Parts of this section are published in (Effinger, Siebenhaller, and Kaufmann 2009a).





**Figure 3.4.:** Example for a BPMN-graph and the inserted structural edges for preserving mapping *swimlane*. (a) Input sketch  $\Sigma$ . (b) Temporary structural edges and nodes are highlighted (fat solid lines and points on border). The connecting edge is connecting boxes and BPMN-graph (attached at start event). (c) Resulting layout after SDL for BPMN.

**Definition 8 (Division of BPMN-Graphs).**

A Division of a BPMN-graph  $G = (V, E)$  with given constraints  $C$  partitions  $G$  into sets of nodes  $V_1, \dots, V_k$  with  $k \geq 2$  such that  $V_i \cap V_j = \emptyset, \forall i \neq j$ . The subgraphs induced by  $V_i$  on  $G$  have to satisfy  $C$ . Edges of  $A = \{(v, w) \in E \mid v \in V_i, w \in V_j, i \neq j\}$  are called division connections.  $\square$

In BPMN-graphs, the aim of a division is to minimize  $|A|$  and to obtain subgraphs of nearly equivalent size in terms of nodes or area size. For a division, a route for a cut in a BPMN-graph  $G$  has to be found and the resulting subgraphs will then be used as an input sketch for SDL for BPMN.

In order to find appropriate routes, we introduce the idea of a *center band*. The *center band* is a rectangular space in the graph through which a cut runs, see Figure 3.5(a). Depending on user's preferences or given constraints  $C$ , the *center band* can be given as user input or it can be set automatically to a default fraction of the graph's size.

We now describe how to perform a horizontal cut, a vertical cut is performed analogously. For a horizontal cut, the width of the center band is set to the width of the bounding box surrounding the graph. The height of the center band is set to a predefined value  $y_o - y_u$  around the midpoint center  $y_m$  of the *center band*, see Figure 3.5(a).

**Algorithm 1:** SDL for BPMN

---

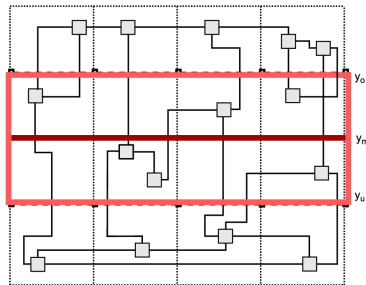
**Input:** Graph  $G(V, E)$

```

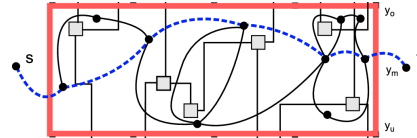
// construct boxes and store segments
1  $E_s \leftarrow$  segments of boxes;
2  $V_s \leftarrow$  points of boxes; // store shared vertices of boxes
3  $e_c \leftarrow$  insert connecting edge;
4  $G_s \leftarrow (V \cup V_s, E \cup E_s \cup e_c)$ ;
   // TSM phase 1: planarization
5  $G'_s \leftarrow TSM\_planarize(G_s)$ ;
6  $Q \leftarrow shape(G'_s)$ ;
   // TSM phase 2: orthogonalization
7  $N \leftarrow network(G'_s, Q, \alpha, \beta, \gamma)$  // create modified Kandinsky network
8  $N \leftarrow (Q, E_s, \infty)$  // set bend penalty for edges  $E_s$  to  $\infty$ 
9  $N \leftarrow (Q, e_c, 0)$  // set bend penalty for  $e_c$  to 0
10  $N' \leftarrow solve\ min - cost - flow(N)$ ;
11  $G''_s \leftarrow apply(N', Q)$ ;
   // TSM phase 3: compaction
12  $G_{comp} \leftarrow TSM\_compact(G''_s, Q)$ ;
13  $G_{final} \leftarrow (V_{comp} \setminus V_s, E_{comp} \setminus (E_s \cup e_c))$  // remove structural nodes and
   edges

```

---



(a) Placing a horizontal center band on the underlying graph. The red box denotes the center band. Swimlanes are depicted by dotted lines.



(b) Determining a division using the dual graph of the cut graph. Dashed blue edges denote a shortest path from  $s$  to  $t$  and induce a cut with a minimum number of split edges.

**Figure 3.5.:** Determination a route for a cut in a BPMN-graph.

The predefined value, if not given by the user, is preset with a default fraction of the graph's height. In our application cases, we found that a value that corresponds to 10% of the graph's height is a reasonable choice for finding a cut.

The core of the division algorithm is a dual graph routing inside the *center band* using Dijkstra's shortest path computation (Cormen et al. 2001). Therefore, the dual graph  $G'_D = (V'_D, E'_D)$  of the *cut graph*  $G' = (V' \subseteq V, E' \subseteq E)$  is constructed. The cut graph is the subgraph of  $G$  which is induced by the *center band*.

Since there is a one-to-one relation between edges of  $E'$  and edges of  $E'_D$ , it is easy to set higher weights for specific connecting objects of  $E'$  that should not become division connections. Those specific weights can be set by the user or they can be set to comply with BPMN specific semantic preferences, e.g. a data object assigned to a connecting object should not be cut. The connecting object weights are then passed to the corresponding edges of  $E'_D$ .

An example of a horizontal cut can be found in Figure 3.5(b). Analogously to horizontal cuts, a vertical cut is performed by using a vertical *center band*.

Performing a shortest path computation on the dual graph, we obtain the division connections for the original graph. Those edges have to be removed in order to split the graph. However, a connecting object removal causes information loss. Thus, we insert two replacement objects (links) for each such connecting object, analogously to the insertion of two links in (Effinger, Siebenhaller, and Kaufmann 2009a). After the computation of a cut, the resulting submodels are used as sketches for the SDL-approach for BPMN. Thus, mapping *swimlane* is preserved and angles of bends and edges of the original graph are used as sketch and the mental map of the user (induced by the original large graph) is kept. Eventually, the submodels can be identified to originate from the large original BPMN-model because the mental maps match.

## 3.2. Pattern-based BPMN-Layout

### 3.2.1. Motivation

The existing layout approaches for BPMN, i.e. (Effinger, Siebenhaller, and Kaufmann 2009a; Kitzmann et al. 2009) or SDL (Section 3.1), are based on the underlying structure of a BPMN diagram. The structure is considered to be a graph with nodes (BPMN elements) and edges connecting the nodes (in BPMN: sequence flow, conditional flow, default flow). However, the graph structure might be enriched with meta-data, e.g. BPMN-Graph, but known layout approaches for BPMN are not adapted for treating different types of the nodes differently. Therefore, BPMN semantics are not considered in previous layout approaches. For instance, when creating a layout for a BPMN diagram with the approach of Section 2.4, the final layout was not influenced by the distinction if an element was a *start event* or a *gateway*. For a user, this distinction can matter when 'reading' the layout of a BPMN model. The resulting layout might not look as expected because BPMN semantics induce distinct treatment of elements already within the layout algorithm.

In the following, we present three patterns that are designed to overcome the gap between graph structure and semantics of BPMN diagrams when creating a layout. Patterns are a common method to express abstract similarities in (process) models or graphs, e.g. *action patterns* are used to formalize semantic analysis of BPMN process models in (Smirnov et al. 2009) or for organization in large business process model repositories (Smirnov et al. 2010). The term 'layout patterns' is also used in (Maier and Minas 2010) for the expression of layout constraints which represent conditions for generic layout algorithms. There, the term has a similar understanding to the presented layout aesthetics in Section 2.3.

Our new patterns aim at a) reducing cluttering in diagrams, b) highlighting the logical structure of a BPMN diagram (induced by gateways) and c) accentuating the process flow. The patterns can be integrated as extensions to the SDL layout approach from Section 3.1.

This section<sup>3</sup> is organized as follows: In the next section, we will define our new patterns and provide details of the algorithms that are used for applying the patterns. Then, we will give an experimental evaluation of the new patterns combined with SDL in Section 3.2.3 and point out related work before summarizing.

### 3.2.2. Layout Patterns

Inspecting the list of standard layout aesthetics, see Section 2.3, one realizes that BPMN semantics in terms of element types, or mapping *vertex\_type*, are not taken into account in any of the aesthetics. The lack of support for BPMN element types is a gap between the graph structure and BPMN semantics. By introducing new layout patterns in this section, we attempt to take a first step to overcome this syntax-semantics gap in BPMN layout.

The patterns also affect different layout aesthetics. As shown later in the experiments, the patterns might alter the layout of an existing diagram layout and change measurements of aesthetics. However, the patterns should reduce the possibly negative effects of alterations in aesthetics and, at the same time, increase the support of BPMN semantics. For instance, a pattern might, on the one hand, enlarge the area size of the diagram and insert new crossings between flows, but on the other hand, bends might be removed and lengths of flows might be reduced. This shows the tradeoff between an aesthetically pleasing solution and consideration of semantics.

---

<sup>3</sup>Parts of this section were published in (Effinger 2011).

### Geometry pattern (GeoP)

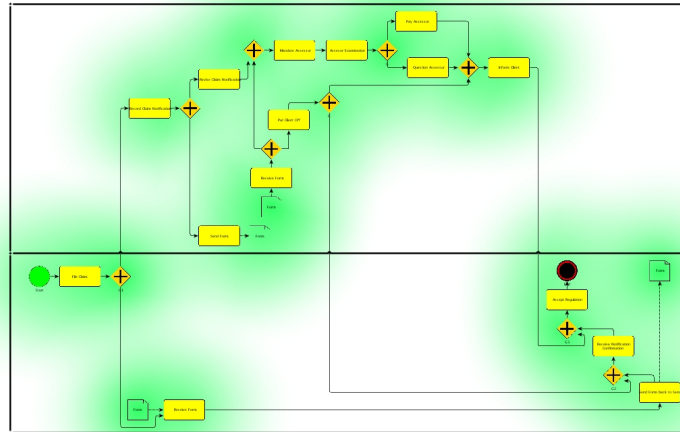
The first pattern that we present is the Geometry Pattern (GeoP). It aims at reducing visual cluttering. Cluttering describes the occurrence of many elements in a small amount of the diagram area (high element-density). For instance, around gateways with multiple connected elements, the cluttering is higher as the cluttering around a start event with a single outgoing sequence flow. Therefore, reducing the visual cluttering demands a reduction of visual density. At first sight, this pattern appears to be independent from BPMN semantics because it considers visual density of elements only, and not elements types. However, as described before, densities are induced by cluttered nodes which can be caused by parallel flows or complex logic dependencies. Since both, parallel flows and complex logic, are induced by gateways, high-density areas in a layout are more likely to be found around gateway elements and less likely around low-degree nodes, e.g. start/end events. Therefore, GeoP addresses BPMN semantics as well as the visual effect of cluttering.

As a first step, we will show how the density of a BPMN diagram is determined. A simplistic approach is depicted in Figure 3.6. There, around each element  $n$ , we draw a circle  $c_n$ . The circle  $c_n$  is drawn with gradient opacity such that with increasing distance from  $n$ , the transparency of  $c_n$  is augmented proportionally. In the center of  $n$ , opacity of  $c_n$  is 1 and decreases to 0 at a distance of radius  $r$  (in Figure 3.6,  $r$  is set to  $500px$  in a grid drawing). If two circles  $c_1$  and  $c_2$  overlap, the opacity of  $c_1$  and  $c_2$  is added up within the overlapping area. From Figure 3.6, one can easily inspect high-density areas which might benefit from less clutter. However, the gradients do not allow distinct values for density. Also, the gradient circles focus on the density centers, the rest of the diagram is not considered, although this information might be helpful for a better solution where unused area is then activated and is occupied.

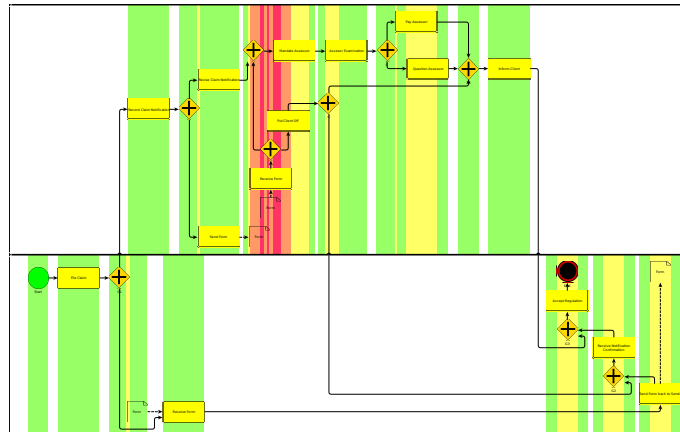
In GeoP, densities are determined as depicted in Figure 3.7. The calculation defines densities that are diametrically opposed to the flow in the model:

In a BPMN-Graph  $G = (V, E)$  with swimlanes  $S$ , we construct for each swimlane  $s \in S$  a set of events  $E_s$  which is given by the node positions of nodes  $N$  with  $swimlane(n) = s$ ,  $n \in V$ . For each node in  $N$ , we add two events to  $E$ : the x-coordinates of the left and right border of the drawn node. The left border is called an *increasing event* and the right border is a *decreasing event*. Then, we sort each set of events  $E_s$  in order of x-coordinates of the events and, let  $(e_1, \dots, e_k)$  denote the sorted set  $E_s$ , proceed for each swimlane with the first event  $e_1 \in E_s$ , store its coordinate in  $x_{pre}$  and initialize the density  $d_s$  of  $s$  with 1. For each subsequent event  $e_i \in \{e_2, \dots, e_k\}$ , we distinguish the following cases:

- If  $e_i$  is a decreasing event: let  $x_i$  denote the x-coordinate of event  $e_i$ . Then, for the segment from  $x_{pre}$  to  $x_i$ , we assign the density  $d_i$  given by the current value of  $d_s$ . Finalizing, we decrease  $d_s$  by 1 and set  $x_{pre}$  to  $x_i$ .



**Figure 3.6.:** Visualization of cluttering (densities) using circle gradients with decreasing opacity.

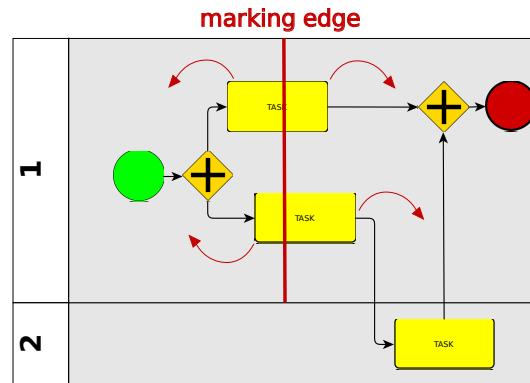


**Figure 3.7.:** Visualization of cluttering with blocks diametrically opposed to swimlane orientation (and FLOW).

- If  $e_i$  is an increasing event: let  $x_i$  denote the x-coordinate of event  $e_i$ . For the segment from  $x_{pre}$  to  $x_i$ , we assign the density  $d_i$  given by  $d_s$ . Finalizing, we increase  $d_s$  by 1 and set  $x_{pre}$  to  $x_i$ .

After the last event  $e_k$ ,  $d_s$  is 0 because there are no events in  $E_s$  that change  $d_s$  and  $d_s$  represents, at any event, the number of currently *active* elements. An element with left border  $x_l$  and right border  $x_r$  is active if  $x_l \leq x_{pre} \leq x_r$ . Now, to each segment in  $s$ , a density value  $d_s$  is assigned. In Figure 3.7, we visualized the density values of the segments by normalizing the density values to the color range from green to red (omitting segments with  $d_s = 0$ ).

After having detected dense areas, we now extend the SDL approach from Section 3.1 to provide support for GeoP. Therefore, we insert temporary edges  $E_t$  that mark these dense areas for the algorithm. An edge  $e \in E_t$  is inserted for each dense segment. Note that, in a typical application case of GeoP, only a subset of all segments is selected, e.g., the 10 segments with highest density are chosen. The important step is to add edge  $e$  as a structural edge to  $E$ . Remember that in SDL, structural edges are not allowed to bend (since a bend in a structural edge causes very high cost). The edges are inserted orthogonally to the swimlane orientation and are attached to the structural edges that represent the swimlanes, see Figure 3.8. They are aligned to the center of the segments. Note that edges  $E_t$  are allowed to overlap BPMN elements. Then, the algorithm of SDL extended by edges  $E_t$  aims at resolving the introduced overlaps by moving (sifting) nodes affected by marking edges.



**Figure 3.8.:** Schematic view of the algorithm for the Geometry Pattern. The marking edge is inserted into a dense segment and attached to the structural edges of the swimlanes. Red arrows depict the direction of possible sifting moves when resolving the overlaps.

This approach for GeoP is related to the *Sifting Algorithm (SA)* (Matuszewski, Schönfeld, and Molitor 1999; Rudell 1993). In general, SA tries to move one element at a time along an ordering of other elements until a goal function reaches a (local) minimum. In our case, we move all elements that are overlapped by a temporary marking edge. An element can be moved in parallel to its swimlane. Thus, SDL moves an element in either direction until the overlap with the marking edge is resolved. This spreads the original density center and the cluttering is reduced. Remember that SDL aims at keeping the sketch. Thus, the move distance of a node is kept as low as possible such that the created conflict, caused by the *marking edge/node*-overlap, is resolved. Note that other nodes might also be moved when resolving a marking edge/node-overlap. Thus, resolving overlaps in GeoP might change the overall area size (height and width) of diagram. The effect of GeoP on the area size is analyzed in the evaluation in Section 3.2.3.

### Gateway Pattern (GaP)

Our second new pattern, the Gateway Pattern (GaP), aims at highlighting the logical structure of a BPMN process model. The logic structure of a process model is induced by the combination of gateways. Gateways determine the process flow based on logical expressions that are evaluated when a gateway is passed. Evaluations of gateways may cause splits or joins of process flow(s). Since BPMN is not a block-structured notation language, but a graph-based notation language (Kopp et al. 2009), the underlying logic structure is not trivially induced by blocks. In (Dumas, García-Bañuelos, and Polyvyanyy 2010), the challenges of 'unraveling' (transforming) a non-well-structured process model to a well-structured model are described.

In general, if a process model is not well-structured, the determination of a pair of one opening split and one closing join, that represent a block, is not unique. We will now show how to find pairs that represent possible blocks in a non-well-structured model. In the following, we call such a pair *GaP-pair*.

If we can find such a GaP-pair  $p_G = (n_{split}, n_{join})$  that encloses a block structure, the pattern GaP requires that:

- no element of the block induced by  $p_G$  is placed *before* the opening split gateway  $n_{split}$  (with respect to the process model flow orientation), and
- no element of the block induced by  $p_G$  is placed *after* the closing join gateway  $n_{join}$ .

Thus, the goal of GaP is to highlight the semantic block structure by introducing an implicit block structure that is implied by the GaP-pair.

In most process models, we cannot find well-structured parts when analysing the gateways. Therefore, we apply a method that allows the construction of blocks that do not require well-structured processes. The details of this algorithm and the extension to SDL are described in the following.

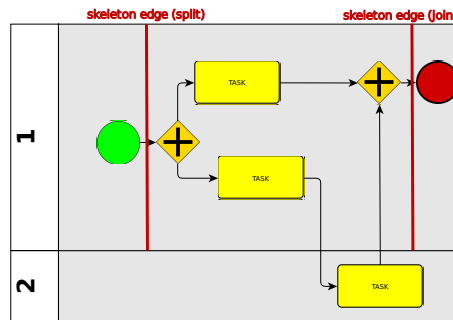
For our approach, we consider paths between gateways in a BPMN graph. The idea is to count the number of paths between two gateways and then construct blocks around a GaP-pair of two gateways that have the most paths together. Each path starts at an opening gateway and increases the path counter for each reachable closing gateway.

We define a GaP-pair as follows:

#### Definition 9 (GaP-Pair).

A split-gateway  $g_1$  and a join-gateway  $g_2$  form a **GaP-pair**, if, for all possible paths starting at  $g_1$ , the size of the subset of paths arriving at  $g_2$  is maximal among all reachable closing gateways. □





**Figure 3.9.:** Insertion of skeleton edges to SDL when applying GaP.

In other words, we count the number of paths  $c_p(G)$  arriving at any gateway  $g \in G$ , where an opening gateway  $g_1$  is the root node of each path and  $G$  the set of gateway nodes; then, for a closing gateway  $g_2$  where its path counter  $c_p(g_2)$  is maximal among all  $c_p(G)$ ,  $g_1$  and  $g_2$  form a GaP-pair.

The calculation is performed by using a variant of breadth-first-search (BFS) (Cormen et al. 2001): we start a BFS-run from every split-gateway (or other gateways that perform process flow split, e.g. complex gateways). Since BFS is able to handle cycles (by storing visited nodes) and employing our path counting method, we are able to find GaP-pairs in non-well-structured models.

If, for a split-gateway  $g_1$ , there are two join-gateways  $g_2$  and  $g'_2$  with the same path counter, the lower path distance between  $g_1$  and either  $g_2$  or  $g'_2$  is taken as criteria for determining the GaP-pair.

After finding GaP-Pairs, we insert surrounding temporary edges (skeleton edges) into the SDL model, analogously to GeoP, orthogonally to the flow orientation. Skeleton edges are inserted into the swimlane(s) of the gateways forming the GaP-pairs, see Figure 3.9. For every pair, one skeleton edge is introduced before the split-gateway and a second skeleton edge after the join-gateway. The skeleton edges delimit the surrounding box representing the block that is formed by the GaP-pair. The skeleton edges prevent nodes, which are contained in the block and which are part of a path between the surrounding GaP-pair  $p_G$ , to be moved to the outside of the block. If a node  $n$  would be moved to the outside of a block  $b$ , it would cause two crossings between one of the two skeleton edges of  $b$  and two edges  $e_n^1$  and  $e_n^2$  incident to  $n$ . Note that  $e_n^1$  and  $e_n^2$  exist because  $n$  was on the path between the opening and closing gateway of  $p_G$ . If it was not on this path, it would not have been part of the block  $b$ .

For model decomposition that was performed when analyzing the graph for GaP-pairs, a preferred decomposition for process models is often using SPQR-trees for a subsequent analysis of the structure or application of rules for abstraction (Polyvyanny, Smirnov, and Weske 2009) or for aggregation and hiding using process fragments

(Yongchareon et al. 2010). However, SPQR trees cannot handle the issue of loops and cycles in business process models as shown in (Polyvyanyy, García-Bañuelos, and Dumas 2010). The immense reduction in complexity when analyzing process models for strictly well-structured processes can be inspected in (G.-W. Kim et al. 2010).

In (Siebenhaller 2009), an aesthetics BIMODAL is proposed that can be credited to consider semantics. It requires incoming and outgoing edges that are incident to a node to be opposite, e.g. incoming edges are attached on the left side, outgoing edges are attached on the right side of the node. The effect of GaP on measurable layout aesthetics is analyzed in Section 3.2.3.

### Start-End-Pattern (SEP)

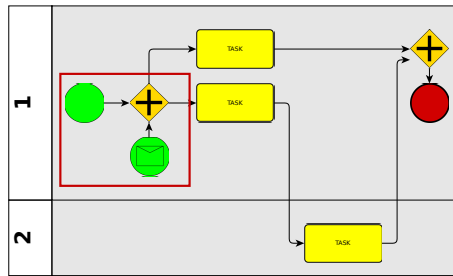
The third new pattern, the Start-End-Pattern (SEP), formalizes the compliance of placing start- and end-events in a swimlane strictly according to aesthetics FLOW. These elements should be placed such that they follow the orientation of the process flow (or 'reading' direction of the user). When SEP is activated, it guarantees that a start-event is set to the 'beginning' of its assigned swimlane, and an end-event is set to the 'end' of its swimlane. The move of a node across a major part of a diagram might introduce multiple crossings by incident edges and therefore affect CROSSING. Also, lengths of edges connected to moved events might increase severely. In order to possibly reduce the increase of edge length and prevent bends, we support two variants of SEP:

1. **Dynamic SEP:** Events affected by SEP are set to the border of the swimlanes but may move in parallel to the swimlane orientation in order to reduce unnecessary long edges or prevent bends, see Figure 3.10.
2. **Locked SEP:** All start- and end-events of a swimlane are aligned in the beginning/end of the swimlane, see Figure 3.11. The events are locked in a box surrounded by skeleton edges that guarantee that they do not move to the outer side of the box when performing a new layout.

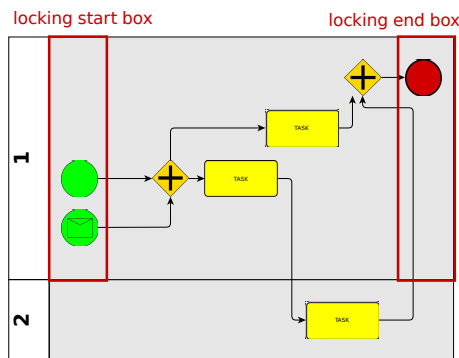
The second variant might be more appropriate for process models that have highly parallel process flow and, thus, several starting/terminating events that can be easily inspected when they are placed in an vertically aligned fashion at the swimlane border.

As mentioned in the two variants, we employ for SEP the idea of *skeleton edges* that was also used in GaP. The skeleton edges ensure the structure of the locking box and keep the nodes aligned in the beginning/end of the swimlanes when introduced as structural edges into the SDL model.

The movements of start- and end-elements into the locking boxes correspond to a shift in the  $x$ -coordinates such that the elements are centered in the boxes. Thus, the



**Figure 3.10.:** Dynamic Start-End-Pattern: Start- and End-events are moved to the borders of the swimlane but are allowed to optimize their position, e.g. see the node MESSAGE\_START\_EVENT that moved below the gateway in order to prevent a bend in the connecting edge.



**Figure 3.11.:** Locked Start-End-Pattern: Start- and End-events are kept static in boxes built by skeleton edges, compare Figure 3.10. Multiple events in one box are vertically aligned, compare with start events in the diagram.

elements are aligned in  $x$ . For the  $y$ -coordinates, we use a simple heuristic to optimize the vertical order of the elements: For each box  $b$ , we order the elements in  $b$  vertically according to the mean of the  $y$ -coordinates of all neighbours of  $n$ . The goal is to adapt the vertical position of the start- and end-elements to the positions of their successors or predecessors, respectively, in the graph. This heuristic attempts to keep changes in elements' positions, performed by the following SDL call, in the embedding small. Note that, for the computation of the vertical order, we use a padding between two vertically adjacent elements in order to prevent element overlaps.

A downside of the alignment of events is the requirement of a move inside the swimlane. This move might introduce multiple crossings when many edges have to be traversed. For SEP, we resolve this problem by storing all traversed edges and, after the SDL call, we initiate a post-processing stage that reroutes the stored edges if the crossing, that was caused by the move, persists. For the rerouting of edges, we use the concept of dual-graph-routing that was presented in Chapter 2.

### 3.2.3. Evaluation

We now present the results of an experimental evaluation of the presented BPMN layout patterns. The evaluation is designed to show effects of the patterns concerning aesthetics since formal aesthetics enable experiments that result in measurable numbers.

In order to facilitate comparisons with given layouts, we compare the results of SDL extended by our new layout patterns with the approach for BPMN-layout that was presented in Section 2.4. Therefore, we compute layouts with the given approach and then apply SDL with the corresponding pattern(s) of the test case and analyze differences in terms of formal aesthetics. The goal of the experimental evaluation is to find hints on the impact that the patterns might have towards layout aesthetics. Also, we like to show a quantitative analysis of the impact. If the impact of a pattern on an aesthetics is such that the aesthetics cannot be qualified as 'fulfilled', the pattern might not be appropriate since it violates the list of standard layout aesthetics from Section 2.3.

For the evaluation, a set of industrial business process models is processed. The set was created from tests in (Fahland et al. 2009). Sizes of the graphs are  $|V| \leq 145$  and  $|E| \leq 264$  with an average node degree of 2.2. For each process model, we test a total of 11 test cases which represent combinations of the new patterns. The combinations are depicted in Table 3.1. In Figure 3.12, we depict a BPMN process models and the resulting visualizations after the application of three different test cases.

Patterns \ Test Cases	Test Cases										
	1	2	3	4	5	6	7	8	9	10	11
GeoP	X	-	-	-	X	X	X	-	-	X	X
GaP	-	X	-	-	X	-	-	X	X	X	X
SEP (dynamic)	-	-	X	-	-	X	-	X	-	X	-
SEP (locked)	-	-	-	X	-	-	X	-	X	-	X

**Table 3.1.:** Combinations of patterns usage for experimental evaluation. 'X' denotes that the pattern was activated in the corresponding test case.

The implementation was integrated into a test-suite that is part of *BPMN-Layouter* which runs with Sun JAVA™ 1.5. Tests were performed on a Intel® Core™2 Quad CPU Q9300 with 2.50GHz and 3GB of RAM. The operating system is *Ubuntu 10.04 LTS - Lucid Lynx*.

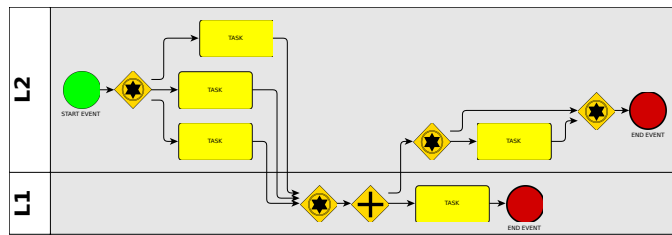
Since GeoP requires a predefined number which states how many of the most dense centers should be considered, we perform, for every GeoP test case run, 3 distinct runs

with a randomly chosen number  $1 < k \leq 10$  where  $k$  determines the number of selected dense centers. The dense centers are selected in decreasing density order such that the centers with the highest density are always chosen. Higher numbers of chosen dense centers are not appropriate because this leads to a wide-spread and lengthy diagram due to many sifting operations (negative effect on aesthetics AREA).

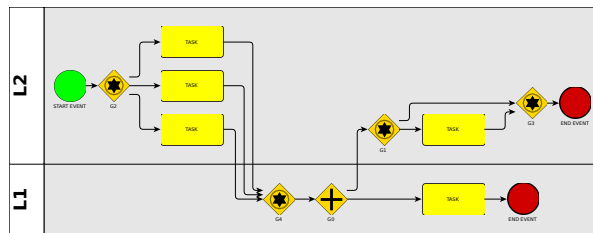
The results of the experimental evaluation are depicted in Figures 3.13 - 3.16. The first fact that comes to mind when inspecting the diagrams is the high running of time of test cases using GaP, see Figure 3.13. This is a hint to the hidden complexity that stems from the calculation of number of paths in a graph. In larger process models one might switch to a high-performance implementation of the pre-processing step to find all paths that employs a smarter data-structure for storing paths that overlap, e.g. a sorted set of path lists, and to exploit the topological ordering of elements in the graph in the sorting.

Test cases using GeoP (cases 1,5,6,7,10,11) show a positive behaviour of aesthetics, the changes are very low except for a tendency towards higher edge lengths (cases 5,10), see Figure 3.16. Number of bends and crossings are not affected by a high deviation. Test cases analysing GaP (2,5,8,9,10,11) show higher deflection, especially case 2 where bends are reduced but area size and edge lengths are increased. The difference between SEP (locked) and SEP (dynamic) becomes obvious when comparing cases 3 and 4: SEP (locked) produces clearly more crossings and bends than SEP (dynamic), however, both approaches need slightly more area space, see Figures 3.14 and 3.15. When comparing test cases with a combination of two patterns (test cases 5,6,7,8,9) or three patterns (test cases 10,11), the maximum and minimum deflection decrease, i.e., see Figure 3.14 for crossings and bends. This indicates that patterns affect aesthetics in an orthogonal way, e.g. a pattern A affects aesthetics AA positively whereas pattern B affects AA negatively, in sum, A and B end up to neutralize the measured numbers of AA. Note that this does not mean that the layout is left unchanged since A and B still must be ensured in the final layout.

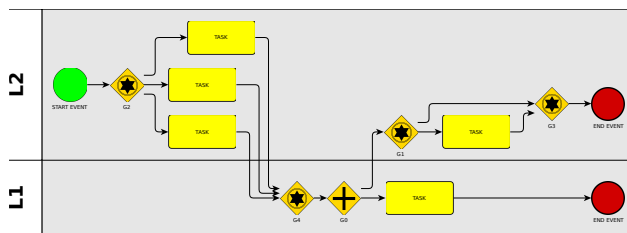
In summary, the impact of the new patterns on the measured aesthetics is not significantly high and does not turn a diagram layout unreadable (e.g. very flat diagrams or diagrams with a high fraction of unused area). This allows to state that, from an experimental point of view, the new patterns do not render accepted aesthetics invalid or infeasible. Therefore, the patterns may be considered an improvement for BPMN layouts. Moreover, including semantics in layout approach is a preferable method to enrich the previously only structure-based layouts. Layouts with support of semantics are more expressive since they contain implicit information that might not be expressed textually without overloading the diagram.



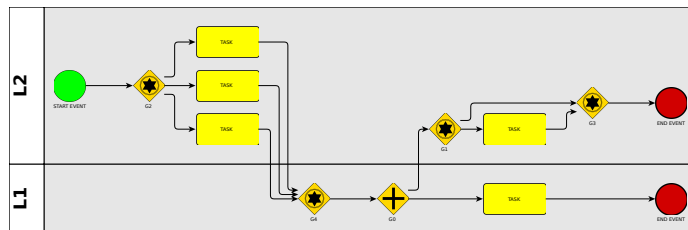
(a) Input example.



(b) Test case 1: GeoP.

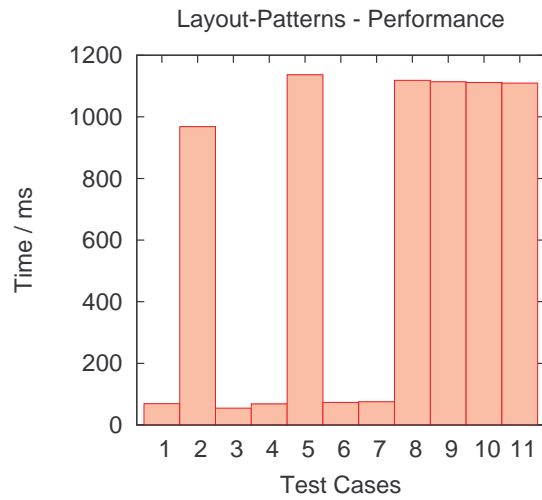


(c) Test case 4: SEP locked.

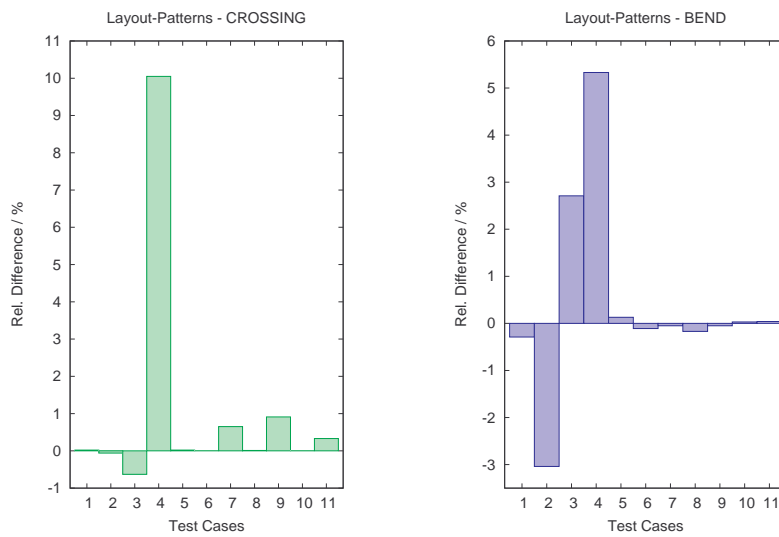


(d) Test case 11: GeoP, GaP and SEP locked.

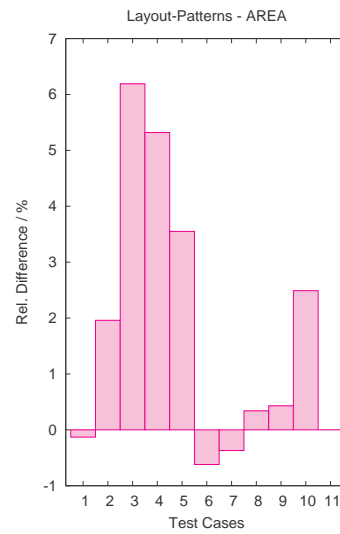
**Figure 3.12.:** Example for the application of the layout patterns. (a) The process model from Figure 2.16 is taken as input. The layout is computed by the static layout approach from Section 2.4. (b) Test case 1 (GeoP) is applied. Note that the tasks are now vertically aligned due to the move of the tasks to the same side of the marking edge. (c) Test case 4 (SEP locked) is applied: the start- and end-events are attached to the left/right borders. (d) Test case 11 (GeoP, GaP and SEP locked) is applied. The result is a combination of the layouts in (b) and (c) because GaP has no effect (the gateways already are in the optimal positions).



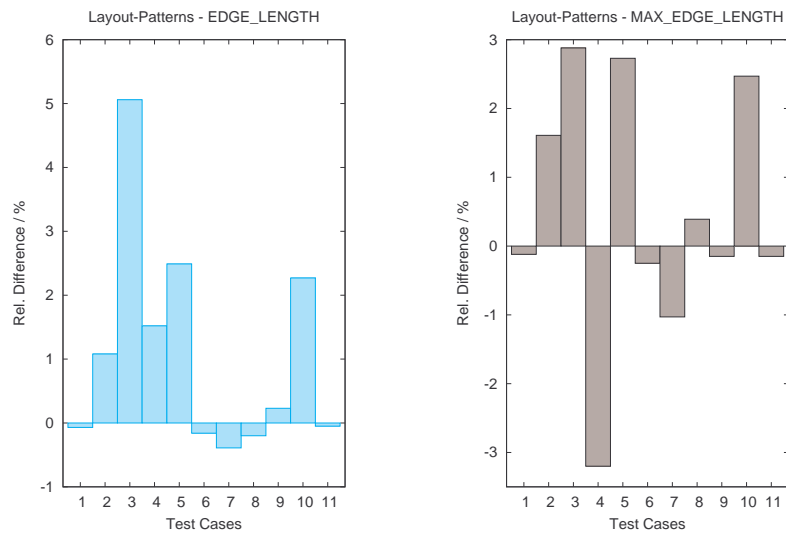
**Figure 3.13.:** Performance of the layout patterns. For each test case, total layout time including call of SDL with pattern extensions is measured. Note that the maximum running time does not exceed 1.2 seconds. Test cases 8-11 only differ marginally.



**Figure 3.14.:** Effects of layout patterns on the number of crossings and bends in the layouts. Relative differences to the input layouts that are computed with the approach from Section 2.4 are given.



**Figure 3.15.:** Effects of layout patterns on area size. Note that the maximum area difference is only 6% which is also due to the limited number of selected dense centers when GeoP is activated.



**Figure 3.16.:** Effects of layout patterns on the sum of edge lengths and the maximum edge length.



### 3.2.4. Summary

In this section, we introduced new layout patterns for BPMN diagrams. The layout patterns allow to include BPMN semantics in layout algorithms for BPMN process models. The layout patterns take into account standard aesthetics of BPMN diagrams. The patterns address the following issues in BPMN diagrams:

- Cluttering of nodes in diagrams (Geometry Pattern, GeoP),
- Perception of the logical structure of a BPMN diagram (Gateway Pattern, GaP),
- Accentuation of the process flow at start events and process' termination in end events (Start–End–Pattern, SEP).

We also presented algorithmic details for the automatic execution of the layout patterns integrated into the SDL algorithm. The algorithms are evaluated in experiments that analyze the impact of the pattern on a list of common layout aesthetics for BPMN.

## 3.3. A Layout Approach for BPEL–workflows

In this section<sup>4</sup>, we present an approach for producing layouts of workflows that are based on the *Business Process Execution Language* (BPEL) (Alves et al. 2007). BPEL is a verbose and hierarchical workflow language. A BPEL model consists of execution paths that can be nested, alternative or concurrent; any combination of these attributes at a time is also possible.

We formalize the BPEL specifics (aesthetics) that a layout algorithm has to fulfill as a set of layout criteria. The set is not given by any standardization document and has to be manually created as a step towards a BPEL layout approach. The aesthetics aim to enhance readability and to comply with commonly adapted styles for BPEL models and more generally for workflow diagrams or flow charts (Ambler 2005).

Our approach enhances the Sugiyama algorithm (Sugiyama, Tagawa, and Toda 1981), see Section 2.1.1, by modifying and extending the steps of the original algorithm. The new algorithm allows to set fixed paths, called *pathways*, in a workflow that correspond to *parallel execution paths* and, therefore, should be drawn in parallel. This goal requires modifications of the original Sugiyama algorithm since pathways consist of a sequence of edges and these edges create additional dependencies between vertices on subsequent layers in the model of the Sugiyama algorithm. Moreover, nested structures, i.e., clusters, in the workflow are supported in our approach such that the result

---

<sup>4</sup>Parts of this section were joint work with Benjamin Albrecht, Markus Held, Michael Kaufmann and Stephan Kottler. Parts of the results were published in (Albrecht, Effinger, Held, Kaufmann, and Kottler 2009) and (Albrecht, Effinger, Held, and Kaufmann 2010).

of a layout clearly highlights the nesting of BPEL elements. In addition, our approach uses different graphical representations for distinct types of BPEL elements. The representations also allow for enhancing the visualization of existing parallel structures in a BPEL model.

This section is structured as follows: in Section 3.3.1, a short introduction to the specifics of BPEL is given, followed by the determination of the set of aesthetics for BPEL layout. After pointing to related work in Section 3.3.2, the new algorithm is presented in Section 3.3.3 including a time complexity analysis of the approach.

### 3.3.1. Preliminaries

#### The Business Process Execution Language (BPEL)

BPEL is an XML-based language for orchestration of web services and has been standardized by the Object Management Group (OMG). BPEL has been derived from two earlier workflow execution languages, i.e., the block-oriented XLANG language by Microsoft, and the graph-oriented Web Services Flow Language by IBM. In the following, we will shortly describe the properties of BPEL, as defined in the WS-BPEL 2.0 OASIS standard from 2007 (Alves et al. 2007).

BPEL is an imperative and structured programming language which contains a mixture of block-oriented and graph-oriented elements. In contrast to high-level imperative languages, BPEL does neither encompass any concepts of modules, libraries or classes. The only way to decompose a BPEL workflow is to encapsulate functionality in another workflow. BPEL does not contain any concept of functions or procedures. We now give a short summary of element types in a BPEL model, as given in (Held and Blochinger 2009):

*Activities:* The BPEL language distinguishes between *basic activities* and *structured activities*. Atomic tasks are modeled as basic activities which are treated by the workflow as "black boxes". Variable values can be changed using the Assign activity. It may contain an arbitrary number of assignment operations, expressed as copy elements. The activities receive and reply are used to model communication with a client of the BPEL process, while Invoke calls an operation on a Web Service. All communication partners assume roles defined in *partner links*. Input and output data are passed via variable references. Most control structures in BPEL are expressed as *structured activities*. In contrast to basic activities, structured activities contain child activities. Therefore, BPEL workflow models can be nested.

*Sequences and Scopes:* The structured activity Sequence is used to express the subsequent execution of an ordered set of activities. The functionality of Sequence resembles the concept of bracket–enclosed blocks in C–like programming languages. However, Sequence–activities cannot be used for local variable declarations. The BPEL language provides the structured activity Scope to deal with local declarations of variables and event handlers. Scope contains one child activity.

*Conditionals, Events and Loops:* The If–activity contains a set of child elements which are bound to conditions. Either the first child activity, whose condition evaluates to *true*, or a default activity, or no child activity is executed. The Pick–activity resembles the If–activity, but depends on external events rather than conditions. Pick can be used to wait for the occurrence of one out of a set of messages or timeout events. Loops are declared with the activities While, RepeatUntil, and ForEach.

*Parallelism:* Concurrency can be modeled using the structured activities Flow and ForEach. Flow allows the definition of directed acyclic graphs (DAG) of activities, while ForEach loops may be marked as ‘parallel’. Links between activities are always declared inside a Flow activity, and must not form a cycle. Links declared inside a Flow may cross the boundaries of structured activities within the Flow.

The aforementioned BPEL elements all have specific requirements if developing a layout algorithm. However, for each element, its respective requirements have to be fulfilled by the algorithm. In the following, we present a representation of the BPEL–specific requirements by introducing layout aesthetics for BPEL models.

### **Layout aesthetics for BPEL models**

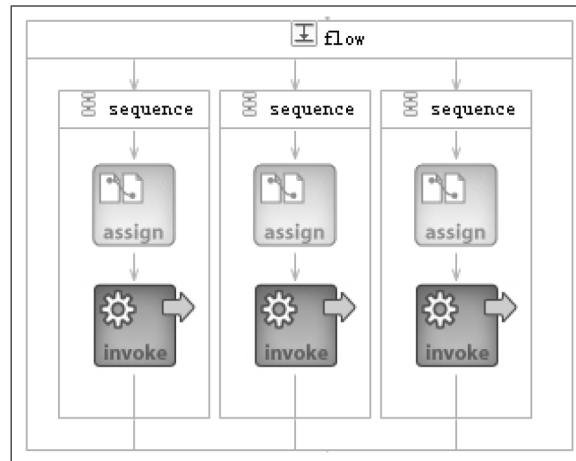
Generally, a BPEL model represents a workflow. Thus, aesthetics *FLOW* is the most important aspect. Often a workflow contains some central paths which are most relevant to understand and follow the whole process. Thus, the algorithm supports highlighting of such pathways, formalized in the following aesthetics:

- Maximize the number of fixed pathways (PATHWAY).

Also, for nested or symmetric structures in BPEL (e.g. Flow, ForEach), we define the following aesthetics that have to be supported by the algorithm:

- Structures may be nested related to the concept of graph clustering (CLUSTER).
- Structures may have a symmetric structure that is to be visualized (SYMMETRY).

Following typical layouts of workflows (Diguglielmo et al. 2002), we state all supported aesthetics and define the following set of aesthetics as requirements for a BPEL layout approach:



**Figure 3.17.:** Image of a layout by a simple approach for BPEL layout. Source: (Zhao, Han, and Y. Huang 2009). Links are not supported, see Figure 3.27 for comparison.

*FLOW*, *OVERLAP*, *PATHWAY*, *CLUSTER*, *CROSSING*, *BEND*, *EDGE\_LENGTH*, *ORTHOGONAL*.

Moreover, aesthetics *SYMMETRY* is applied when the process consists of parallel structures with child activities, e.g. *Flow* and *ForEach*.

Analogously to aesthetics for BPMN models, see Section 2.3, a precedence of the aesthetics is only available in parts if we consider user studies, e.g. (Purchase, Cohen, and James 1997). Therefore, we assume that *CROSSING* is the most important aesthetics, in accordance with (Purchase 1997) and the focus of Sugiyama’s algorithm on crossing minimization.

### 3.3.2. Related Approaches

In (Zhao, Han, and Y. Huang 2009), a simple approach for automatic layout of BPEL models is proposed. However, this approach is limited in several ways: first, the layout algorithm is limited to a simple alignment along the x- and y-axis based on a tree structure. Second, links between elements in the BPEL are not considered which allows the artificial simplification that the underlying BPEL model is represented as a tree in the algorithm, see Figure 3.17. This assumption does not correspond to real-world BPEL models and avoids the more realistic challenges of BPEL layout on graph structures. These limitations render the approach to be insufficient for the integration into applications, e.g. BPEL modeling tools.

ActiveBPEL® Designer by Active Endpoints is a wide-spread visual modeling tool for BPEL processes. However, the vendor stopped support of ActiveBPEL® Designer and

the publicly available open-source ActiveBPEL® Engine<sup>5</sup> does not contain a visual modeling tool. Testing an available former version (3.0.3) of ActiveBPEL® Designer, we found the layout results to be rather poor, see Figure 3.18: overlapping of elements occurred and link routing over elements was not prevented.

The Eclipse BPEL Designer (EBD) is a plug-in for the Eclipse IDE<sup>6</sup> provided by the BPEL Team of the Eclipse Foundation. It is a GEF-based (Graphical Editing Framework) editor<sup>7</sup> that provides graphical means to design BPEL processes. The latest version 1.0.0, a major release, dates from June 2012. The results of EBD can be inspected in Figure 3.19. Overlapping can occur in complex cases if not manually prevented and routing of links is poor since it overlaps with other elements and structural graphics. In Figure 3.20, the identical process is depicted using our approach. We can observe that links do not overlap and the hierarchical structure becomes more easily perceptible to the user. Also, our approach optimizes the ports of a link to its partner by using a technique that extends the Sugiyama algorithm and is presented in (Siebenhaller 2009). Remember that a port is the coordinate offset at a BPEL element at which the link is connected to. The optimization of ports for the links prevents unnecessary crossings during the link insertion.

### Related work on layout of workflow and processes

There are several approaches tackling the field of business processes that are related to BPEL: In (Rinderle et al. 2006), business process visualization is proposed using Sugiyama algorithm and force–scan, based on (Yang et al. 2004). The approach supports typed nodes, but cannot handle nesting, hierarchical structures and it is limited to series–parallel graphs that require acyclic input graphs which imposes a major limitation considering general (cyclic) workflow graphs. In (Six and Tollis 2002), a linear-time algorithm for processes with partitions is proposed. Business process graphs in a more general notation are handled in (Wittenburg and Weitzman 1997). However, both approaches do not take into account that there exist distinct types of nodes which require different handling in the layout. Additionally, nesting is not supported.

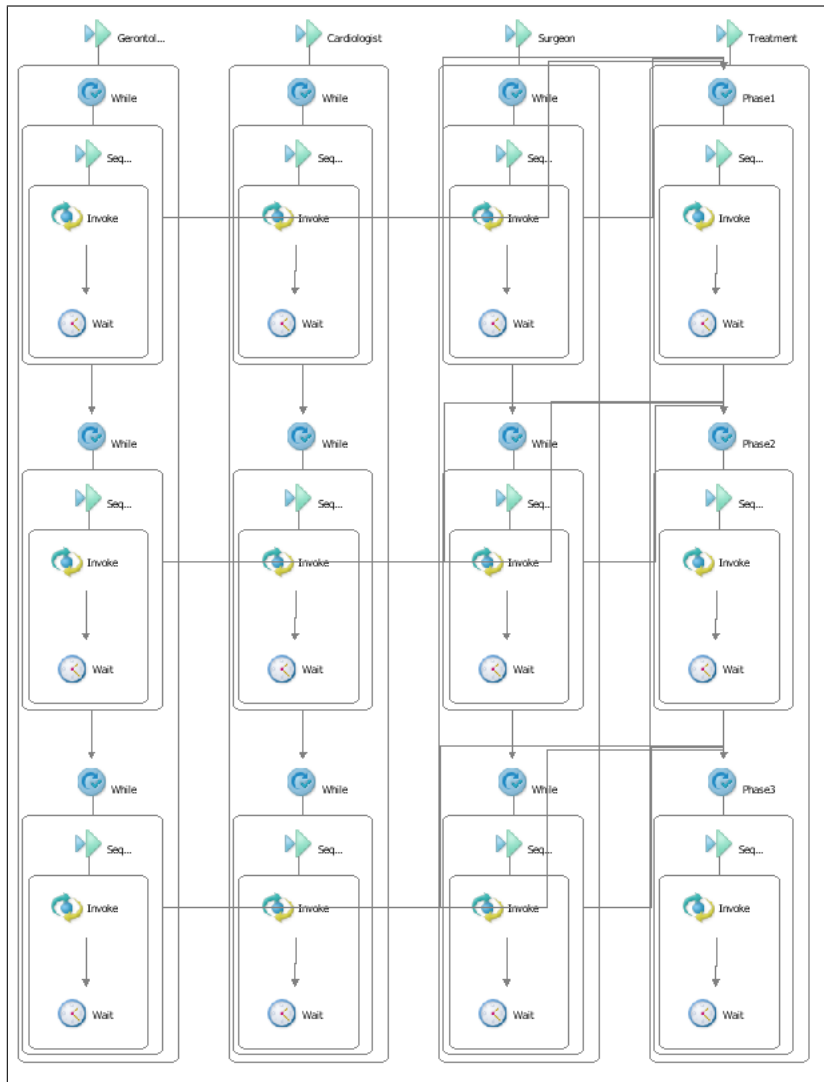
In (Diguglielmo et al. 2002), graph layout for workflow is presented within the commercial solution *ILOG JViews* that is now integrated into the software department of the company of *IBM*. The solution also features incremental drawings and grouping. However, details of the algorithm are not given and BPEL specific elements cannot be handled.

---

<sup>5</sup>see <http://www.activevos.com/community-open-source.php>, 2012–09–30.

<sup>6</sup>see <http://www.eclipse.org/bpel/>, 2012–09–30.

<sup>7</sup>see <http://www.eclipse.org/gef/>, 2012–09–30.

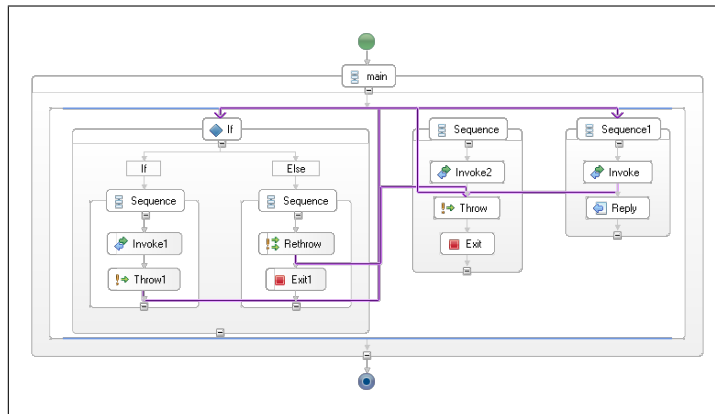


**Figure 3.18.:** A workflow layout by the ActiveBPEL® Designer (version 3.0.3) of Active Endpoints. Note that edges are routed on top of node labels and on top of other edges.

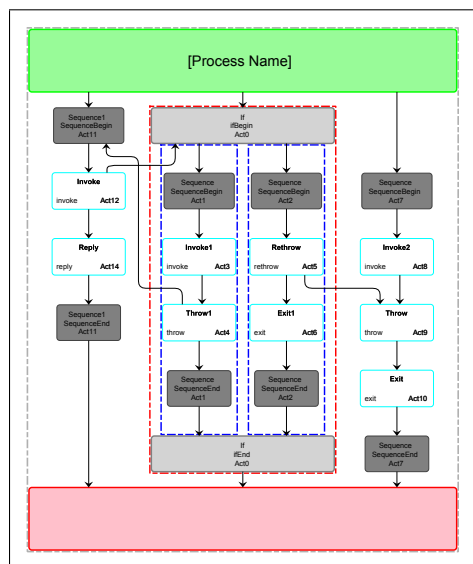
### 3.3.3. Layout Algorithm for BPEL-workflows

A visualization of BPEL processes with its flows and activities suggests a layered drawing technique. Thus, our layout approach is based on modified phases of the Sugiyama algorithm (Sugiyama, Tagawa, and Toda 1981), see Section 2.

For our purposes, we need the following definitions for an underlying graph  $G = (V, E)$  of a BPEL model. For the layout it is important to increase the readability of *pathways* denoted by the set of pathways  $\mathcal{P}$ .



**Figure 3.19.:** A workflow layout by the Eclipse BPEL Designer (version 0.4.0). Note that edge routing causes unnecessary crossings and edges are routed on top of other edges and nodes.



**Figure 3.20.:** The same workflow as represented in Figure 3.19, constructed with our layout algorithm.

**Definition 10.**

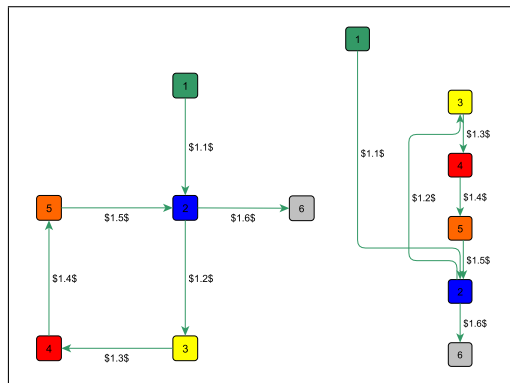
A pathway  $P \in \mathcal{P}$  is a sorted set of nodes  $(v_1, \dots, v_k) \in V$  which represent a path in  $G$  from  $v_1$  to  $v_k$  and which must be vertically aligned in the resulting layout. A node  $v \in V$ , that is part of at least one pathway  $P$ , is called pathway node, an edge  $e \in E$ , contained in at least one pathway  $P$ , is called pathway edge respectively.  $\square$

**Definition 11.**

A node  $v \in V \setminus \bigcup_{i=1, \dots, k} P_i$ , i.e.,  $v$  is not part of a pathway  $P_i \in \mathcal{P}$ , is called standard node. Analogously, an edge  $e \in E \setminus \bigcup_{i=1, \dots, k} \{(q, r) | q, r \in P_i\}$ ,  $e$  is not contained in a pathway, is called standard edge.  $\square$

Drawing a path vertically highlights the containing nodes in a way such that the reader can easily follow it (aesthetics FLOW and PATHWAY). Thus, if possible, each pathway should be drawn straight from top to bottom without any bends. To achieve this goal, we perform the following extensions to the phases of the Sugiyama algorithm:

*Extensions to the Cycle Removal Phase:* Regarding cycles in the graph, we consider the special case of a cycle which consists of *pathway edges* only. In this case, we choose the cycle edge  $e_{\min} = (v_l, v_m)$  which is contained in the least pathways  $\mathcal{P}' \subseteq \mathcal{P}$ . After that each pathway  $P = (v_1, \dots, v_l, v_m, \dots, v_n)$  of  $\mathcal{P}'$  is divided into two paths  $P_1 = (v_1, \dots, v_l)$  and  $P_2 = (v_m, \dots, v_n)$ . Finally,  $e_{\min}$  is reversed. Each edge which is reversed during this step is stored in  $\mathcal{S}$ . At the end of the algorithm, each edge in  $\mathcal{S}$  is restored to its original direction. A simple example of this extension is given in Figure 3.21. The algorithm is described in Algorithm 2.



**Figure 3.21.:** (left) Given a pathway  $P = (1, 2, 3, 4, 5, 2, 6)$  containing a cycle  $(2, 3, 4, 5, 2)$  consisting of pathway edges only. (right) The pathway  $P$  can not be drawn straight from top to bottom. Thus, it is split up into two pathways  $P_1 = (1, 2)$  and  $P_2 = (3, 4, 5, 2, 6)$ . Since node 2 belongs to  $P_1$  as well as to  $P_2$  it can either be drawn below node 1 or node 5. The labels on the edges denote the pathway number followed by the segment number of this edge in the path.

*Extensions to the Layer Assignment Phase:* To ensure that each pathway is drawn from top to bottom further constraints are necessary: Given a pathway  $P = (v_1, \dots, v_n)$ ,



**Algorithm 2:** Extended cycle removal phase

---

```

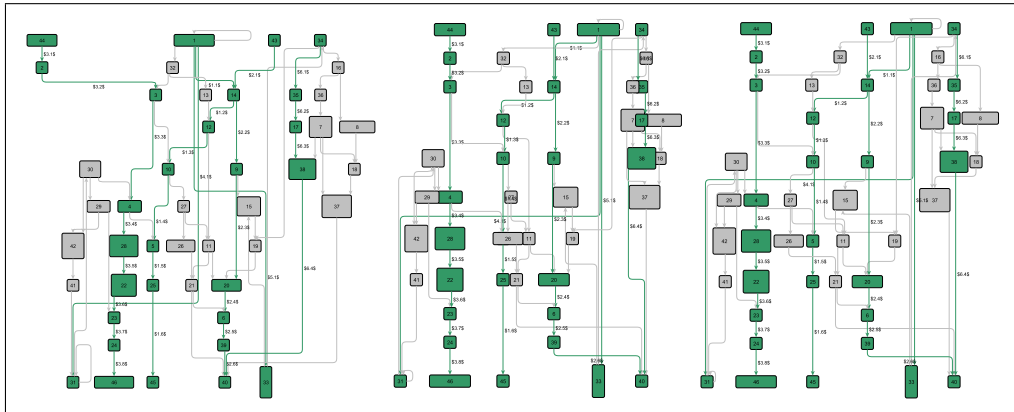
1  $S \leftarrow \emptyset$ ; // set of the reversed edges
2 while  $G$  contains cycle  $C$  do
3    $E' \leftarrow$  set of all cycle edges in  $C$ ;
4   if  $E'$  contains a standard edge then
5      $e \leftarrow$  a standard edge of  $E'$  contained in most cycles;
6     reverse  $e$  in  $G$ ;
7      $S \leftarrow S \cup \{e\}$ ;
8   else
9      $e_{\min} \leftarrow$  pathway edge of  $E'$  which is contained in the least pathways in  $\mathcal{P}'$ ;
10    foreach  $P \in \mathcal{P}'$  do // split each path of  $\mathcal{P}'$ 
11      given  $P = (v_1, \dots, v_l, v_m, \dots, v_n)$  and  $e_{\min} = (v_l, v_m)$ ;
12       $P_1 = (v_1, \dots, v_l)$ ;
13       $P_2 = (v_m, \dots, v_n)$ ;
14       $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{P\}) \cup \{P_1\} \cup \{P_2\}$ ;
15      reverse  $e_{\min}$  in  $G$ ;
16       $S \leftarrow S \cup \{e_{\min}\}$ ;

```

---

for each adjacent pair of pathway nodes  $v_i$  and  $v_{i+1}$ , we demand that  $v_i$  is assigned to a layer above  $v_{i+1}$ . Since we removed all cycles consisting only of pathway edges in the former step, this can always be fulfilled.

*Extensions to the Computation of Horizontal Coordinates:* For the computation of the horizontal coordinates, the standard algorithm is applied in a first step followed by a post-processing step: For each pathway  $P \in \mathcal{P}$ , the barycenter  $b$  of the  $x$ -coordinates of all pathway nodes in  $P$  is computed. The horizontal coordinates of nodes contained in exactly one path  $P$  are set to  $b$ . To avoid overlapping pathway nodes, the set of all barycenters  $\mathcal{B}$  is processed such that, for each  $b_i, b_j \in \mathcal{B} : |b_i - b_j| \geq d_{\min}$  holds, where  $d_{\min}$  is a constant denoting the minimal node distance. For nodes contained in pathways  $P_1, \dots, P_k, k \geq 2$ , the  $x$ -coordinate is set to the barycenter of  $b_1, \dots, b_k$ . Finally, in order to avoid overlapping nodes, the distance between each standard node  $v$  and its neighbour nodes in the same layer of the Sugiyama model is tested. If the distance to one of these nodes is smaller than  $d_{\min}$ , a new  $x$ -coordinate  $x(v)$  for  $v$  is computed: According to the location of its parent nodes  $\mathcal{S}_p$ , the shifting direction  $d$  is determined at first (see Algorithm 4). Then,  $v$  is shifted towards this direction  $d$  to the  $x$ -coordinate  $x(v)$  such that its distance to its previous neighbour node is exactly  $d_{\min}$  and, subsequently, the distance to the its new neighbour-nodes,  $v_l$  and  $v_r$ , is tested (see Algorithms 3 and 5). If the distance to a new neighbour node  $v_j$  is smaller than  $d_{\min}$  and  $v_j$  is a standard node a further shifting



**Figure 3.22.:** An example showing the different steps of the computation of the horizontal coordinates. (left) Input to the extended phase of computation of horizontal coordinates; pathways are highlighted in green. (middle) All pathway nodes are set to the barycenter of the corresponding pathway. (right) All standard nodes are shifted such that every node in the graph adheres to the minimal node distance.

step to  $v_j$  towards direction  $d$  is applied; otherwise, if  $v_j$  is a pathway node,  $v$  is shifted again (see Algorithm 6). Hence, for each standard node exactly one shifting phase is performed. An example for this extension phase is depicted in Figure 3.22.

### BPEL-specific steps in our algorithm

In addition to the modifications of the phases in the Sugiyama algorithm, our approach consists of further steps that adapt the layout to BPEL specifics. The activities of a BPEL model form a nested structure. We assume one top level Flow-activity, which contains all other activities. Resolving this nested structure we get a graph  $G$  in which paths split up and merge again. For generating an 'adequate' layout, pathways are embedded into  $G$ . Here, 'adequate' means that the produced layout should highlight execution paths by drawing them in parallel (SYMMETRY). In order to add further information to the layout, modifications are applied to the graphical representation of the BPEL elements in the layout.

In a BPEL model, we distinguish between two different types of edges:

- *Pathway edges* are induced by the activity structure, that is designed by the user, and are created while resolving the nested structure of the workflow.
- *Transverse edges* are not induced by the activity structure, but they can additionally be added by the human workflow designer to connect any kind of activities, e.g. an Invoke call to a non-incident BPEL element can be represented by a transverse edge.

An example for the distinction between the types of edges can be inspected in Figure 3.23.

**Algorithm 3:** Extended phase for computation of horizontal coordinates

---

```

1 apply standard algorithm; // see (Brandes and Köpf 2001).
2 foreach  $P \in \mathcal{P}$  do
3    $c_{average} \leftarrow (\sum_{v \in P} x(v)) / |P|$ ;
4   foreach  $v \in P_i$  do
5     // set  $x$ -coordinate of  $v$ 
6      $x(v) \leftarrow c_{average}$ ;
6 foreach Layer  $L \in G$  do
7   foreach node  $v \in L$  do
8      $v_r \leftarrow$  right neighbour of  $v$ ;
9     if  $\text{DISTANCE}(v, v_r) < d_{min}$  then
10      if  $v$  is standard node then
11         $d \leftarrow \text{CHECKSHIFTDIRECTION}(v)$ ; // see Algorithm 4.
12        if  $d = \text{RIGHT}$  then
13           $x \leftarrow x(v_r) + d_{min}$ ;
14           $\text{SHIFTNODE}(v, d, x)$ ; // see Algorithm 5.
15        else
16           $x \leftarrow x(v_r) - d_{min}$ ;
17           $\text{SHIFTNODE}(v, d, x)$ ;
18      else
19         $d \leftarrow \text{CHECKSHIFTDIRECTION}(v_r)$ ;
20        if  $d = \text{RIGHT}$  then
21           $x \leftarrow x(v) + d_{min}$ ;
22           $\text{SHIFTNODE}(v_r, d, x)$ ;
23        else
24           $x \leftarrow x(v) - d_{min}$ ;
25           $\text{SHIFTNODE}(v_r, d, x)$ ;

```

---

**Pathway Construction**

Since there is no unique method to derive pathways from a BPEL model, we consider the number of descendant BPEL activities of structured activities for path construction and embedding. There is one pathway leading from the top to the bottom of the activity structure. Whenever a pathway  $P$  is split up, copies  $P_1, \dots, P_n$  of path  $P$  are generated. The end of these new pathways  $P_1, \dots, P_n$  is the node where they meet path  $P$  again. Thus, each execution path in BPEL is also a pathway in the graph and is drawn in

**Algorithm 4:** CHECKSHIFTDIRECTION( $v$ )

---

```

1  $\mathcal{S}_p \leftarrow$  set of all predecessors of node  $v$  in  $G$ ;
2  $c_{coverage} \leftarrow (\sum_{v \in \mathcal{S}_p} x(v)) / |\mathcal{S}_p|$ ;
3 if  $c_{coverage} \neq 0$  then
4   if  $x(v) > c_{coverage}$  then
5     return LEFT;
6 return RIGHT;

```

---

**Algorithm 5:** SHIFTNODE( $v, d, x$ )

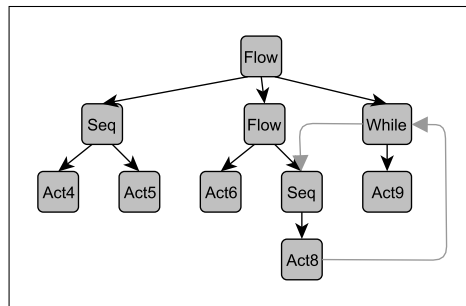
---

```

1  $x(v) \leftarrow x$ ;
2  $v_l, v_r \leftarrow$  left and the right neighbour-node of  $v$  in the layer;
3 CHECKNEIGHBOURDISTANCE( $v, v_r$ ); // see Algorithm 6.
4 CHECKNEIGHBOURDISTANCE( $v, v_l$ );

```

---



**Figure 3.23.:** The simple input structure representing a BPEL workflow. Black edges represent *pathway edges*, gray edges represent *transverse edges*.

parallel by the extended layering algorithm described above. Transverse edges are not considered during the construction of the pathways. However, in the later step that aims at reducing the numbers of edge crossings, these edges affect the relative position of the computed pathways in the final layout: Given a layout containing pathways in the relative ordering  $P_1, \dots, P_n$ , a transversed edge  $e$  between a node of  $P_1$  and a node of  $P_n$  would cause a lot of edge crossings. Thus, by considering  $e$  in the crossing minimization phase, the algorithm correctly places  $P_1$  and  $P_n$  side by side. In Figures 3.24 and 3.25, an example for pathways is depicted using a different color for each pathway.

**Algorithm 6:** CHECKNEIGHBOURDISTANCE( $v, v_j$ )

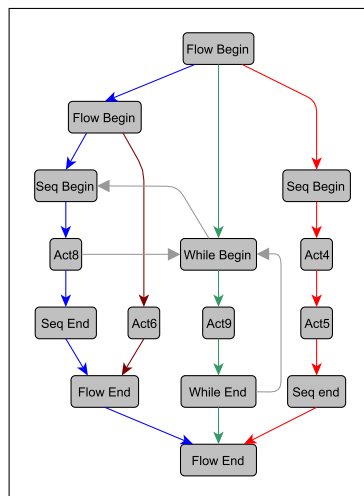
---

```

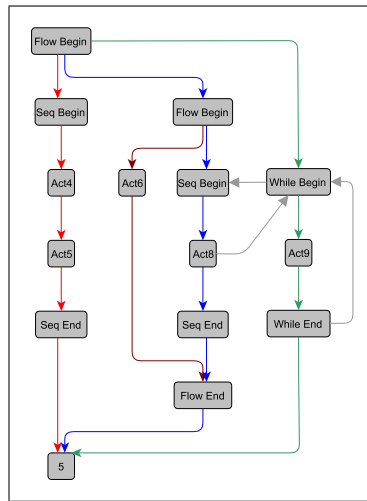
1 if DISTANCE( $v, v_j$ ) <  $d_{min}$  then
2   if  $v_j$  is a standard node then
3     if  $d = \text{RIGHT}$  then
4        $x \leftarrow x(v) + d_{min}$ ;
5       SHIFTNODE( $v_j, \text{RIGHT}, x$ );
6     else
7        $x \leftarrow x(v) - d_{min}$ ;
8       SHIFTNODE( $v_j, \text{LEFT}, x$ );
9   else
10    if  $d = \text{RIGHT}$  then
11       $x \leftarrow x(v_j) + d_{min}$ ;
12      SHIFTNODE( $v, \text{RIGHT}, x$ );
13    else
14       $x \leftarrow x(v_j) - d_{min}$ ;
15      SHIFTNODE( $v, \text{LEFT}, x$ );

```

---



**Figure 3.24.:** Result of the extension for pathway construction from the input structure to a graph  $G$ . In  $G$ , each structured activity, e.g., a Sequence–activity, gets an end–node forming a nested structure. In addition, pathways (colored blue, green, red and brown) are embedded to draw each execution path in parallel. The edges colored in gray are standard edges linking two execution paths.



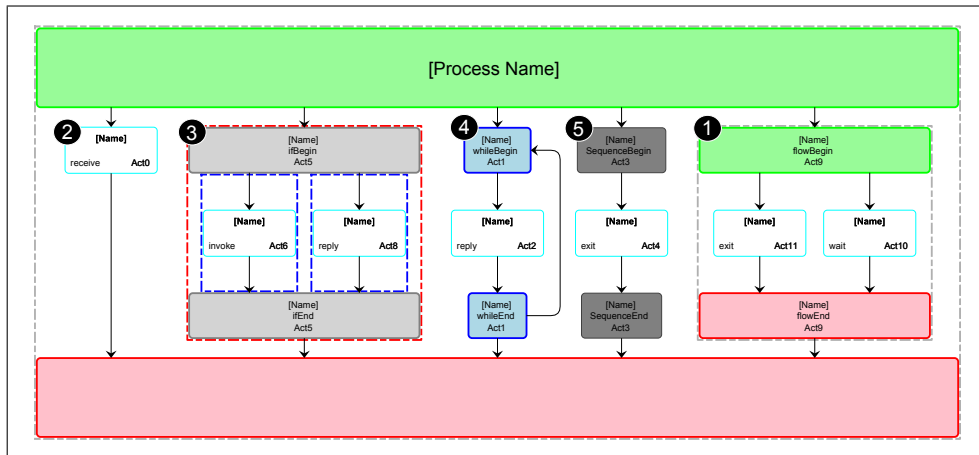
**Figure 3.25.:** Result for the layout of  $G$  of Figure 3.24 respecting the drawing of each pathway straight from top to bottom. Each pathway is highlighted by its own edge coloring. Since each pathway starts at the root node the first edge of the blue pathway belongs also to the brown pathway. Note that colors are used for description purposes only and don't contribute to semantics of a BPEL model.

### Modifications of the Graphical Representation

Each node in the layout is assigned a specific shape and is labeled according to its represented BPEL element, i.e., basic activities are represented by a simple node containing its ID and its type. Structured activities consist of two nodes, its start and its end node containing the ID, see Figure 3.24. The type is expressed by setting the color of the according nodes (green for 'start' and red for 'end' nodes). To *FLOW*-, *PICK*- and *IF*- activities, rectangles are added which surround all subordinate activities. *IF*- and *PICK*-activities have additional rectangles that contain all nodes of each conditional case. Also, in the layout, a label is added to each node that contains information derived from the corresponding node in the original BPEL model (e.g. declarative name, etc.). An example of a layout with BPEL-specific shapes is given in Figure 3.26.

### Meta-Data of the BPEL process

A BPEL process may contain additional (meta-) data for its elements, e.g. conditions for *IF*-activities or comments on elements. The data is relevant for the later transformation to an executable process on a process engine. The visualization of meta-data is not always desirable since important data, i.e., element type, may interfere perception of less relevant data, i.e., comments. Therefore, we provide the meta-data in a table that is constructed after the layout algorithm. The elements are connected to the table



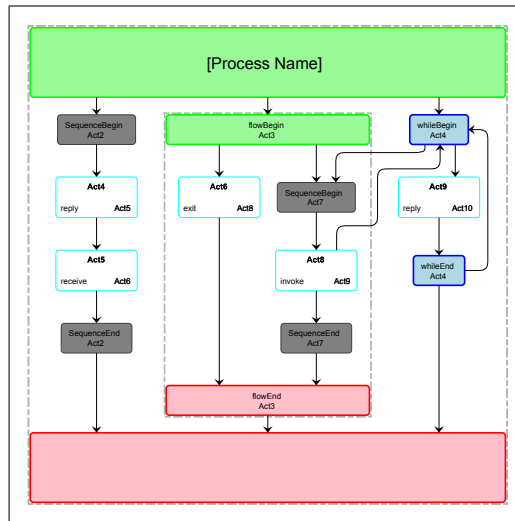
**Figure 3.26.:** ❶ A *FLOW*–activity surrounded by green and red nodes and a gray rectangle; surrounding rectangles are drawn in dashed lines. ❷ A *BASIC*–activity containing the type (left) and the type (right). ❸ A *IF*–activity surrounded by gray nodes and a red rectangle; each case is outlined in a blue rectangle. ❹ A *WHILE*–activity surrounded by blue nodes. ❺ A *SEQUENCE*–activity surrounded by small gray nodes. Each activity also contains a labels that displays its name.

data by their corresponding IDs. Conditions, e.g. *IF*–conditions, are linked by edge labels. For ease of use, the table can be exported to HTML format. An example of such a table is given in Figure 3.28.

### Outline of the Layout Algorithm

In the following, we give a short overview on the complete algorithm:

- Input: A graph structure representing the workflow, see Figure 3.23.
- Step 1: Extension of the input structure to a graph with pathways embedded, result see Figure 3.24.
- Step 2: Extended cycle removal phase.
- Step 3: Extended layer assignment phase.
- Step 4: Standard crossing minimization phase.
- Step 5: Extended assignment of horizontal coordinates. Figure 3.25 as an example for the result of step 2–5.
- Step 6: Adaption of the graphical representation of the layout image, see an example in Figure 3.27.
- Step 7: Handling of meta data: creation of a table linked to layout result, see Figure 3.28 for a small example and its corresponding meta–data table.
- Output: The layout graph representing the workflow and a table containing the meta–data for the corresponding graph.



**Figure 3.27.:** Result of the final layout for BPEL model of Figure 3.24. Each node is assigned its specific shape corresponding to its represented BPEL activity.

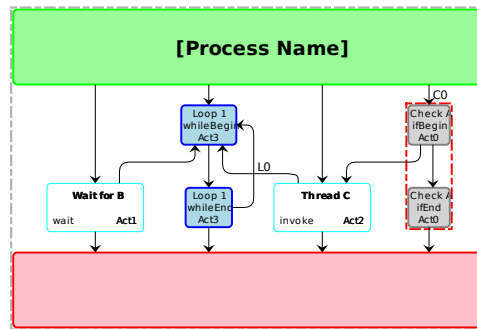
### Analysis of the time complexity

Using heuristics in the Sugiyama algorithm and an acyclic input graph  $G = (V, E)$ , it is possible to reduce the running-time to  $O((|V| + |E|) \log |E|)$  (Eiglsperger, Siebenhaller, and Kaufmann 2005). We give now a worst-case running time of our layout algorithm. All extensions from Section 3.3.3 and further the BPEL-specific steps must be considered.

**Extensions to the Cycle Removal Phase** Obviously, a path division can be done in  $O(|E|)$  time. This division is performed for each pathway containing the cycle edge which is reversed. In practice, the number of cycles is low, thus, given there are  $k$  cycles containing only pathway edges during the Cycle Removal Phase, this extension step can be done in  $O(k|\mathcal{P}||E|)$  time where  $|\mathcal{P}|$  is the number of pathways.

**Extensions to the Phase of Computation of Horizontal Coordinates** First the barycenter of each pathway  $P \in \mathcal{P}$  is computed. Therefore, the  $x$ -coordinate of each pathway node of  $P$  must be taken into account. Hence, the running time for the computation of the barycenters is  $O(|\mathcal{P}||V|)$ . Respecting the minimum node distance  $d_{\min}$ , the distance between two adjacent barycenters must be set at least to  $d_{\min}$ . If the set of barycenters  $\mathcal{B}$  is sorted, this can be done by a simple traversal through  $\mathcal{B}$  in  $O(|\mathcal{B}|) = O(|\mathcal{P}|)$  time or  $O(|\mathcal{P}| \log |\mathcal{P}|)$  including a sorting algorithm, e.g. MergeSort. During a shifting phase of a standard node  $O(|V|)$  locations must be checked. Hence, the running time for shifting all non-path nodes is  $O(|V|^2)$ .





**[Process Name] - Caption**

Node ID	Activity Type	Node Name	Comment	Condition	Join Condition	For Start-Condition	For Stop-Condition	While-Condition
Act0	if	Check A	-	-	C0: A = true	X	X	X
Act1	wait	Wait for B	-	-	-	X	X	X
Act2	invoke	Thread C	-	-	-	X	X	X
Act3	while	Loop 1	-	-	-	X	X	L0: B not ready
ROOT	flow	[Process Name]	-	-	-	X	X	X

**Figure 3.28.:** A table containing the meta–data for each node of the process depicted on top ('-' means the field is not set, 'X' denotes that this field is not available for this activity type).

**Path Construction** For generating the layout graph, the nested activity structure must be resolved. For this purpose, every activity is regarded once and the pathways can be embedded by a simple breadth–first–search (BFS).

Considering all extensions for the worst–case running–time of our layout algorithm, we obtain:

$$\begin{aligned}
 O(S + k|\mathcal{P}||E| + |E| + |\mathcal{P}||V| + |V|^2 + |V| + |E|) \\
 = O(S + k|\mathcal{P}||E| + |V|^2)
 \end{aligned}$$

where  $S$  denotes the dominating running–time of the Sugiyama algorithm. Since pathways do not necessarily have to be node–disjoint, there can be many pathways in a general graph. However, in general BPEL models,  $|\mathcal{P}| \ll |V|$  holds and, thus, only few pathways exist.

### 3.3.4. Summary

In this section, we tackled the challenge of computing layouts for BPEL models. The main contribution is a layout algorithm that is highly adapted to the needs of BPEL and uses the principles of Sugiyama's algorithm enriched with major extensions and modifications. For real-world application purposes and prospective empirical evaluations, the approach is integrated in HOBBS (Held and Blochinger 2008, 2009).

The approach extends the existing BPEL modeling platform HOBBS to enhance communication between designers by discussing and exploring BPEL models. With our approach, complete BPEL models can be analyzed as visual models that do not rely on the original BPEL files in XML-format.

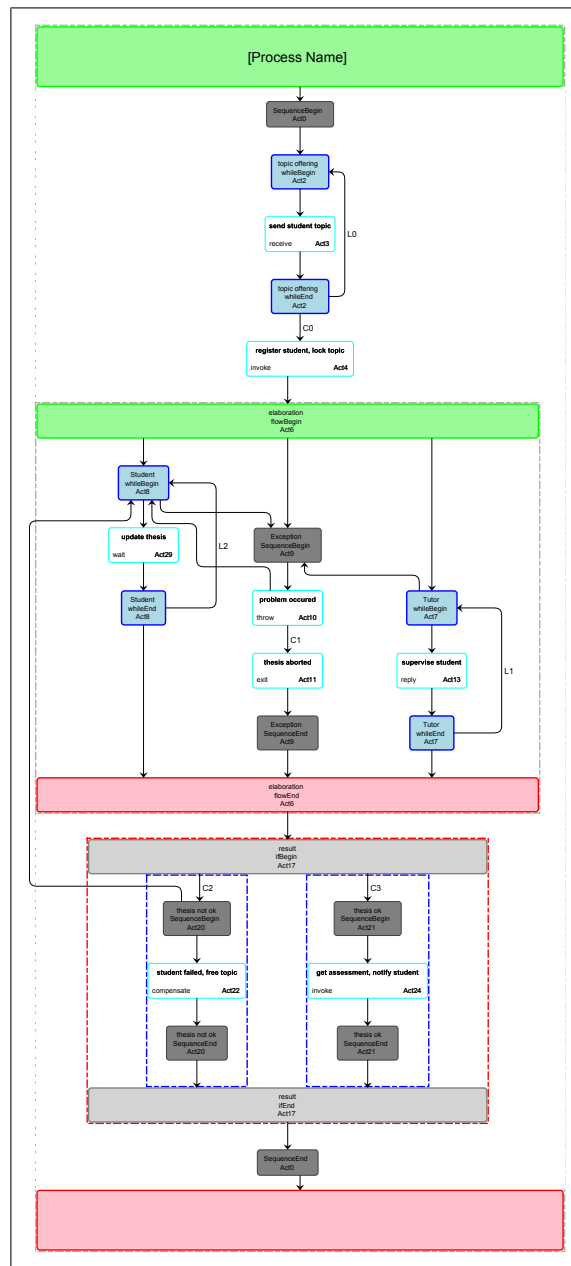


Figure 3.29.: Layout example of BPEL process that represents the workflow of a student that pursues to complete a thesis.



# Chapter 4

## Summary for Part I

In the first part of this work, we presented techniques for visualizations of business process models in *2D*. Visualizations adhere to layout aesthetics. We presented a study on layout aesthetics of visualizations for business process models in BPMN, see Section 2.3. We defined a set of aesthetics that is to be supported by layout approaches for BPMN models. The set comprehends the following aesthetics:

FLOW, PARTITION, OVERLAP, ELEMENT\_SIZE, EDGE\_LENGTH, CROSSING, ORTHOGONAL, LABEL, AREA, BEND.

Furthermore, we adapted in Section 3.1 an approach for dynamic visualizations that considers a given embedding of a graph when computing a layout. The approach Sketch-Driven-Layout (SDL) is realized as an extension to the TSM approach which computes visualizations for graphs in Kandinsky model. The extended approach of SDL is able to consider the partition of a business process models, in the case of BPMN, a partition corresponds to swimlanes.

In the subsequent Section 3.2, three patterns for visualizations of business process were presented. They can be applied to SDL and are targeted at semantic considerations of business process models in visualizations. The patterns and their goals are:

- Geometry pattern (GeoP): reduce visual cluttering of process model diagrams.
- Gateway Pattern (GaP): highlight the logical structure of process models that is induced by gateways.
- Start-End-Pattern (SEP): enforce aesthetics *FLOW* on start and end event. Two variants are provided: dynamic or locked SEP.

We analyzed the effects of the patterns on the layout aesthetics when a visualization is computed using SDL enriched by patterns. The results show that the effects are

manifold and vary for different single patterns or combination of patterns and different layout aesthetics.

In Section 3.3, we presented algorithms for the computation of visualizations for BPEL processes. There, a transformation from XML-based files to graphs was introduced and shapes and structures were integrated in the visualizations to highlight the structure and semantics of BPEL process elements. The layout approach stresses the hierarchical and sequential structure of BPEL processes by defining paths in the layout that are to be visualized in a straight fashion. The algorithms are backed onto the Sugiyama framework. Modifications of the steps in the framework towards the visualization of BPEL processes are elaborated in detail.

In the following second part of this work, we extend the display space for visualizations to three dimensions (*3D*) while applying the concept of two-and-a-half dimensions (*2.5D*). We then present and analyze three different algorithmic approaches for the computation of visualizations for business process models in *2.5D*.

**Part II.**

**Business Process Visualization  
in *2.5D***





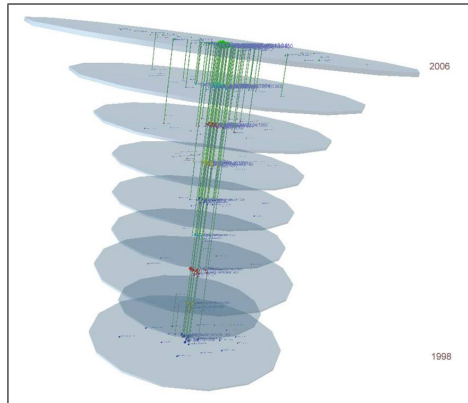
## Introduction to 2.5D–Visualizations

### 5.1. Motivation

In this second part, we present methodologies for visualizations that use 3–dimensional–space ( $3D$ ) for the display of business process models. Visualizing in  $3D$  offers one more dimension of freedom to exploit for the presentation of models. Also, when extending the projection display, e.g. a computer screen, to virtual reality, user perception and interaction can be done more efficiently compared to  $2D$ –presentation (Ware and Franck 1996, 1994). However, from an algorithmic point of view, the additional dimension of freedom, that is given in  $3D$ , has to be controlled and supported by the approaches that are used for visualizations. Moreover, visualizations in  $3D$  are in danger of *clutter* (Mian, Bennamoun, and Owens 2005) and *occlusion* which is a very known phenomenon for the overlap of objects in  $3D$  (Ware 2004). In order to reduce the 'chaos' in  $3D$ –images that can occur when the display (with parameters focus and viewing angle) is not able to master clutter and occlusion, we reduce the  $3D$ –visualizations in this work to two-and-a-half dimensions ( $2.5D$ ). The principle of  $2.5D$  is the following: all elements of a model are fixed to planes in  $3D$ . The planes have fixed depth coordinates. In Figure 5.1, we depict an example for a visualization in  $2.5D$ .

This part applies the concept of  $2.5D$ –visualizations to business process models. We describe the framework that we developed for display and presentation, and we present algorithms for computing  $2.5D$ –visualizations specifically for business process models. The structure of this part is as follows:

In Chapter 6, we will present new methods to create and compute  $2.5D$ –visualizations for business process models. The presentation of the approaches will be followed by an analysis and thorough benchmark process of the performance and layout quality in Chapter 7.



**Figure 5.1.:** Example for a visualization in 2.5D. Here, the planes are used to represent points in time. Source: <http://sydney.edu.au/engineering/it/~shhong/valacon3.htm>, 2012-09-30.

We will now define the terminology that is used in this part for the representation of business process models in two-and-a-half-dimensions. Also, we will point out related work to the concept of 2.5D-visualizations. This chapter concludes with the presentation of our framework for 2.5D-visualizations which is part of *BPMN-Layouter*.

## 5.2. Terminology and related work

First, we define the 2.5D-graph and its properties that are necessary to create a 2.5D-visualization for a business process model in BPMN.

### Definition 12 (2.5D-BPMN-Graph).

A 2.5D-BPMN-Graph is a graph  $G = (V, E, LE)$  with a set of nodes  $V$ , a set of edges  $E$  and a set of layer edges  $LE$ .

Also,  $G$  has the following additional information (meta-data):

- a mapping  $\text{vertex\_type}: V \rightarrow T$ , where  $T$  denotes the set of possible types of a BPMN-element for a vertex  $v \in V$ , see Section 2.
- a mapping  $\text{swimlane}: V \rightarrow S$ , where  $S$  denotes the set of swimlanes. Each vertex  $v \in V$  is assigned to exactly one swimlane  $s \in S$ . A swimlane  $s \in S$  is described by a string and is assigned an identifier (a unique number  $n \in \mathbb{N}^+$ ).
- a layer assignment  $\text{layer}: V \rightarrow L$ , where the ordered list  $L = \{L_1, \dots, L_k\}$ ,  $k \geq 1$  denotes the set of layers.
- the position  $p(v)$ ,  $v \in V$ , of a node in 3D-space is denoted by the coordinates  $\text{pos}_x, \text{pos}_y, \text{pos}_z \in \mathbb{R}$ . □

A layer  $l \in L$  is a rectangular plane in the  $3D$ -space. The sizes and locations (except for  $z$ -coordinates) of the rectangular planes are identical for all layers  $l \in L$ . A layer edge  $le = (u, w) \in LE$  with  $u, w \in V$  is an edge with  $layer(u) \neq layer(w)$ . For all edges  $e = (u', w') \in E$  with  $u', w' \in V$ , it holds:  $layer(u') = layer(w')$ . Thus,  $E \cup LE = \emptyset$ .

We also define a distance measure for layer edges:

$$layerdistance : LE \rightarrow \mathbb{N} : |layer(u) - layer(w)|, \forall le = (u, w) \in LE$$

In a  $2.5D$ -BPMN-Visualization, the following holds: for any vertex  $v \in V$  and its associated layer  $layer(v) = l_v \in L$ , the position of  $v$  in the  $3D$ -space lies within the rectangular plane of  $l_v$ . The layers  $L$  can be considered a stack of rectangular planes in the three-dimensional space differing in  $z$ -coordinates. In Section 5.3.1, we describe how an assignment for  $layer$  can be found for a BPMN-graph and we describe the relationship and analogy of *swimlane* and *layer*.

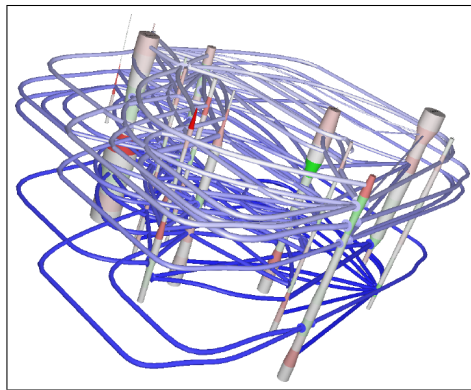
### 5.2.1. Related work on $2.5D/3D$ -graph-layout

Several other approaches have been made in employing  $2.5D$ -techniques for visualizing complex data. One of the fundamental works was presented in (Dwyer 2004) where the usability advantages of  $2.5D$ -layouts for structured and time-lined data are examined. The work is based on series graphs, where layers are used to represent distinct points in time and nodes (dis-) appear over time. For recognizing a node at two points in time (equal to two layers), the points are virtually connected using *virtual edges*. However, virtual edges are employed to link identical nodes over time but not considered during the layout computation. For  $2.5D$ -representation, a *stratified graph*  $G$  is defined as a series of subgraphs  $S_G = (G_{t_0}, \dots, G_{t_\tau})$  with  $\bigcup_{i=0}^{\tau} G_{t_i} = G$  where  $t_0, \dots, t_\tau$  is an ordered sequence of time instants. Each subgraph is mapped to a perpendicular plane to the  $z$ -axis, or *stratum*. Stratified graphs are related to evolving graphs (Erten et al. 2003). For layout algorithms for stratified graphs, it is stated in (Dwyer 2004): “Since a node may appear in multiple strata it would make sense to position the node at the same position in each stratum”. Therefore, the proposed approach for a layout to a stratified graph is “to find a  $2D$ -embedding of the union graph  $G$ , then extrude into  $3D$ , placing edges and nodes at the appropriate depths”. The constraints for a  $2.5D$ -layout of stratified graph are:

- (C1) Edges must only connect nodes with the same  $z$ -coordinate.
- (C2) Edges must lie in a plane orthogonal to the  $z$ -axis and intersecting the  $z$ -axis at the same level as the nodes.

Due to C1 and C2, stratified graphs do not possess layer edges. The given approaches in (Dwyer 2004) for 2.5D-layouts of stratified graphs are: (a) a force-directed approach for 2D with an extension for strata, called *worms*, which represent identical nodes in different subgraphs over time (Dwyer and Eades 2002), (b) a hierarchical layout approach where Sugiyama's algorithm is applied with adaptations in the crossing minimization phase, by weighting a crossing by the number of times that the causing edges are present in a common stratum, and by allowing overlaps of dummy nodes in the horizontal coordinate assignment phase which leads to "significant improvement in aspect ratio of the final layout" (Dwyer 2004). In Figure 5.2, an example for a layout of a stratified graph is depicted.

Unfortunately, the approaches in (Dwyer 2004; Dwyer and Eades 2002) do not consider layer edges. Therefore, the given algorithms cannot be applied to 2.5D-BPMN-graphs where optimal routing of layer edges is a critical requirement.



**Figure 5.2.:** Example for a 2.5D-layout of a stratified graph with the hierarchic approach of (Dwyer 2004). The pillars, or worms, represent identical nodes of subgraphs over time. Each layer, or stratum, represents a time instant. Note that layer edges, i.e., edges connecting or crossing different strata do not exist in a stratified graph. Source: (Dwyer 2004).

Other works include research into navigation methods for three-dimensional layouts of clustered graphs and trees (Ahmed and Hong 2007) and also an approach for hierarchical drawings of directed graphs in 2.5D (Hong, Nikolov, and Tarassov 2007) which we will describe in more detail in Section 6.4.1 due to similarities to our visualization approach (PSL) described in Section 6.4.

An adaption of a 2D-layout approach that is extended to 3D by wrapping a 2D layered drawing around a cone or a cylinder is presented in (Ostry 1996). The system GIOTTO3D uses the following approach (Garg and Tamassia 1996b): GIOTTO3D employs a 3-phase algorithm for drawing hierarchies in 3D. In the first phase a planarization method is used to draw the graph in 2D; then, nodes and edges are assigned

z-coordinates such that all edges point into the same vertical direction and the total edge length is minimized. In the last phase, the shape of nodes and edges are fixed.

A 3D layout approach for UML models that computes layouts with a force-directed approach is designed in (Dwyer 2001). Visualizations for Object Oriented Software (OOC) in 3D are presented in (Ware, Hui, and Franck 1993) which also states the requirement for resulting layouts that "the nodes should be laid out in a top-down fashion in horizontal layers" using a topological sort. The rise of new modeling languages allowed more fields of higher dimensional graph layout, e.g. for class template diagrams (Hoipkemier, Kraft, and Malloy 2006) or JAVA code (Fronk, Bruckhoff, and Kern 2006).

Many layout approaches exploit the structure of the underlying graph structure, e.g. a tree structure (Balzer and Deussen 2004) or hierarchical structure (Wettel and Lanza 2007) which is an example for the *City* metaphor in visualization.

Related work that employs techniques or algorithms similar to our approaches, presented in Chapter 6, will be mentioned in the description of the corresponding approach.

### 5.2.2. Related work on (Business Process) Visualization in 3D

Visualization is a huge field even if restricted to 3D. A comprehensive and excellent overview can be found in (Teyseyre and Campo 2009). Early approaches on visualization in 3D stem from the graph drawing community presenting interactive graph visualization (Bruß and Frick 1995) and program information (Reiss 1994) in 3D. A framework for 2.5D-visualizations for trees (PolyPlane) is presented in (Hong and Murtagh 2004) and is extended to three dimensions (MulitPlane) in (Hong 2005) which is also part of the GEOMI framework (GEOMetry for Maximum Insight) (Ahmed, Dwyer, et al. 2005). The graph visualization system *WilmaScope* (Dwyer and Eckersley 2001) is able to be employed for computing complex visualizations in 3D.

Other works make use of the 3D hyperbolic space for the investigation of methods for visualizations of larger graphs (Munzner 1997).

For our 3D-navigation tool, we adopted conventions for navigation and interaction from (Herman, Melançon, and Marshall 2000), a comprehensive survey on navigation and interaction techniques.

In (Bobrik, Reichert, and Bauer 2007), visualization methods are analyzed with respect to individual views of different users on a process. A case study on business processes in the 3D space is performed in (Schönhage, Ballegooij, and Eliëns 2000). The study concluded that, on the one hand, 2D diagrams are more easily accessible but, on the other hand, using 3D-space enables to combine more information in a single scene. Also, it is stated that visualizations in 3D are not yet widely accepted among business

people. In (Jablonski and Götz 2007), perspectives of views on a process are explored following the approach of perspective oriented process modeling. We make use of the findings of (Jablonski and Götz 2007) in Section 5.3.1.

A representation of business process models that allows for modeling in 3D is developed in (Betz et al. 2008) and (R. A. Brown and Recker 2009). A dedicated environment for modeling in 3D with support of various types of (workflow) diagrams is presented in (Pilgrim and Duske 2008). The approach also offers integration into the powerful Eclipse editing framework GEF<sup>1</sup>. Modeling environments in 3D that use, for instance, Second Life<sup>®</sup><sup>2</sup> are presented in (R. Brown 2010); an extended version for collaborative modeling in 3D is shown in (West, R. A. Brown, and Recker 2010).

### 5.3. A 3D-Framework for 2.5D-Visualizations

This section<sup>3</sup> presents our framework for displaying visualizations in 2.5D. Development of the 2.5D-framework started in 2006 (Jainek 2006) with basic graph presentation in 3D and graph data structures for representations of 2.5D-graphs. Also, initiating work integrated basic actions for a viewer interface, e.g. a graph displayed in a 3D-environment could be rotated, tilted and zoomed in or out. Further extensions and features were added by (Spielmann 2009; Stegmaier 2011), see implementation details in Section 5.3.2.

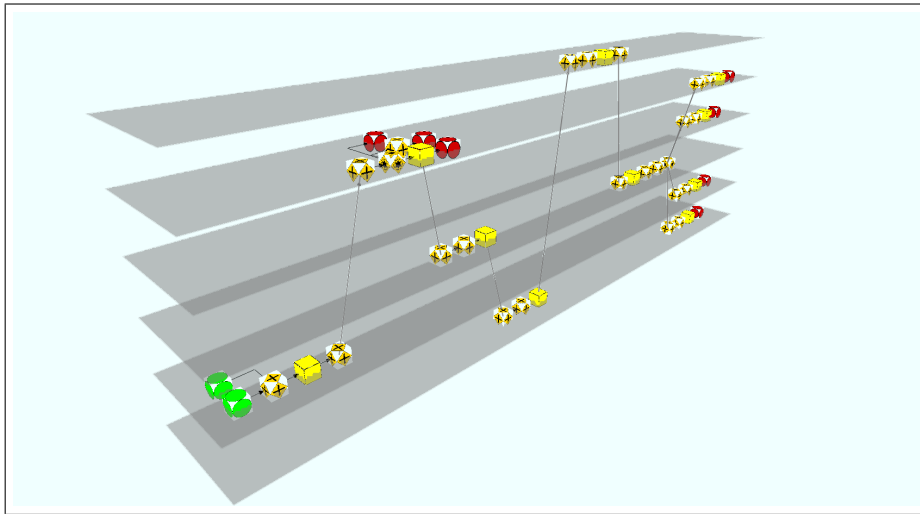
We now add data structures and rendering features for the presentation of business process models, see Figure 5.3 for an example of rendering a business process model in our 2.5D-framework. Also, the framework was integrated into *BPMN-Layouter*. Thus, *BPMN-Layouter* together with the *3D-Navigator* of the framework support layout and navigation for business process models in 2D and 2.5D/3D.

In our *Navigator* which is the interactive user interface of the framework, we represent a single BPMN element by applying its given BPMN shape, e.g. the texture for a task element, to all sides of the corresponding 3D-cube in the 2.5D-BPMN-visualization. This ensures that the type of an element in the 2.5D-BPMN-visualization can always be recognized regardless of the current viewing angle in the Navigator, see the example in Figure 5.4. Also, the guidelines presented in (Ware 2001), suggest to “Use 3D objects to represent data entities”. According to (Ware 2001), evidence from cognitive psychology and experiments suggest that renderings of 3D objects provide more recognisable *glyphs* in an information visualisation than 2D symbols. In information visualization, a glyph describes either a 2D symbol or 3D object representing a data entity (Dwyer 2004).

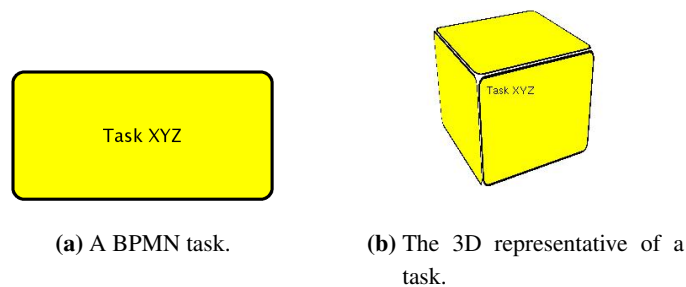
<sup>1</sup><http://www.eclipse.org/gef3d/>, 2012-09-30.

<sup>2</sup><http://secondlife.com/>, 2012-09-30.

<sup>3</sup>Parts of this section are published in (Effinger and Spielmann 2010).



**Figure 5.3.:** Example for the rendering of a business process model in the 3D-framework.



**Figure 5.4.:** In the 2.5D-BPMN-visualization, textures are applied on each side of the cube representing a BPMN-element.

For the projection of business process models, based on 2D-graphs, into 3D-space and 2.5D-graphs, we developed three different *perspectives* which are presented in the following.

### 5.3.1. Criteria and Perspectives

In general, a prerequisite of the concept of 2.5D is that elements of the underlying graph must be assigned to layers. The definition of criteria for the assignment to the layers is elementary for the resulting visualization. A perspective on a business process provides a mapping for every element to exactly one layer. Thus, perspectives define the assignment of elements and must be chosen carefully in order to produce

meaningful 2.5D-visualizations. In our case, the layering mapping *layer* for the 2.5D-BPMN-visualization in the framework can be derived from a perspective.

In (Jablonski and Götz 2007), the following set of perspectives on a general business process is presented:

- Functional perspective: identifies process steps and defines its purpose.
- Data (flow) perspective: defines data used in the process and the flow of data between steps. This perspective also involves external data flow.
- Operational perspective: specifies which operation (service) is invoked in order to execute a process step. It relates to services derived from (external) service libraries.
- Organizational perspective: defines agents, e.g. users or roles in general, that are responsible for process steps. Also, agents can be external.
- Behavioural perspective: defines causal dependencies, also called control flow, between modeling elements.

Note that the aforementioned perspectives were developed independently from a modeling notation. Thus, the perspectives cannot be adopted to BPMN without further modifications. For example, since BPMN is not an executable language, the operational perspective which represents (external) library calls, performed during execution run-time, is not applicable to BPMN. Also, since the control flow in a BPMN model is explicitly defined by the sequence flow, a behavioural perspective corresponds to a common control flow (workflow) in the process. No new insights can therefore be gained by applying a behavioural perspective to a BPMN visualization.

Thus, for introducing perspectives for BPMN, we modified the remaining perspectives for layer assignment in 2.5D-visualizations and present them in the following:

- Organizational perspective: Layers are assigned to elements according to the attributed swimlane exploiting the mapping *swimlane* in the BPMN-graph. Since swimlanes represent, i.e., departments in a company, the 2.5D-visualization is an intuitive view on the process, e.g. regarding the organizational chart of a company.
- Control flow/Data flow perspective: In BPMN, the *Sequence flow* corresponds to control flow and *Message flow* corresponds to data flow. Both flows are treated separately in this perspective. Elements are assigned to layers depending on incident connecting objects. Elements that provide incoming messages to the process are assigned to higher layers than elements that receive messages from the process. Elements that only have sequence flow objects are assigned to the so-called *main process flow layer* that represents the process' flow. Assuming a message triggers a subprocess in a different pool, we can have several *process*



*flow layers* which are assigned to subsequently lower layers than the initiating *main process flow layer*.

Thus, sequence flow and message flow are visualized in a diametrical manner; message flow runs vertically (orthogonally to the layers) and sequence flow runs horizontally (within the layers). This segregation of flows offers a information gain from the 2.5D-visualization.

- **Functional perspective:** The BPMN elements types defined by *vertex\_type* are used for layer assignment. Moreover, the process flow is traceable from top to bottom because start events are assigned to higher layers than intermediate events and end events. Thus, this perspective provides an analytic view on the process model; e.g. distinct views on the usage of certain elements can be inspected. The process model structure becomes clearly visible, e.g. many gateways may indicate a high complexity of the process model.

In the following, we provide an example and discuss the benefits and drawbacks for each of the perspectives.

In Figure 5.5(a)-(d), a simple process and its perspectives are depicted. Although the example is very simple, it becomes clear that not all perspectives fit best, i.e. in the functional perspective, see Figure 5.5(d), a layer only contains one element. This leads us automatically to a *rating* among the perspectives and distinct preferred use cases:

A perspective can be considered 'better' than other perspectives when the layering of nodes is nearer to an equal distribution among the layers while keeping the total number of layers low. Thus, it is neither desirable that all nodes are assigned to a single layer, nor that only one node is assigned to one layer. Examining the three perspectives with respect to this rating, they all have different use cases where their application fits better or is less appropriate:

- **Use cases for Organizational perspective (Figure 5.5(b)):** Since the organizational perspective is tied to an organizational chart of a company, it has a high similarity to the structure of a process in a company with several participating departments. The perspective is superior to the others if the elements of the process (e.g. tasks) are nearly equally distributed among the departments (and thus among the layers) and not too many departments are involved which would always lead to an increased complexity of a process, in the 2D-diagrams and in the 2.5D-BPMN-visualization as well.
- **Use cases for Control flow/Data flow perspective (Figure 5.5(c)):** The perspective is useful when there exist data that are passed in the process. Otherwise, all nodes are part of the control flow and, therefore, they are assigned to the *main process flow layer*. This represents a worst case for the rating of the perspective. On the other hand, if the amount of messages is higher, e.g. the process needs

interaction with (external) data providers, the flow of messages can be visualized in an intuitive way using the perspective. Recall that the data flow is arranged from top to bottom in the 2.5D-BPMN-visualization.

- Use cases for Functional perspective (Figure 5.5(d)): If the process mainly consists of one start event, numerous tasks that are to be executed sequentially and eventually lead to one end event, this use case represents the worst case of the functional perspective since all elements except start/end event are assigned to the same layer. The gain of the perspective is increasing for a process that makes use of more than only a few basic element types. Then, e.g. the workflow that is arranged from top to bottom can be inspected and the user can analyze if one element type is over-represented and may cause redundancies.

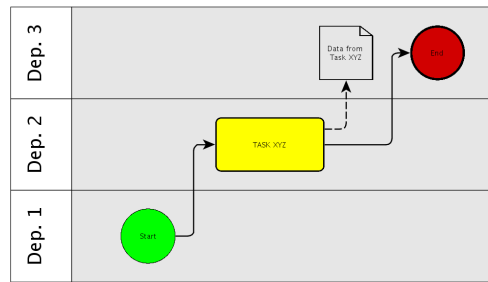
Our framework comprises an implementation of all three perspectives. Examples of the resulting visualizations are given in Figure 5.8. In the following, we give details of the framework implementation and then describe the navigation support that is provided in the Navigator.

### 5.3.2. Implementation

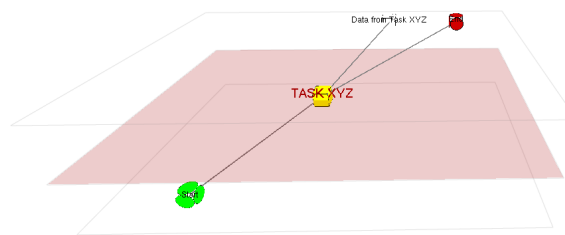
For the display of our 2.5D-BPMN-visualizations, a basic framework for 2.5D-visualizations was initiated in (Jainek 2006) and continued in (Spielmann 2009). Our framework is developed using Sun JAVA™ 1.5 and the graph library *yFiles* from yWorks (Wiese, Eiglsperger, and Kaufmann 2001) for basic graph data structures. For displaying and rendering objects in 3D and animation of transformations, we chose the Java Open Graphics Library (*JOGL*).

Using the general implementation presented in (Spielmann 2009), it is simple to create 2.5D-visualizations for many kinds of graphs. To achieve this, we built an abstraction that moderates between the different base libraries we use. This abstraction is divided into two parts: the creation, or factory class on the one hand, and the display classes on the other hand.

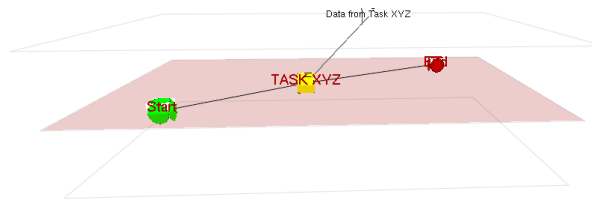
The central element of the creation library is the Graph-2.5D-Factory which stands as the top level interface for transforming 2.5D-graphs into a 2.5D graph representation. For the transformation into 2.5D, a mapping to the layers must be given. The layer mapping is given by one of the above described perspectives that is chosen by the user. After a 2.5D-graph is created by the Graph-2.5D-Factory class, the display classes render the 2.5D-graph into the 3D-Navigator of *BPMN-Layouter*. The display classes exploit the metadata of the 2.5D-graph given by *layer*, *vertex\_type* and node positions  $p(V)$ . Note that *vertex\_type* is given by the 2D-graph whereas initial positions of  $p(V)$  are set to positions of nodes in the input 2D-graph ( $pos_x$  and  $pos_y$ ) and the position of the assigned layer from *layer* in the layer stack ( $pos_z$ ).



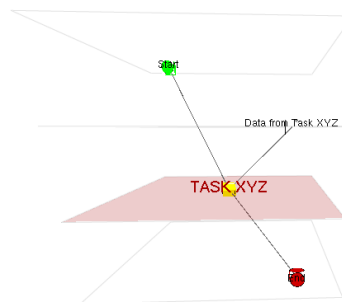
(a) BPMN-Graph of a simple BPMN process.



(b) Organizational perspective.



(c) Control/Data flow perspective.



(d) Functional perspective.

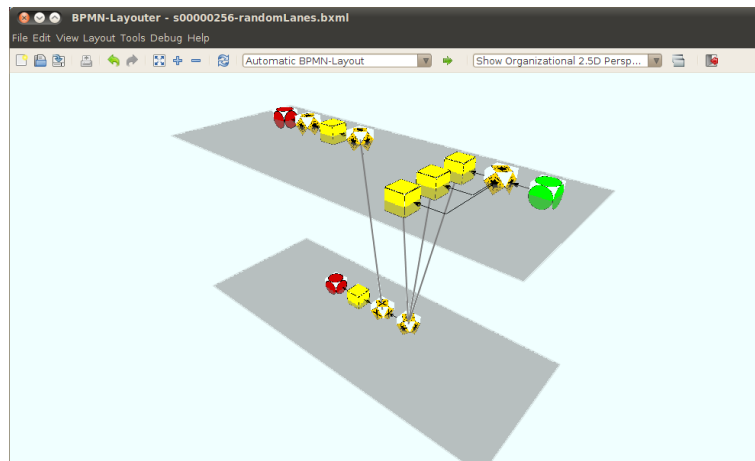
**Figure 5.5.:** Example process in 2D-layout (a). The three different perspectives (b) - (d) presented in Section 5.3.1 are applied to the example of (a). The display is centered on the layer containing the task (red).

### 5.3.3. Navigation Support

We chose an interactive model of navigation in a 2.5D-visualization display. Our Navigator offers mouse-actuated navigation on the currently active viewing plane and rotation of that plane, as well as changing the viewing height to accommodate viewing of different layers. Apart from that, we use keyboard shortcuts for rotation, tilt and change of viewing height of the display. Moreover, layer distance and graph element zoom scale can be changed dynamically using keyboard shortcuts. Thus, the user can navigate freely in the 2.5D-visualization of the BPMN model and adapt the display to individual preferences.

For computer-aided navigation, layers can be inspected individually and the focus of the display is automatically adapted to center the currently active layer. Another key point of the navigation is the ability to create a history of viewpoints. With a single keystroke, the current position is saved and added to a list of viewpoints. These can then be traversed in chronological order, thus allowing to backtrack and review already visited parts of the visualization.

All transitions between different viewpoints are animated with soft movements so to not disturb the mental model of the viewer. With these techniques combined, we achieve a persistent model with convenient navigation handling that allows viewers to survey large and complicated processes swiftly. In Figure 5.6, a screenshot of our 2.5D-Navigator is depicted with its integration into the *BPMN-Layouter*.

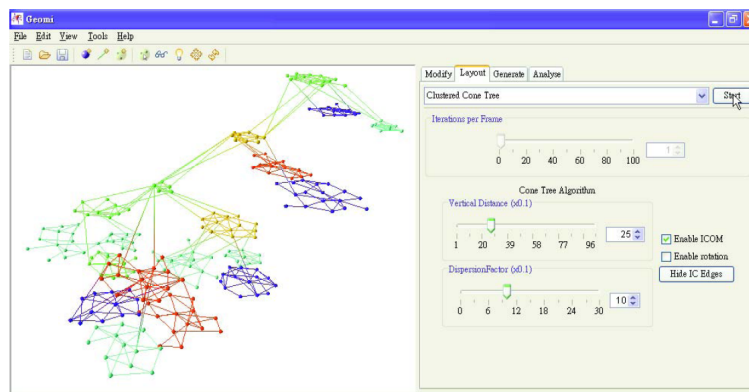


**Figure 5.6.:** Screenshot of the 2.5D-visualization display of our 2.5D-Navigator embedded in *BPMN-Layouter*.

### 5.3.4. Summary

In comparison to other frameworks for 3D-environments, e.g. WilmaScope (Dwyer and Eckersley 2001), GEOMI (Ahmed, Dwyer, et al. 2005) and GEF3D (Pilgrim and Duske 2008), our 3D-Navigator has the benefit of the integration into *BPMN-Layer* such that modeling in 2D, inspection and analysis in 2D/2.5D using one of the above presented perspectives can be performed seamlessly. According to WilmaScope’s website<sup>4</sup>, the framework was not updated after October 2003 which does not indicate the support for state-of-the-art layout techniques. The focus of GEF3D is on the modeling part of business process models. As the author of GEF3D states himself in July 2009, ”layouts are a big issue in GEF3D”<sup>5</sup>.

GEOMI is a powerful framework that also supports layout algorithms which can be attached as plug-ins. GEOMI is also based on previous work on WilmaScope. It attempts at ’visually explore networks and discover patterns and trends’ (Ahmed, Dwyer, et al. 2005). Currently, GEOMI supports various layout algorithms for 3D: force-directed layout, clustered graph layout and hierarchical layout. The approach for hierarchical layout is described in (Hong, Nikolov, and Tarassov 2007) and we will compare this approach in-depth with our visualization approaches in Section 6.4.1. A weakness of GEOMI is the missing modeling part. Graphs can be loaded from files or generated by a built-in graph generator. However, graphs cannot be created and designed within the framework. In Figure 5.7, a screenshot of the GEOMI-framework is depicted.



**Figure 5.7.:** Screenshot of the GEOMI-framework with activated plugin for clustered graph layout. Source: (Ahmed, Dwyer, et al. 2005).

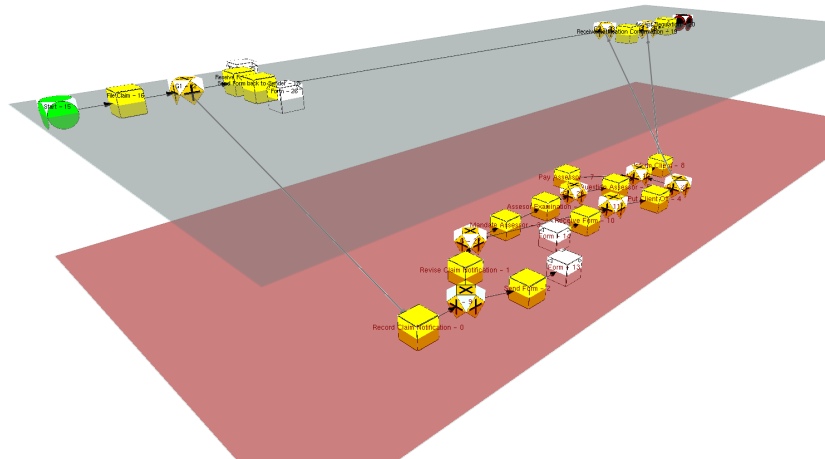
Our framework benefits from the fact that the visualization approaches that we will present in the next chapter are also integrated into our 3D-framework. Together

<sup>4</sup>see <http://wilma.sourceforge.net/>, 2012-09-30.

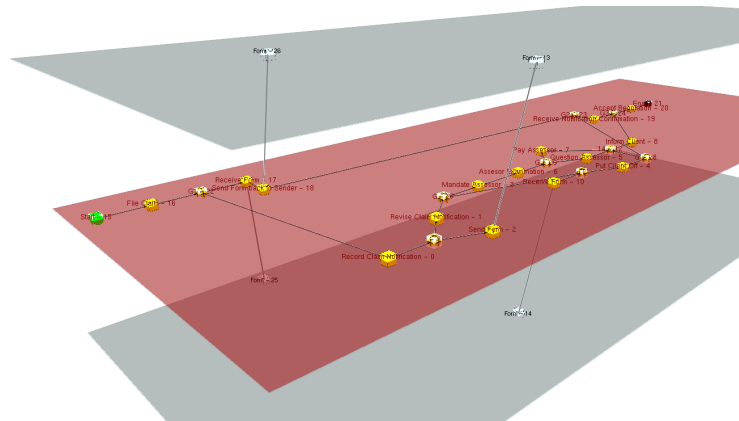
<sup>5</sup>see forum entry at <http://www.eclipse.org/forums/index.php/m/563904/>, 2012-09-30.

with the modeling in 2D, layout in 2D and the interactive 3D-Navigator, our *BPMN-Layouter* is a very powerful tool for designing, analysing and presenting BPMN business process models.

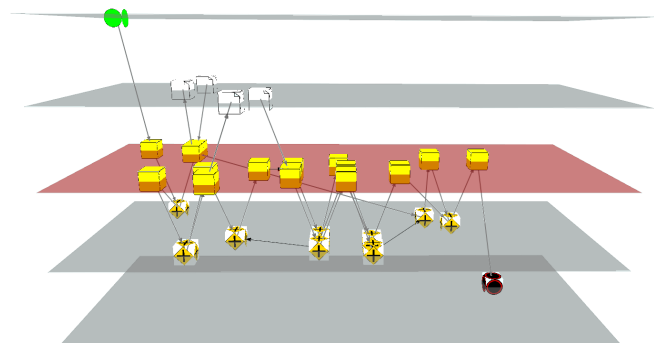
In our framework, only few technical considerations must be observed, since the foundation of our implementation is based on well-understood and widely used libraries. The frame rate of our 3D-framework is acceptable on modern computers, settling for small models with approximately 30 elements on around 50 fps and for larger models ( $|V| \approx 100$ ) on around 25 fps. The frame rate is limited mainly by the number of objects in the scene. Thus, visualizations are limited to relatively small graphs. However, this does not pose a limitation to this application since BPMN graphs are not expected to be very large ( $|V| < 150$ ) in general. This expected size for business process models will be confirmed in Chapter 7 when the visualization approaches for 2.5D, that we present in the next chapter, will be subject to an analysis and evaluation using real-world process models.



(a) Organizational perspective of a BPMN model. The elements are assigned to the layers depending on their swimlane assignment *swimlane* in the BPMN model. Highlighting layers (see red marked layers and node labels) is a feature of 2.5D-Navigator for accentuating a specific layer.



(b) Control/Data Flow perspective of the same BPMN model as in (a). Incoming data flow is assigned to the upper layer. Elements incident to control flow edges are drawn in the highlighted main process flow layer. Outgoing data flow is placed in the lower layer.



(c) Functional perspective of the BPMN model of (a). Elements are assigned to layers according to their BPMN types *vertex\_type*. For better distinction, the layer containing all tasks is highlighted and the node labels are hidden.

**Figure 5.8.:** Application of the perspectives to an example process. Selected layers are marked in red and the labels of the corresponding elements are emphasized.

We observe that, for the example process, the organizational perspective fits best since it offers the best distribution of elements to layers and, at the same time, occupies the lowest number of layers. However, the functional perspective shows clearly that *task* elements dominate the element type in the example process.





# Visualization Approaches for Business Process Models in 2.5D

## 6.1. Motivation

This chapter comprehends a description of approaches for 2.5D-visualizations. We present three different approaches which employ distinct graph drawing techniques. At first, we will present an approach that uses a *layer sweep* technique in order to sequentially improve the drawing layer-by-layer. As a second approach, we will model an ILP (integer linear program) for finding an optimal solution of the 2.5D-visualization problem. The third approach exploits the sparse structure of process models and tries to find hierarchical paths, using a ranking of nodes for quickly finding a partitioning, and also applies Sugiyama's algorithm (Sugiyama, Tagawa, and Toda 1981) for the final visualization.

All three approaches have the following objectives:

- **OBJ1** – aesthetics *FLOW*: nodes should be placed in the natural order of the process' sequence flow. This goal has highest priority.

For an edge  $e \in E$  in a graph  $G = (V, E)$ , *FLOW* is fulfilled if the following implication is true:

$$impl : e = (u, w) \Rightarrow pos_d(u) < pos_d(w)$$

where  $pos_d(u)$  denotes the position of node  $u$  in coordinate  $d$ ;  $d$  defines the axis of the flow orientation. In the visualizations, the flow is defined to be oriented left-to-right in a two-dimensional plane, therefore,  $d = x$ .

The goal of **OBJ1** is to maximize the number of edges for which implication *impl* evaluates to *true*, or

$$\mathbf{OBJ1} : \max |\{e \in E | \text{impl}(e) = \text{true}\}|.$$

- **OBJ2** – low edge lengths: edges, and layer edges in particular, should have lowest possible edge length.

In general, for a 2.5D–BPMN–Graph  $G = (V, E, LE)$ , the objective **OBJ2** can be expressed as an optimization problem:

$$\mathbf{OBJ2} : \min \left( \alpha \cdot \sum_{(u,w) \in E} \mathcal{L}(p(u), p(w)) + \beta \cdot \sum_{(u,w) \in LE} \mathcal{L}(p(u), p(w)) \right)$$

where  $\mathcal{L}$  is a metric for the distances between the nodes positions  $p(u), p(w)$  and  $\alpha, \beta$  are weights for the different edge types  $E$  and  $LE$ , i.e., in order to prioritize short layer edges  $LE$  over regular edges  $E$ , it is set  $\beta > \alpha$ .

- **OBJ3** – low area size: the amount of area space used for the visualization should be small. In 2.5D, the area is given by the size of the rectangle that is consumed by the largest of the planes surrounding the layers. For a 2.5D–BPMN–Graph  $G = (V, E, LE)$  and a rectangle  $r$  induced by  $(p_{min}^x, p_{min}^y, p_{max}^x, p_{max}^y)$ , where

$$\begin{aligned} p_{min}^x &= \min \{pos_x(v) | v \in V\}, \\ p_{max}^x &= \max \{pos_x(v) | v \in V\}, \\ p_{min}^y &= \min \{pos_y(v) | v \in V\}, \\ p_{max}^y &= \max \{pos_y(v) | v \in V\}. \end{aligned}$$

**OBJ3** aims at minimizing the size of  $r$ , or

$$\mathbf{OBJ3} : \min (|p_{max}^x - p_{min}^x| \cdot |p_{max}^y - p_{min}^y|).$$

After the description of the algorithms in this chapter, we will analyze the approaches with respect to **OBJ1–OBJ3** and create performance benchmarks in the next chapter.

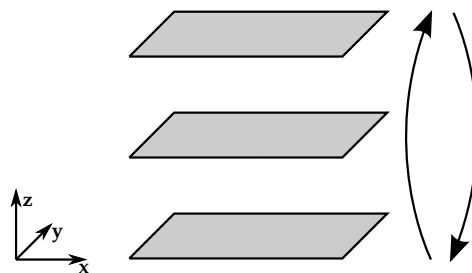
## 6.2. Approach 1: Sequential Layer Sweep

The first approach tries to sequentially improve the layout layer-by-layer using a sweep line algorithm. A sweep line algorithm is a technique that is used in various problems of computational geometry, e.g. for the construction of Voronoi diagrams (Fortune 1987).

A sweep line algorithm handles events that occur when the (sweep) line/plane passes points in Euclidean space. The set of points is given as input and is ordered/sorted,

e.g. in their  $x$ -coordinates. The line/plane represents the current status of the algorithm; when a point  $p$  is passed, the algorithm checks if the current line together with the point  $p$  cause an event that might update the current status. For a detailed description of the general concept of sweep line algorithms, we recommend, for instance, the introduction to computational geometry in (Berg et al. 2000, pp.20 ff.).

In our case, instead of the set of points, the line passes the layers of the  $2.5D$ -graph. Since the layers are placed in  $3D$ -space as a stack, we start from the top layer. The ordered input set for the sweep line algorithm corresponds to the stack of layers in the  $2.5D$ -graph sorted by  $z$ -coordinate, see Figure 6.1.



**Figure 6.1:** Layer stack in  $3D$ -space and (alternating) sweep line direction (arrows on the right).

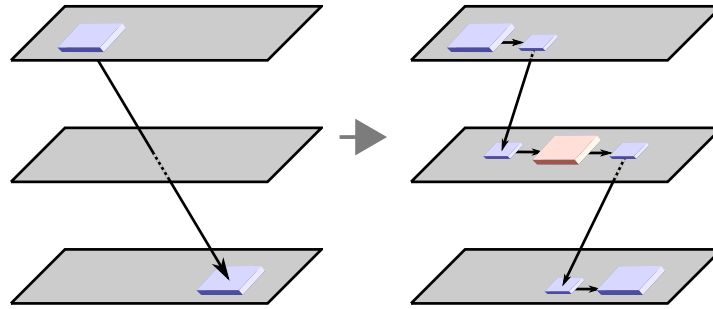
During the sweep line algorithm, we apply, for each layer, a call of the layout algorithm that is adopted from the approach presented in Section 3.1.1, followed by an update stage that we describe later.

In order to render the approach feasible for  $2.5D$ , we now must adapt the approach from Section 3.1.1 in order to handle layer edges. This is achieved by performing the following: a layer edge  $le$  is replaced by a pair of dummy nodes  $(u; w)$  where  $u$  is connected to the source  $s$  of  $le$  and  $u$  is placed in the layer of  $s$ ; and  $w$  is connected to the target  $t$  of  $le$  and  $w$  is placed in the layer of  $t$  respectively. The dummy nodes represent symbolic links to the corresponding source/target of the former layer edge. The symbolic link between  $u$  and  $w$  is stored in a map *references*. Dummy nodes are stored in a set  $D$ . We call  $u$  the *counterpart* of  $w$  and vice versa.

If a layer edge  $le$  connects two non-adjacent layers (*layerdistance*  $> 1$ ), we need to insert placeholders in intermediate layers for splitting the layer edge into segments with layer distance equal to 1. We call these new placeholders *principal nodes*. In Figure 6.2, the replacement step is described. Each principal node is connected to two dummy nodes that represent links to the next lower/higher layer.

After the replacement of layer edges by dummy nodes and principal nodes, we start the layout computation by initiating the first sweep. The sweep comprehends several

rounds. A round starts from the top layer and continues downwards, until the bottom layer is reached. Then, the sweep direction is reversed and the algorithm continues (upwards), as depicted in Figure 6.1.



**Figure 6.2.:** Replacement step of a layer edge  $le$  by dummy nodes (small blue boxes) and principal nodes (red box). Principal nodes are inserted only if  $le$  crosses intermediate layers.

Each round contains  $|L|$  steps. A step  $i$  corresponds to the handling of one layer  $L_i$ , therefore  $1 \leq i \leq |L|$ . In each step  $i$  of the sweep, a layout for a single layer  $layer(i)$  is computed. The layout also includes the inserted dummy nodes  $u_1, \dots, u_k$  and principal nodes  $p_1, \dots, p_l$  in that layer.

After computing the layout of a layer  $L_i$  in step  $i$ , we perform the update stage. In the update stage, we consider the dependencies of the former layer edges, now represented by dummy nodes and principal nodes. We update the positions of dummy nodes  $w_1, \dots, w_k$  in other layers  $l \in L \setminus L_i$  that represent the corresponding counterparts of  $u_1, \dots, u_k$ , see Algorithm 7. The update stage changes the positions of the counterparts  $w_1, \dots, w_k$  as follows:  $w_1, \dots, w_k$  are placed orthogonally to the layer stack (or parallel to the  $z$ -coordinate) to be vertically aligned with  $u_1, \dots, u_k$ . After the update stage, the  $x$ - and  $y$ -coordinates of two nodes  $u_i$  and  $w_i$  are identical. Note that only positions of dummy nodes are changed in the update stage; principal nodes and regular nodes are considered fixed.

In the following step  $i + 1$ , we perform the layout computation in layer  $L_{i+1}$ . In this step, we take into account that positions of dummy nodes might have been changed in previous steps. Since our applied layout approach from Section 3.1.1 attempts to preserve the given embedding as a sketch, dummy nodes positions, that were updated during an update stage of step  $j < i + 1$ , influence the final positions of nodes that are incident to dummy nodes.

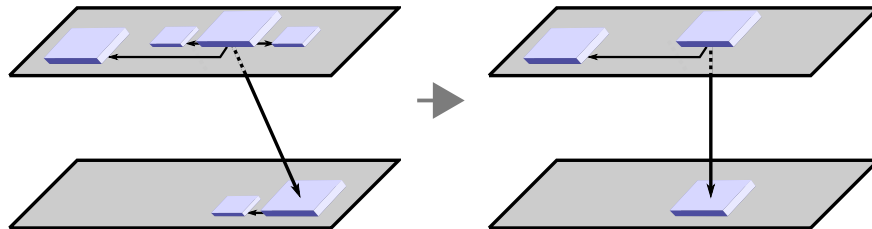
There are two variants for a termination of the algorithm: (a) terminating after a predefined number  $k \in \mathbb{N}$  of rounds or (b) measuring the difference of node positions  $\Delta$  for each round and terminate if  $\Delta < \epsilon$ , where  $\epsilon$  defines an a-priori threshold for termination, similar to the termination threshold from force-directed layout approaches (F. J. Brandenburg, Himsolt, and Rohrer 1995). Since the dependencies of dummy nodes and their position changes during the update stage might cause circular dependencies, variant (b) would not guarantee that the algorithm terminates if  $\epsilon$  is unfeasible ( $\epsilon$  is set to low). This is the case if, in every round, an update stage moves a single (counterpart) dummy node a distance greater than  $\epsilon$ . Therefore, we decided to prefer variant (a) and analyze after what number  $k$  of rounds the changes in node positions diminish without entering a cycle of dummy node dependencies, see Chapter 7. The sweep layer algorithm is thus stopped after a fixed number  $k$  of iterations (rounds), changing sweep direction after each round (when reaching the last/first layer).

Before returning the layout result after the last sweep round, we have to remove the dummy nodes and principal nodes. This is performed in a post-processing stage that we describe in the following:

Before finally removing all dummy nodes after the last round, we store their position coordinates. For a node  $n$  that is adjacent to one or more dummy nodes, we then choose the *best location* among its current position  $p(n)$  and the positions  $P = \{p(u_1), \dots, p(u_k)\}$  of the adjacent dummy nodes  $u_1, \dots, u_k$ . The best location is selected by a weighted ranking on all possible positions  $P \cup \{p\}$  in order to reduce edge lengths of edges incident to  $n$ . For each position, we compute the weighted distances to all neighbours of  $n$  in the original graph  $(V \setminus D)$ . The distances of non-layer neighbors are assigned a weight  $w_{le} \geq 1$  than neighbours in the same layer as  $n$ , see Figure 6.3. The objective of the ranking for a node  $n$  is given by

$$\min \left( \sum_{\{e=(u,w) \in E | u=n \vee w=n\}} \text{length}(e) + \sum_{\{le=(u,w) \in LE | u=n \vee w=n\}} w_{le} \cdot \text{length}(le) \right).$$

Using this placing heuristic as post-processing step, we aim at reducing layer edge lengths (**OBJ2**) by setting  $w_{le} \gg 1$  and routing them as near-orthogonally to the layer stack as possible such that the length is reduced to the distance given by *layerdistance*. In Chapter 7, we discuss benefits and issues of the layer sweep approach. One major issue, the usage of area space for dummy nodes during the sweep rounds (**OBJ3**), is addressed in the following approaches. Furthermore, in Chapter 7, we will argue that SLS might have issues to guarantee **OBJ1** (*FLOW*) due to the dependency on the underlying layout algorithm used for the layer-wise layout. In Algorithm 8, all steps of the complete algorithm for SLS are summarized.



**Figure 6.3.:** Placing heuristic for the ranking of positions in the post-processing stage where dummy nodes (small blue boxes) are removed.

---

**Algorithm 7:** Update stage for SLS: `updateLayer()`

---

**Input:** Set of dummies  $D$ , Layer  $l$ , Map *references*)

// perform update for each dummy in dummies set  $D$ .

```

1 foreach Node  $d$  in  $D$  do
2   if  $layer(d) == l$  then
3     Position  $p = p(d)$ ;
4     // get dummy node counterpart of  $d$ 
5     Node  $counterpart = references(d)$ ;
6      $pos_x(counterpart) = pos_x(d)$ ;
7      $pos_y(counterpart) = pos_y(d)$ ;

```

---

**Algorithm 8:** Sequential layer sweep (SLS)

---

**Input:** 2.5D-graph  $G(V, E, LE)$ , Iterations  $k$ , Algorithm  $LA$ , Weight  $w_{le}$

```

1 Map references;
2  $D \leftarrow \emptyset$ ; // set of dummy nodes
3  $P \leftarrow \emptyset$ ; // set of principal nodes
4 foreach Layer edge  $le = (a, b) \in LE$  do
    // create dummies  $u, w$  for  $le$ 
5    $V \leftarrow V \cup \{u\}$ ;  $layer(u) \leftarrow layer(a)$ ;  $E \leftarrow E \cup (u, a)$ ;
6    $V \leftarrow V \cup \{w\}$ ;  $layer(w) \leftarrow layer(b)$ ;  $E \leftarrow E \cup (b, w)$ ;
7    $D_{le} \leftarrow \{w\} \cup \{u\}$ ;
8   if  $layerdistance(le) > 1$  then
    // insert principal nodes to intermediate layers
9      $minlayerindex = \min(layer(a), layer(b))$ ;
10     $maxlayerindex = \max(layer(a), layer(b))$ ;
11    foreach Layer  $l$  with  $minlayerindex < index(l) < maxlayerindex$  do
12       $V \leftarrow V \cup \{p\}$ ;  $layer(p) \leftarrow l$ ; // create principal node  $p$ 
13       $V \leftarrow V \cup \{p_w\}$ ;  $layer(p_w) \leftarrow l$ ;  $E \leftarrow E \cup (p, p_w)$ ;
14       $V \leftarrow V \cup \{p_u\}$ ;  $layer(p_u) \leftarrow l$ ;  $E \leftarrow E \cup (p, p_u)$ ;
15       $P \leftarrow P \cup \{p\}$ ;  $D_{le} \leftarrow D_{le} \cup \{p_u, p_w\}$ ;
    // storing of symbolic links to counterparts
16    update map references  $\leftarrow references \cup \{le, P, D_{le}\}$ ;
17     $D \leftarrow D \cup D_{le} \cup P$ ;
18 while  $k -- \geq 0$  do
19   foreach Layer  $l$  in  $G$  do
    // see layout algorithm SDL from Section 3.1.1.
20     call layout algorithm  $LA$  on  $l$ ;
21     updateLayer( $D, l, references$ ); // see function for update stage
    on p.118.
    // ranking of positions and removal of inserted nodes.
22  $D \leftarrow D \setminus P$ ;  $V \leftarrow V \setminus P$ ;
23 Map positions = Map( $V, p(V)$ );
24 foreach Layer  $l \in L$  do
25   foreach Node  $d \in D$  with  $layer(d) == l$  do
    // get original, adjacent node of  $d$ 
26     positions(references.orig( $d$ ))  $\leftarrow positions(references.orig(d)) \cup \{p(d)\}$ ;
27      $V \leftarrow V \setminus \{d\}$ ;
    // OBJ2 for layer edges: rank position for each node using
    layer edge weight factor  $w_{le}$ .
28 positions.rank( $w_{le}$ );
29 foreach Node  $n$  in positions do
30    $p(n) \leftarrow positions(n)$  with best rank;

```

---

### 6.3. Approach 2: ILP formulation

The formulation with integer-linear-programming (ILP) for our task of computing 2.5D-visualizations for process models is an attempt to find an optimal solution with **OBJ1-OBJ3** in mind.

ILP formulations are used in several layout approaches throughout the graph drawing community, e.g. for orthogonal graph drawing incorporating constraints (Eiglsperger, Fößmeier, and Kaufmann 2000), a formulation of the crossing minimization problem (Jünger and Mutzel 1997) or for drawing metro maps (Nöllenburg and Wolff 2005).

Although ILP is NP-hard in general, it provides a preferred method in graph drawing for two reasons (Nöllenburg 2007) : (a) with ILP, one can find a 'quality benchmark for heuristics and approximations' and (b) create 'high-quality drawings for small and medium size graphs'. As for NP-hard problems, the sizes of graphs are the limit for the applicability because the maximum allowed running time might be limited. For large graphs, a ILP formulation might contain many constraints that, in total, cannot be reduced and solved by a ILP solver in guaranteed amount of time. In our case, the graphs have limited size ( $1 \leq |V| \leq 200$  and  $1 \leq |E| \leq 300$ ) because they are modeled and designed by human hand which sets a limit by humans' perception capability. This allows us to design an approach for 2.5D layout using ILP that can be solved within a predefined amount of time.

In the following, we will present our model for representing a 2.5D layout approach with ILP. Then, we will show the details of the complete algorithm of this approach.

Our approach has the above mentioned objectives **OBJ1-OBJ3** that have to be modeled to be incorporated in the objective function:

- consider sequence flow orientation (**OBJ1**): the orientation of edges should be according to the overall flow orientation of the model, for instance, respecting aesthetics *FLOW*. Edges that are directed but oriented in reverse flow orientation are penalized.
- short (layer) edge lengths (**OBJ2**): the lengths of edges should be as minimal as possible; edges that are longer than the predefined minimum edge length (that corresponds to the minimum node distance) are penalized in the objective function. The lengths of layer edges are also penalized; the penalties are multiplied by a factor since high layer edge length is considered to be more harmful than high edge length.
- good area usage (**OBJ3**): consumption of area should be reduced. A 2 : 1-ratio of width to height is desired. Note that this objective conflicts with the above objective of flow orientation if an edge is oriented orthogonally to the



flow orientation in order to reduce area consumption. Thus, we model the area usage in the following way: a start node (with given fixed position) is selected to be the *global origin* of our layout; then, the penalty for non-optimal orientation of an edge (and the resulting placement of the target) depends on the distance to the start node. The longer the distance, the higher the penalty for preferring flow orientation over area usage becomes.

Note that distances between nodes are modeled using the  $\mathcal{L}_1$ -norm (or Manhattan distance) because using linear constraints only, it is not possible to model the  $\mathcal{L}_2$ -norm (or Euclidean distance) which is defined as  $d_2(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_1)^2}$  for two points  $p = (p_1, p_2), q = (q_1, q_2) \in \mathbb{R}^2$ . The Manhattan distance  $d_1$  for two points  $p, q \in \mathbb{R}^2$  that is used in the following model is defined as follows

$$d_1(p, q) = |p_1 - q_1| + |p_2 - q_2|.$$

### Model

We will now introduce the variables of our ILP model. Also, we will denote the auxiliary boolean variables that are necessary to express different cases.

We require the following variables for a node  $v_i \in V$ : we denote its location in  $x$ - and  $y$ -coordinates with  $pos_x(v_i), pos_y(v_i)$  and  $pos_z(v_i)$  and the assigned layer by  $layer(v_i)$ . For each edge  $e \in E$  (that is not a layer edge), we define a variable  $edge\_length(e)$  for the two-dimensional Manhattan-distance between  $source(e)$  and  $target(e)$ :

$$edge\_length(e) = d_1(source(e), target(e))$$

We also add an auxiliary boolean variable  $x_1gx_2(u, w)$  that holds the order of  $x$ -coordinates:  $x_1gx_2(u, w)$  is set to true iff the  $x$ -value of  $u$  is greater than  $x$ -value of  $w$ . For each layer edge  $le \in LE$ , we define a variable  $layer\_edge\_length(le)$  that models the three-dimensional Manhattan-distance between  $source(le)$  and  $target(le)$ , for instance, including the difference of the  $z$ -coordinate values  $|pos_z(source(le)) - pos_z(target(le))|$ . For completeness of  $(layer\_edge\_length)$ , we need to define for each (layer) edge  $(u, w)$  the coordinate-wise distances  $\Delta x(u, w)$  and  $\Delta y(u, w)$ . As the distance between two adjacent layers is fixed by a constant value  $z_l$ , we express the  $z$ -distance of two nodes  $(u, w)$  by the fixed term

$$|layer(u) - layer(w)| \cdot z_l = |pos_z(source(u)) - pos_z(target(w))| \cdot z_l.$$

In order to model flow orientation, we also need to handle the case that an edge  $e$  is routed parallel to the  $x$ - or  $y$ -axis: we store an edge's  $e$  state in the boolean variables  $x_1eqx_2(e)$  and  $y_1eqy_2(e)$ :

$x_1eqx_2(e)$  is set to true iff  $pos_x(source(e)) == pos_x(target(e))$  ( $e$  is parallel to the  $y$ -axis);  $y_1eqy_2(e)$  is set to true iff  $pos_y(source(e)) == pos_y(target(e))$  ( $e$  is parallel to the  $x$ -axis).

Area minimization is achieved by the following: we select a start node  $s \in V$  (that exists always as an entry point in a valid process model) to be our *global origin* and we define, for each node  $v \in V \setminus \{s\}$ , a variable  $startnode\_dist(v, s)$  measuring its distance to  $s$ . The sum of these variables is then to be minimized.

The constraints of our ILP model now employ the defined variables to ensure the distance requirements:

- edge lengths:

$$\forall e = (u, w) \in E : edge\_length(e) = \Delta x(u, w) + \Delta y(u, w)$$

- layer edge lengths:

$$\begin{aligned} \forall e = (u, w) \in LE : layer\_edge\_length(e) = \Delta x(u, w) + \Delta y(u, w) \\ + |layer(u) - layer(w)| \cdot z \end{aligned}$$

where  $z$  is a fixed input parameter for the distance between two adjacent layers.

- area usage in terms of distance to start node  $s$ :

$$\forall v \in V \setminus \{s\} : startnode\_dist(v, s) = \Delta x(v, s)$$

Note that we only use  $\Delta x(v, s)$  for area usage. Since flow orientation is oriented in direction of the  $x$ -axis (see objective function below), we hereby strive to prevent unnecessarily long edges and high area consumption.

The objective of considering sequence flow orientation is contained in the following objective function (see Equation 6.4):

$$min : c_e \cdot \sum_{(u,w) \in E} edge\_length(u, w) \quad (6.1)$$

$$+ c_{le} \cdot \sum_{(u,w) \in LE} layer\_edge\_length(u, w) \quad (6.2)$$

$$+ c_s \cdot \sum_{v \in V \setminus \{s\}} startnode\_dist(v, s) \quad (6.3)$$

$$+ c_{dir} \cdot \sum_{(u,w) \in E} (x_1eqx_2(u, w) + x_1gx_2(w, u)) \quad (6.4)$$

where  $c_{dir}, c_{le}, c_e, c_s \in \mathbb{R}$  define the cost factors. Edge lengths are modeled in Equation 6.1 for edges  $E$  and Equation 6.2 for layer edges  $LE$ . Equation 6.3 states the area consumption as dependency to the distance from the start node and Equation 6.4

ensures edge orientation according to *FLOW*. The values for the cost factors are by derived by the priorities of the objectives **OBJ1–OBJ3**:  $c_{dir} \gg c_{le} > c_e > c_s$ .  $c_{dir}$  penalizes if edges do not obey *FLOW* (**OBJ1**).  $c_{le}$  and  $c_e$  penalize long (layer) edges (**OBJ2**), and  $c_s$  penalizes placement of nodes at high distance from the global origin, thereby reducing area size (**OBJ3**).

To ensure correctness of the ILP model during the solving process, we have to insert the following additional constraints:

$$edge\_length(u, w) \geq min_e (> 0) \quad \forall_{(u,w) \in E} \quad (6.5)$$

$$layer\_edge\_length(u, w) \geq z_l \quad \forall_{(u,w) \in E} \quad (6.6)$$

$$\Delta x(u, w) \geq 0 \quad \forall_{u,w \in V} \quad (6.7)$$

$$\Delta y(u, w) \geq 0 \quad \forall_{u,w \in V} \quad (6.8)$$

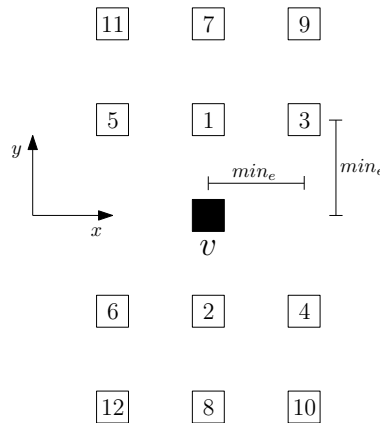
$$startnode\_dist(v, s) \geq min_e \quad \forall_{v \in V \setminus \{s\}} \quad (6.9)$$

$$pos_x(v) \geq 1 \quad \forall_{v \in V} \quad (6.10)$$

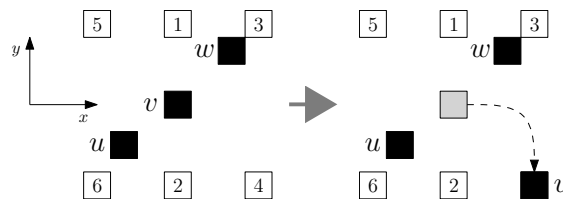
$$pos_y(v) \geq 1 \quad \forall_{v \in V} \quad (6.11)$$

**Remark** Note that we don't prevent explicitly that two nodes  $u, w$  with  $layer(u) = layer(w)$  overlap, e.g.  $pos_x(u) = pos_x(w) \wedge pos_y(u) = pos_y(w)$ , and we don't enforce minimum node distances on nodes, but for  $edge\_length$  where a minimum value  $min_e > 0$  is preset. The reason for this relaxation is the following: our above ILP formulation only contains  $O(|E| + |V| + |LE|)$  constraints; adding overlap prevention for each pair of nodes would add  $O(|V|^2)$  constraints. We note that overlaps can only occur in the case that two non-adjacent nodes overlap because, for adjacent nodes, a minimum value for  $edge\_length$  is enforced by constraints. Since Equation 6.4 strongly penalizes wrong edge orientation, overlaps are rare special cases for which we added a post-processing step that resolves overlaps: we adapted a concept presented in (Tunke-lang 1994) which locally searches for free node positions in the surrounding of the current position while enforcing minimum node distances to surrounding nodes, see Figure 6.4. This concept might enlarge edge lengths but tries to minimize the amount of enlargement by starting the search in the very local surrounding of the current position and, if no possible position was found, gradually increasing the radius of the surrounding to be searched in, see example in Figure 6.5.

Although the ILP problem is generally NP-hard, running times of ILP formulations depend on the number of constraints. In our case, the overlap relaxation causes a significant speed-up (by a factor of  $> 10$ ), see discussion in Section 7.



**Figure 6.4.:** Placing heuristic for overlaps. New possible node positions of node  $v$  are searched locally for conflicts with other nodes (in the order of given numbers). In the implementation, the numbers of positions to check are not bound by 12. In practice, this bound suffices to resolve overlaps in process models.



**Figure 6.5.:** Example for the removal of an overlap: nodes  $u, v, w$  are in a conflict set; we analyze alternative positions for node  $v$  (see numbered nodes); positions 1, 2, 3 are not available because the conflict persists. Position 4 is free and the resulting move of node  $v$  to position 4 (right) resolves this overlap.

## Algorithm

We will now summarize the algorithm for the ILP formulation. Also, the pre- and post-processing steps are described in the following. An overview of the algorithm is given in Algorithm 9.

### Pre-processing steps

1. *ensurePositiveCoordinates()*: the function shifts the coordinates of the model such that all elements have positive coordinates. This is necessary to fulfill Equations 6.10 and 6.11. If these equations are not fulfilled, the solver might not find a valid start solution because the solver doesn't have any knowledge about how to change initial variables, or more precisely, how to move node positions to obtain a valid start solution. Therefore, all node positions are shifted by vector  $v$  with

$$v = (|\min_{v \in V} pos_x(v)| + 1, |\min_{v \in V} pos_y(v)| + 1).$$

2. *integerCoordinates()*: moves all nodes to integer coordinates (except for  $z$ -coordinates which are not subject to change during the ILP) by rounding double values to integer values.
3. *selectStartNode()*: the global origin is selected by searching the node set  $V$  for a node  $s$  with  $vertex\_type(s) = START$ . If  $s$  is not unique, that is  $\exists s, s', s \neq s' \wedge vertex\_type(s) = vertex\_type(s') = START$ , a random node  $r \in S$  is selected where  $S = \{s \in V \mid vertex\_type(s) = START\}$ .

Then, the ILP solver is called with input of the ILP model that is created as described above. After the solver process returned, see Chapter 7 for technical details of the solver in use, the following post-processing steps are executed.

### Post-processing steps

1. *applySolution()*: the solution output from the solver is stored and then parsed and analyzed for variable assignments and solver statistics (solver time, number of constraints, final cost of objective function, etc.). In the variable assignments, the new positions of nodes are contained. The positions are applied to the graph  $G$ . In this step, the drawing corresponds to the ILP solver result (all positions values are identical to variables in the ILP solution).
2. *repositionNodesToOrigin()*: after the application of the ILP solution to the graph  $G$ , the shift performed in step *ensurePositiveCoordinates()* is reversed. This is achieved by adding  $-v$  to every node position.
3. *removeOverlaps()*: the rare case of overlaps is resolved here. Note that overlaps may occur only between non-adjacent nodes in the same layer. The resolution algorithm is described in Algorithm 10. For each layer, we create a set of conflict points  $P$ . A conflict point  $p \in P$  is a position of node  $v \in V$  where a node  $v' \in V \setminus \{v\}$  with position  $p'$  has distance  $d_1(p, p') < min_e$ . For each conflict point  $p \in P$ , we keep the set of nodes  $C_p$  that are affected by this conflict point. A node  $u \in V$  with position  $p_u$  is affected by a conflict point  $p$  iff  $d_1(p_u, p) < min_e$ . For each conflict point  $p$ , we then resolve the corresponding set  $C_p$  using the method presented on page 123 and described in Figure 6.4.

Note that in line 16 in Algorithm 10, we store all nodes that are resolved and, thus, these nodes cannot be participate in another conflict. Because of line 15, these nodes are not considered if they appear in a second conflict set  $C_p$ . Therefore, we prevent cycles when resolving overlaps. A resolved overlap cannot cause a subsequent (new) conflict set  $C_{new}$ .

---

**Algorithm 9:** ILP algorithm

---

**Input:** 2.5D-graph  $G(V, E, LE)$ , weights  $c_{dir}, c_{le}, c_e, c_s$

// pre-processing steps

- 1 ensurePositiveCoordinates(G);
- 2 integerCoordinates(G);
- 3 selectStartNode(G);

// create ILP model

- 4 ILPmodel = createILP( $c_{dir}, c_{le}, c_e, c_s$ );

// start solver

- 5 LPSolver.solve(ilp);

// post-processing steps

- 6 applySolution(ilp);
- 7 repositionNodesToOrigin();
- 8 removeOverlaps();

---

---

**Algorithm 10:** Removal of overlaps after solving of ILP

---

**Input:** 2.5D-graph  $G(V, E, LE)$ 

```

1 solved =  $\emptyset$ ; // store solved nodes
2 foreach Layer  $l \in G$  do
    // store all positions of nodes in this layer in a map
3   Map pos =  $\emptyset$ ; // map nodes to positions
4   foreach Node  $n \in V$  with  $\text{layer}(n) = l$  do
5      $\text{pos} \leftarrow \text{pos} \cup \text{pos}(n)$ ;
    // create set of unique positions
6    $\text{points} \leftarrow \text{pos.values}$ ;
7   Map conflictSets =  $\emptyset$ ; // map points to conflict set of nodes
8   foreach Point  $p \in \text{points}$  do
9     // find overlaps
10    foreach Node  $n \in \text{pos.keys}$  do
11      // check for equality of positions and surrounding by
12      // node size distance
13      if  $p = \text{pos}(n) \vee d_1(p, \text{pos}(n)) < \text{min}_e$  then
14         $\text{conflictSets}(p) \leftarrow \text{conflictSets}(p) \cup \{n\}$ ;
15    foreach Point  $p \in \text{conflictSets.keys}$  do
16       $cs \leftarrow \text{conflictSets}(p)$ ;
17      if  $|cs| > 1$  then
18        // fix overlaps, only if not yet resolved
19        if  $cs \notin \text{solved}$  then
20          // apply concept presented on p. 123
21           $\text{resolveOverlap}(p, cs, \text{points})$ ;
    // store resolved nodes
22     $\text{solved} \leftarrow \text{solved} \cup \text{conflictSets}(p)$ ;

```

---

### 6.4. Approach 3: Partition Supported 2.5D-Layering

Our third approach employs a hierarchical Sugiyama layering (Sugiyama, Tagawa, and Toda 1981) enriched with techniques to handle partitions (Siebenhaller 2006) and 2.5D-projections. The approach *Partition supported 2.5D-layering (PSL)* has simplicity as a major goal, in contrast to the former ILP approach where optimality was a primary goal.

The approach uses the concept of partitions which was presented in Section 2. In order to apply partitions to a 2.5D-graph, we first create a *flattened graph*. A flattened graph is a 2D-graph that has no knowledge of layers. Thus, layer edges have to be converted to regular edges. We then operate on the flattened graph but consider the former assignment of nodes to layers by creating a partitioning of the nodes such that the rows of the partition correspond to the layers, for instance, if a row  $r_i$  corresponds to a layer  $l_i$ , all nodes  $v_1, \dots, v_n$  with  $layer(v_1) = \dots = layer(v_n) = l_i$  are assigned to row  $r_i$  in the partition. Then, we start a layout algorithm that is able to handle partitions and is based on the hierarchical layout approach in (Siebenhaller and Kaufmann 2006b). As a post-processing stage, we fix the horizontal coordinates of the rows/layers such that lengths of the layer edges are minimized. Finally, we project the flattened graph to a final (unflattened) 2.5D-graph.

The approach consists of the following steps which are further described below:

1. Flattening stage for 2.5D-graph
2. Cycle removal
3. Partition creation of flattened graph
4. Sugiyama-style layering
5. Assignment of final horizontal coordinates
6. Projection to 2.5D

In the following, we will describe the steps of the algorithm in detail.

#### 1. Flattening stage:

In this stage, the 2.5D-graph  $G(V, E, LE)$  is converted to a temporary 2D-graph. The conversion creates a new graph  $G_n(V_n, E_n)$  where  $V = V_n$ . For the set  $LE$  of layer edges, we have to create representing new edges  $E_{ln}$  in  $G_n$  because  $G_n$  doesn't contain layers and therefore layer edges cannot be inserted into  $E_n$ . For each layer edge  $le \in LE$ , we add a new edge  $e_{ln}$  to  $E_{ln}$  with  $source(le) = source(e_{ln})$  and  $target(le) = target(e_{ln})$ . The set  $E_n$  is then set to  $E_n = E \cup E_{ln}$ . For reversing the flattening stage and restoring the original graph later, we store the mapping in  $flat : E_{ln} \rightarrow LE$ . Note that mapping *layer* is still applicable to  $G_n$  because  $V_n = V$ . The following steps of the algorithm work on the temporary graph  $G_n$ .



## 2. Breaking of cycles:

Since the creation of a partition on  $G_n$  and the hierarchical layout algorithm applied in step 4 require an acyclic input graph, we have to remove cycles in  $G_n$ . To find and remove cycles temporarily, we apply the algorithm for cycle removal presented in Section 3.3. It is based on performing a depth-first-search (DFS) for finding cycles. Cycle removal is achieved by reversing edges, one for each detected cycle. The reversed edges are stored in  $E_c$ .

## 3. Creating a partition:

Remember that a partition of a graph  $G'(V', E')$  is a mapping  $p : V' \rightarrow \mathbb{N} \times \mathbb{N}$  of the nodes  $V'$  to the coordinates  $(i, j)$  of a cell in a two-dimensional grid. The grid cells build parallel rows and columns; the width of the partition is defined by  $width(p) = \max\{j \mid p(n) = (i, j) \forall n \in V'\}$  and the height is given by  $height(p) = \max\{i \mid p(n) = (i, j) \forall n \in V'\}$ . A row  $i$  of  $p$  is defined by the aggregation of all cells  $(i, j)$  with  $0 \leq j \leq width(p)$ ; a column  $j$  is defined by the aggregation of all cells  $(i, j)$  with  $0 \leq i \leq height(p)$ .

In this step, we assign the nodes  $V_n$  to cells  $(i, j)$  of a partition  $p$ . Therefore, we have to define coordinates  $i$  and  $j$  (row and column) for every node. The row  $r_v = i$  of node  $v \in V_n$  is derived from the layer that contained the node in the original 2.5D-graph  $G$ , thus,  $p(v) = (i, j)$  with  $i = layer(v)$ . For the assignment of the columns, we compute an ordering of the nodes in the (acyclic) graph  $G_n$ . The ordering is found as follows: the set  $S$  of the start nodes in  $V_n$  (according to *vertex\_type*) is taken as the initial sources for a set of breadth-first-search (BFS) on  $G_n$ . For each start node  $s \in S$ , we call a BFS on  $G_n$  with  $s$  as root node. During a BFS, the visited nodes  $V_{BFS} \subseteq V_n \setminus S$  are assigned numbers that represent the level in the BFS (or the graph distance to  $s$ ). The level  $c_v$  for a node  $v$  is stored. Note that since we start  $|S|$  many BFS calls, a node  $v$  may be assigned multiple level numbers  $c_{v1}, \dots, c_{vk}$  where  $k \leq |S|$ . After the last BFS call, we then assign  $c_v$  to be  $c_v = \max\{c_{v1}, \dots, c_{vk}\}$ , the maximal level number for this node. Observe that all nodes  $V_n$  are visited by the BFS calls: if a node  $u$  is not assigned a level number after the BFS calls, it was not reachable by any node  $s \in S$ . Then,  $u$  could never be reached in the process model that is initiated by start nodes only. Therefore,  $u$  cannot be part of a valid process model and cannot exist in a process model.

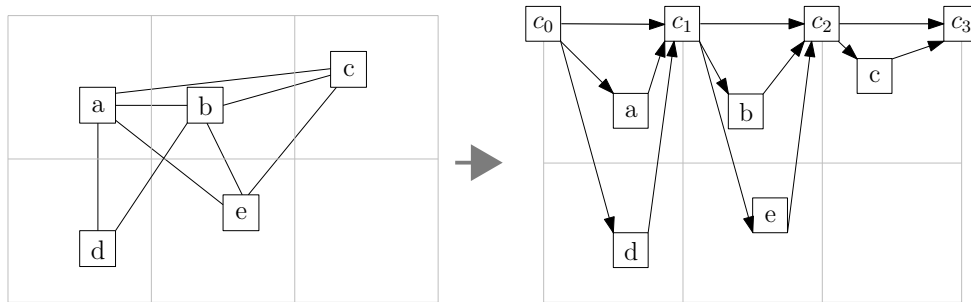
To assign a column to a node  $v \in V_n \setminus S$ , we use the level number  $c_v$ . We set the cell of  $v$  in the final partition to  $p(v) = (layer(v), c_v)$ . For the start nodes  $s \in S$ , we set  $p(s) = (layer(s), 0)$  because the root nodes of a BFS have level number 0. The worst-time complexity for creating the partition is  $O(|S| \cdot (|V| + |E|))$  where  $O(|V| + |E|)$  for one BFS run and  $|S| \ll |V|$  for the number of start nodes.

#### 4. Find partitioned layout:

In this step, we apply on  $G_n$  an extended variant of Sugiyama's algorithm to obtain an upward planarization of the input graph  $G_n$ . For this purpose, the approach in (Siebenhaller 2009, Section 4.4) is adapted to match for our process models. It supports constraints for given partitions and clusters but, in our case, we focus on partitions (not clusters) and its construction.

In the following, we depict the single phases:

- a) PHASE 1 – Layer assignment: In this phase, the first step of Sugiyama is adapted to incorporate the partition cells of the nodes in partition  $p$ . This is achieved by adding dummy nodes and temporary edges to  $G_n$ . Remember that  $width(p)$ , the maximum level number of a column, also gives us the numbers of columns that have to be preserved by dummy nodes. We now fix the columns by inserting  $width(p) + 1$  temporary nodes  $C_t$ , where a column  $j$  is represented by nodes  $c_j, c_{j+1} \in C_t$ . Also, we add temporary edges  $E_C$  such that a node  $c_j \in C_t$  is connected to all nodes of  $G_n$  in column  $j$ , that is  $V_{c_j} = \{v | c_v = j, v \in V_n \setminus C_t\}$ , and edges from nodes  $V_{c_j}$  to  $c_{j+1}$ . The structure of the resulting graph is depicted in Figure 6.6. For the layering, a heuristic GT layering is applied (Eiglsperger and Kaufmann 2001). After this phase, each node is assigned to a (Sugiyama) layer. Also, we obtain as a return parameter the set of edges  $E_l$  that were reversed during the layering in order to ensure that the graph  $G_{nc} = (V_{nc}, E_{nc}) = (V_n \cup C_t, E_n \cup E_C)$  remains acyclic (using a DFS for cycle detection inside the GT layering).



**Figure 6.6.:** Insertion of temporary nodes  $C_t = \{c_0, \dots, c_{width(p)}\}$  and edges  $E_C$  (black edges in (b), original edges are hidden) to preserve columns during Sugiyama layering.

- b) PHASE 2 – Edge preparation: This phase converts the edges for preparation of the crossing minimization. It is called the *normalization phase* because of the following normalization of edges:

Each edge  $e \in E_{nc}$  has to fulfill the following requirements:

- i.  $e$  is upward, e.g.  $e$  does not span from a layer  $i$  to a layer  $i - k, k > 0$ .
- ii.  $e$  has unit-length, e.g.  $e$  does not span from a layer  $i$  to layer  $i+k, k > 1$ .
- iii.  $e$  is an inter-layer edge, e.g.  $e$  does not connect two nodes  $u, w \in V_{nc}$  in the same layer  $i$ .

To ensure the last requirement (condition iii), edges that connect nodes in the same layer are temporarily removed before the next phase (crossing minimization). Edges with non-unit-length (condition ii) are split, using a dummy node for each intermediate Sugiyama layer of a *long* edge and replacing the original edge  $e$  with the chain of dummy nodes  $V_d(e)$  and unit-length edges  $E_d(e)$ . Edges that are not upward (condition i) are temporarily reversed. Changes performed on the edges in this phase are saved in data maps for later restoration of original edges.

- c) PHASE 3 – Position assignment: The crossing minimization phase, or *one-sided two-layer crossing minimization*, reduces the crossings between any two adjacent Sugiyama layers in a layer-by-layer sweep. The phase corresponds to the step of the Sugiyama approach for crossing reduction. In order to preserve the partitions during the crossing minimization, the phase adapts the sweep approach as follows (Forster 2002, 2005; Siebenhaller 2009): Let  $G' = (L_1 \cup L_2, E' \subseteq L_1 \times L_2)$  denote a two-layered graph and  $L_2^i \subseteq L_2$  the set of nodes assigned to the  $i$ -th column, i.e., the nodes  $v \in L_2$  with  $c_v = i$ . We apply the crossing reduction approach separately to the subgraphs  $G'_i$  induced on  $G'$  by the nodes of  $L_1 \cup L_2^i, 0 \leq i < width(p)$ . We process the subgraphs  $G'_i$  in increasing order of  $i$  and concatenate the resulting node orders to obtain the order of  $L_2$ . It is shown that we can separately calculate the node order for each subgraph without losing quality in  $O(|V||E| \log |E|)$  for a graph  $G = (V, E)$  (Siebenhaller 2009)
- d) PHASE 4 – Node placement: This phase assign the  $y$ -coordinates to the nodes  $V_{nc}$ . We use the linear-time algorithm of (Brandes and Köpf 2001) which is a longest path-based heuristic, see Section 2. For constructing the partition, we apply the following: Let  $G_l = (V_l, E_l)$  denote the directed acyclic graph resulting from the layer ordering calculated during the (last) crossing reduction phase. Now, we have to align the dummy nodes  $u_i^{c_j} \in C_l \cup V_d(E_{nc}), 1 \leq i \leq k$  for each column  $c_j, 0 \leq j \leq width(p)$ ,  $k$  is given by number of dummy nodes inserted during normalization, to obtain the vertical grid lines of the partition. This is achieved by mapping all nodes (including temporary nodes inserted by normalization) that should

be aligned to a single node of  $G_l$  such that all nodes representing one column grid line have the same  $y$ -coordinate. After this phase, dummy nodes and temporary edges inserted during normalization are removed. Original edges in  $V_{nc}$  that were removed are now restored and added to  $E_l$ . Edges that were reversed are not yet recovered in their orientation before the next (and last) phase.

e) PHASE 5 – Edge routing: In the final phase, we reroute the edges of  $G_l$ . For each edge, we perform a shortest path–routing on the dual graph, see Section 2. However, we have to restrict the dual graph such that the partition is considered in the routing. Therefore, we introduce two requirements for edge routing in a partitioned graph:

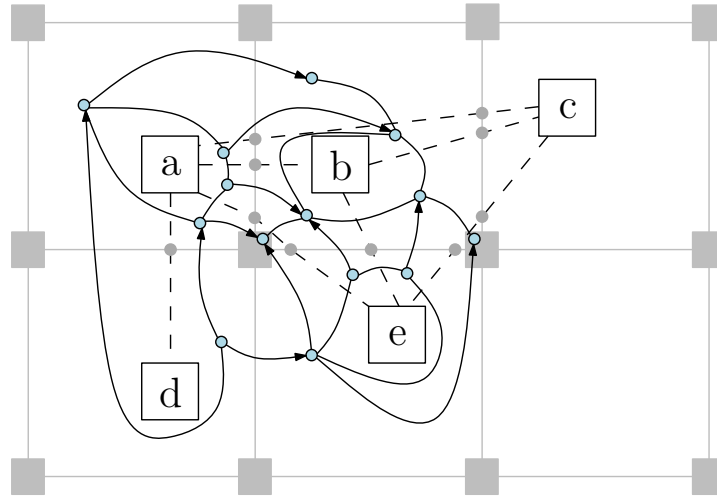
- i. Each route of an edge  $(v, w) \in E_l$  is completely contained inside the smallest rectangle that surrounds partition cells  $p(v)$  and  $p(w)$ .
- ii. An edge is allowed to cross a cell border at most once.

Note that for two nodes  $u, w$  with  $p(u) = p(w)$ , an edge  $(u, w)$  fulfilling the above requirements is not allowed to leave the cell  $c = p(u) = p(w)$ . This prevents unnecessary crossings and higher edge lengths by simple dual graph routing, compared to routing in an un-partitioned graph.

The necessary requirements are guaranteed by modifying the dual graph  $D_{G_l} = (V_{G_l}, E_{G_l})$  of  $G_l$  that is used for edge routing. For the first requirement, for an edge  $e = (u, w)$ , we construct the smallest possible rectangle *rect* that surrounds the cells  $p(u), p(w)$ . Nodes in  $D_{G_l}$  that represent faces outside of *rect* are removed. To guarantee requirement (ii), we direct the edges in  $D_{G_l}$  as follows: assuming  $d_e$  is the (Manhattan) distance between  $p(u)$  and  $p(w)$ . Then, an edge  $e \in E_{D_{G_l}}$  is undirected if source and target of  $e$  are in the same cell, i.e. the distance to  $p(w)$  is not changed if routing over  $e$ . However, if source and target of  $e$  are in different cells  $q, v$ ,  $e$  is directed such that routing over  $e$  from  $p$  to  $q$  reduces the distance of  $d_e$  by 1 (towards  $p(w)$ ), see Figure 6.7. Edge routing can be done in  $O((|V| + x)|E|)$  where  $x$  denotes the number of crossings in  $G_n$ . Remember that crossings were temporarily replaced by dummy edges.

Finally, dummy nodes  $C_l$  inserted for preserving the partition are removed and edges  $E_c$  that were reversed for making the graph acyclic (for the Sugiyama layering) are recovered to the original orientation.

After the extended Sugiyama layering, we obtain a graph  $G_l$  that is embedded as a 2D partitioned layout: all nodes are positioned in their layers such that *FLOW* is considered because we assigned the nodes into the partitions cells according to their BFS-order in the previous step. Since the layering in Sugiyama style



**Figure 6.7.:** Shortest path routing in the (modified) dual graph  $D = (V_D, E_D)$  for the example graph of Figure 6.6 and the rerouting of edge  $e_d = (d, b)$ . The partition is depicted by grey lines and squared grey nodes. Edges of the original graph are represented by dashed black lines. Dummy nodes are drawn as small grey circle shaped nodes. Nodes of the dual graph are drawn as blue circle shaped nodes. Note that an edge  $e \in E_D$  (solid black lines) is directed only if it points to a cell that has lower distance to the target's cell of  $e_d$  than the source's cell of  $e$ . Otherwise,  $e$  is undirected and can be traversed in both directions for rerouting.

of this step preserved the partition, *FLOW* is not violated. The BFS order expresses high parallelisms in process models because parallel elements (started or initiated at the same single element) are then placed in the same column  $c$  in  $p$  (having the same number from the BFS order). Remember that elements in identical columns are not moved apart in the layering due to the inserted dummy nodes bounding the columns in the crossing minimization.

We now rearrange the rows and columns of the partition such that the layout of  $G_l$  is prepared for 2.5D.

##### 5. Assignment of final horizontal coordinates:

From the last step, we obtain a layout considering partition  $p$  for the 2D-graph  $G_l$ . In this step, we arrange the rows of the partition  $p$  such that the rows can be re-mapped to layers and the original layer edges can be recovered in the next step.

Note that layer edges in  $G_n$  correspond to edges connecting two rows  $i, k$ ,  $i \neq k$  in  $G_l$ . To reduce the average length of layer edges in the later projection to 2.5D, we now align the elements in the rows of  $G_l$  in terms of  $y$ -coordinates. Therefore,

we compute the overall median  $M_y$  of the  $y$ -coordinates of the nodes  $V_l$ . We also compute the row-wise average mean of  $y$ -coordinates  $m_1, \dots, m_{\text{height}(p)}$ , where

$$m_i = \text{mean} \{y | y = \text{pos}_y(v) \wedge p(v) = (i, j), 0 \leq j \leq \text{width}(p), v \in V_l\}.$$

Note that for the rows, we compute the mean, and not the median, because we want single nodes in a row  $i$  with high distance to the rest of the nodes in  $i$  (peak nodes) to contribute to the final  $y$ -coordinate  $m_i$  of row  $i$ . Taking the median would diminish the effect of the peak nodes to  $m_i$  because the median is defined such that points with high variance are neglected to reduce the resulting variance and preserve robustness in the presence of outliers (Hogg, McKean, and Craig 2012).

We now set the distance vector for each row  $i$  to

$$d_i = \begin{pmatrix} 0 \\ M_y - m_i \end{pmatrix}.$$

The vector  $d_i$  gives us the direction and amount of the translation for the alignment of the rows. The translation for a node  $n$  in row  $i$  and its position  $(\text{pos}_x(n), \text{pos}_y(n))$  is then given by

$$\begin{pmatrix} \text{pos}_x(n) \\ \text{pos}_y(n) \end{pmatrix} := \begin{pmatrix} \text{pos}_x(n) \\ \text{pos}_y(n) \end{pmatrix} - d_i = \begin{pmatrix} \text{pos}_x(n) \\ \text{pos}_y(n) - M_y + m_i \end{pmatrix}.$$

The assignment of final horizontal coordinates can be done in linear time. In the last step, we will project the aligned graph to 3D to obtain the final layout result.

#### 6. Projection onto 2.5D-stack:

After the last step, every node  $n \in G_l$  has a geometric  $(x/y)$ -location. These locations are final. In this step, we will project the nodes into 3D by assigning each node a  $z$ -coordinate and recover the layers  $L$  from the input graph  $G$ . Also, the layer edges  $LE$  are restored. This step is also called the *un-flattening stage*. Note that the assignment *layer* is still valid for all nodes  $V_l$  in  $G_l$  because all temporarily inserted nodes were removed after the Sugiyama layering. Therefore, we recover the original layers by assigning each layer  $l \in L$  a  $z$ -coordinate  $z^l$  that is recovered from the flattening stage.

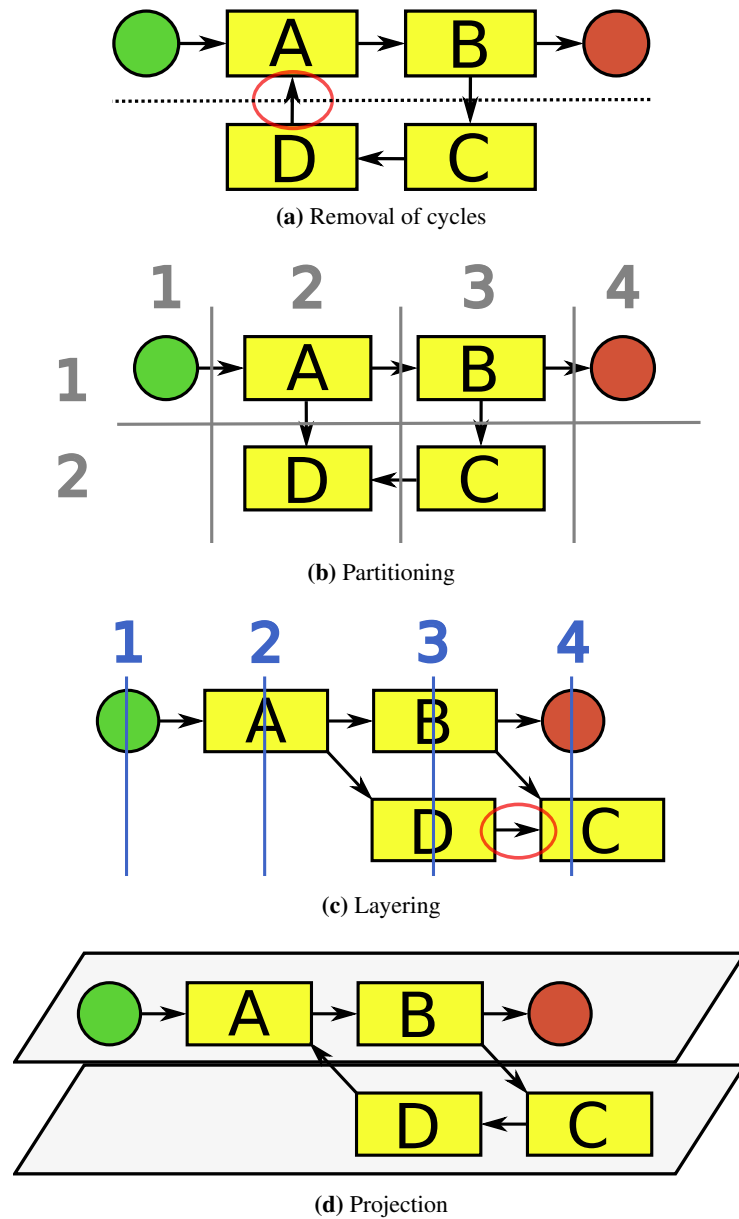
However, if the predefined distance  $z_d$  between each pair of (adjacent) layers in the stack is changed (by user input), the set of layers is translated in direction of  $z$ -axis such that  $z_d$  is maintained for each pair of adjacent layers.

Also, the rectangles representing the layers in the stack are updated because the position of the nodes  $G_l$  after the layout step are different from the input positions in  $G_n$ . We compute the smallest rectangle  $r$  (sides of  $r$  are parallel to  $x$ - and  $y$ -axis) that contains all positions  $(\text{pos}_x(v), \text{pos}_y(v))$  of nodes  $v \in V_l$  by setting

$$\begin{aligned}
p_{min}^x &= \min \{pos_x(v) | v \in V_l\}, \\
p_{max}^x &= \max \{pos_x(v) | v \in V_l\}, \\
p_{min}^y &= \min \{pos_y(v) | v \in V_l\}, \\
p_{max}^y &= \max \{pos_y(v) | v \in V_l\}.
\end{aligned}$$

Rectangle  $r$  is then set to the rectangle induced by  $(p_{min}^x, p_{min}^y, p_{max}^x, p_{max}^y)$ . Then, for each layer  $l \in L$ , the rectangle of  $l$  is set to  $r$  with the  $z$ -coordinate  $z_l$ .

Now, the layers are arranged in a 2.5D-stack and we obtained our final 2.5D-graph layout. In Algorithm 11 on page 137, the algorithm for PSL is summarized with an overview on the different steps. The steps of PSL applied on a simple process model example are depicted in Figure 6.8.



**Figure 6.8.:** Steps of PSL with a small process model (swimlanes are depicted by dashed black line). Numbers ( $x$ ) describe the corresponding stage in the algorithm: (a) Flattened graph after *flattening stage* (1); removal of cycles (edge marked in red is reversed) (2). (b) Partition assignment of nodes (grey lines) using BFS order for columns and layer assignment for rows (3). (c) Sugiyama layering (layers are represented by blue lines, dummy nodes for preserving columns and rows or for normalization of edges are omitted here) (4). (d) Alignment of horizontal coordinates of rows (5) and un-flattening stage with final projection onto 2.5D-stack (6).



**Algorithm 11:** Partition supported 2.5D-layering

---

```

Input: 2.5D-graph  $G(V, E, LE)$ 
// Flattening of graph  $G$ 
1  $G_n = (V_n, E_n) \leftarrow (V, E \cup LE)$ ;
2  $E_c \leftarrow DFS\_acyclic(G_n)$ ; // Cycle removal
// Create partition  $p$ 
3  $BFS\_order(V_n) \leftarrow BFS(startnodesS \subseteq V_n)$ ;
4 foreach  $v \in V_n$  do
5    $p(v) = (i, j) \leftarrow (layer(v), BFS\_order(v))$ ;
// Sugiyama-style partitioned layering
// 1. layer assignment
6  $C_t \leftarrow$  partition  $p$  preserving column nodes;
7  $E_C \subset (C_t \times V_n)$  connecting edges for  $C_t$ ;
// 2. create partition in layering
8  $E_d(E_n), V_d(E_n) \leftarrow$  normalize( $E_n$ ); // normalize edges
9  $E_{nc} \leftarrow E_n \cup E_d(E_n) \cup E_C \setminus$  non-normalized edges  $E_{nm} \subset E_n$ ;
10  $V_{nc} \leftarrow V_n \cup C_t \cup V_d(E_n)$ ;
// 3. crossing minimization
11 one-sided two-layer crossing minimization ( $G_{nc} = (V_{nc}, E_{nc})$ );
// 4. horizontal coordinate assignment
12  $G_l = (V_l, E_l) \leftarrow BrandesKoepf(G_{nc})$ ;
13  $E_l \leftarrow (E_l \setminus E_d(E_{nc})) \cup E_{nm}$ ;
14  $V_l \leftarrow V_l \setminus V_d(E_{nc})$ ;
// 5. rerouting edges
15 partitioned-shortest-path-routing( $D_{G_l}, p$ );
16  $E_l \leftarrow E_l \setminus E_C$ ;
17  $V_l \leftarrow V_l \setminus C_t$ ;
// assigning of final horizontal coordinates
18  $M_y = median\{y | y = p_y(v), v \in V_l\}$ ;
19 foreach layer  $l_i \in L$  do
20    $m_i = mean\{y | y = p_y(v) \wedge p(v) = (i, j), 0 \leq j \leq width(p), v \in V_l\}$ ;
21   foreach  $v \in V_l, layer(v) == l_i$  do
22      $d_i = M_y - m_i$ ;
23     shift  $v$  by  $\begin{pmatrix} 0 \\ -d_i \end{pmatrix}$ ;
// Projection to 2.5D-layer-stack
24 foreach  $v \in V_l$  do
25   set  $p_z(v)$  to  $z_l$  of layer  $l = layer(v)$ ;
26  $E_l \leftarrow E_l \cup LE$ ; // restore layer edges
27 compute layer rectangle  $R$  containing all  $pos_x(v), pos_y(v), v \in V_l$ ;
28 draw layer stack with  $z_d$  as layer distance;

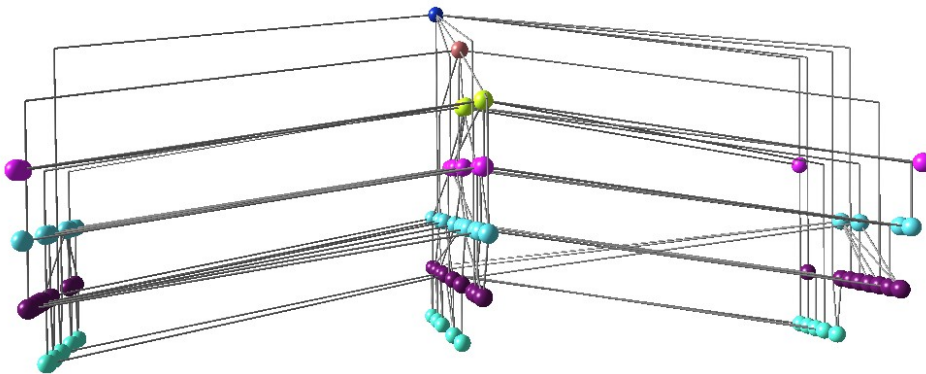
```

---

### 6.4.1. A 2.5D hierarchical drawing of directed graphs

In this section, we will describe the approach *HONG* of (Hong, Nikolov, and Tarassov 2007) to compute hierarchical drawings for directed graphs in 2.5D. Also, we will compare HONG to PSL and point out the distinctions in terms of objectives.

HONG introduces an extra step to the Sugiyama framework, called *wall assignment*, that is inserted into the framework after the layer assignment step and before the computation of vertex ordering. The idea is based on *walls* that occupy parallel planes which are perpendicular to the planes of the layers. Walls partition the vertex set of a layer into subsets; each of these subsets represents one wall. In Figure 6.9, an example for a graph with layout by HONG is depicted.



**Figure 6.9.:** Example for layout with HONG. The layers are vertical and the walls are orthogonal to the layers. Nodes of the same layer are depicted by identical color. Source: <http://rp-www.cs.usyd.edu.au/~visual/valacon/gallery/3DHL/>, 2012–09–30.

The work in (Hong, Nikolov, and Tarassov 2007) presents five algorithms that each tackle a subset of the following criteria:

- **C1.** Even distribution of vertices among walls, i.e., balanced partition of the vertex set into walls.
- **C2.** Minimum number of *inter wall edges*; an inter wall edge is an edge that is incident to two nodes in different walls).
- **C3.** Minimum number of crossings between inter wall edges in the projection of the drawing into a plane which is orthogonal to both the layer planes and the wall planes, i.e. direction of view in a 3D–environment).
- **C4.** Minimum total edge length of inter wall edges.

The first proposed algorithm is targeted at **C1** and **C2**. It is a greedy heuristic algorithm for the minimum bisection problem which is known to be NP-hard (Garey and Johnson 1979). The problem solved by the heuristic is the following:

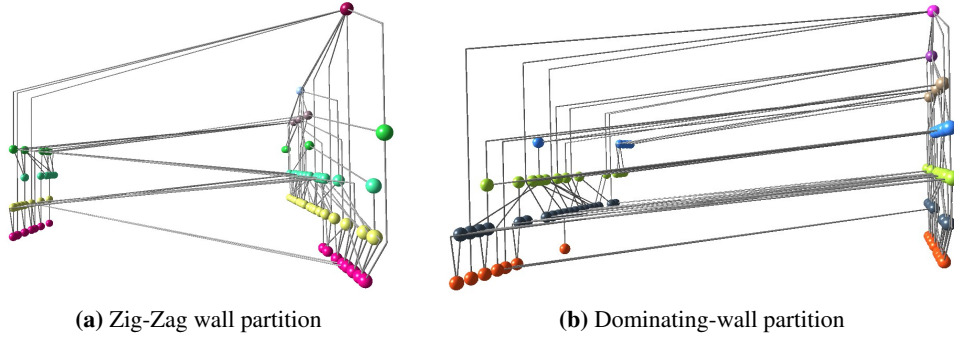
**Definition 13 (Minimum One-Layer Bisection Problem).**

Consider two adjacent layers  $L_{i-1}$  and  $L_i$ . Let  $L_{i-1}$  be partitioned into subsets  $A_{i-1}$  and  $B_{i-1}$ . Find a partition of  $L_i$  into subsets  $A_i$  and  $B_i$  such that  $||A_i| - |B_i|| \leq 1$  and the number of edges between  $A_{i-1}$  and  $B_i$  plus the number of edges between  $A_i$  and  $B_{i-1}$  is the minimum.  $\square$

Note that this problem only considers two walls  $A, B$ . The greedy algorithm given for the problem has two phases and runs in  $O(|V| \log |V| + |E|)$ ; at the first phase each vertex in layer  $L_i$  is assigned to the wall that contains the biggest number of its immediate successors; at the second phase some vertices are moved from one wall to the other in order to achieve a balanced partition. In PSL, the columns correspond to walls in HONG, because they both are oriented perpendicularly to the layers. In contrast to walls in HONG, the number of columns is not limited in PSL. The given algorithm tries to compute a balanced distribution among the walls. In our case, the distribution to columns is induced by the BFS runs in the stage of creating the partition. Our approach is motivated by fulfilling *FLOW* and follows the natural way of thinking 'forward' (from left to right) when analyzing the 2.5D-graph.

Also, **C2** aims at reducing occlusion in the 3D-space. When inspecting the inter wall edges in Figure 6.9, it seems that occlusion is still an issue because it is hard to distinguish two adjacent nodes connected by an inter wall edge. Inter wall edges correspond to layer edges in PSL. Although the number of layer edges  $|LE|$  is not limited, two layer edges  $l_1, l_2 \in LE$  can only cause occlusion in PSL if there exists parallelism in the business process that require  $l_1$  and  $l_2$  to start and end in the same cell in the partition. Since only splitting gateways can initiate parallelisms in business process models, the number of causes for occlusion in PSL is limited by at most the number of gateways  $|G|$  where  $|G| \ll |V|$ .

To satisfy **C3**, HONG proposes two approaches: zig-zag wall partition (ZZ) or dominating wall-partition (DW). Both algorithms scan all layers one by one from bottom to top and partition each of them into two subsets. They start with a random balanced partition of the first layer. Each next layer  $L_i$  is partitioned into  $L_i^1$  and  $L_i^2$  such that  $L_i^1 \cup L_i^2 = L_i$  and  $L_i^1 \cap L_i^2 = \emptyset$  based on the partition of layer  $L_{i-1}$ . Both, ZZ and DW call the procedure `DIVIDELAYER( $i, x, y$ )` which sets  $L_i^x \leftarrow \{v \in L_i : N^+(v) \cap L_{i-1}^y = \emptyset\}$  and  $L_i^y \leftarrow L_i \setminus L_i^x$ ,  $N^+$  denotes the successors of a node. In ZZ, the sources of the inter wall edges alternate between two layers on a path of inter wall edges, whereas in DW, the sources of inter wall edges are in the same wall, see Figure 6.10. In other words, both algorithms assign vertices to two walls, thus, this approach is also limited to two walls. Although both algorithms do not explicitly consider **C2**, it appears from Figure 6.10 that the assignment to two walls according to ZZ or DW increases the negative effect of occlusion and make a distinction between two inter wall edges very hard.



**Figure 6.10.:** Example for zig-zag wall partition and dominating-wall partition in HONG. The vertices are distributed to two walls. Nodes of the same layer are depicted by identical color.

Source: <http://rp-www.cs.usyd.edu.au/~visual/valacon/gallery/3DHL/>, 2012-09-30.

For support of  $k \geq 2$  walls and to satisfy **C4**, the sum of total edge lengths of inter wall edges, HONG proposes the following algorithm *k-wall partitioning*: Similar to the algorithms described above, all the layers are scanned one by one from bottom to top. The first layer is partitioned randomly and each next layer  $L_i$  is partitioned on the basis of the partition of layer  $L_{i-1}$ . For partitioning layer  $L_i$  into  $k$  subsets such that the total edge length of inter wall edges is small, the following is applied for each vertex  $u \in L_i$ : all its immediate successors are considered, and  $u$  is placed in the wall whose number is the closest integer to the average of the wall numbers of the immediate successors of  $u$ . In other words, the wall  $u$  is placed in the barycenter of the walls its immediate successors are placed in. When  $k = 2$ , this algorithm is basically equal to the first proposed algorithm for Minimum One-Layer Bisection Problem. To guarantee a balanced distribution of vertices among the walls (according to **C1**), the author describe a simple technique that computes for each vertex a barycenter value  $b$  that alternates for giving preference to the walls with fewer number of vertices. For a vertex  $u$ , the wall assignment is computed by the barycenter  $b_u$  that is given by

$$b_u = \left\lfloor \frac{\sum_{j=1}^k j \cdot \max\{0, \text{neighbours}[j] - |L_i^j|\}}{\sum_{j=1}^k \max\{0, \text{neighbours}[j] - |L_i^j|\}} + 0.5 \right\rfloor.$$

The underlying problem of balanced partitioning with  $k \geq 2$  subsets and the minimum number of edges between the subsets is a generalization of the minimum bisection problem and, therefore, it is NP-hard. The given algorithm for  $k$ -wall partitioning has time complexity  $O(|V| + |E|)$  because each vertex and each edge is scanned only once.

The test data set for HONG is taken from the ROME graph library (Di Battista et al. 1997) with limited graph size,  $10 \leq |V| \leq 100$ . Also, the direction of edges is assigned randomly.

We can conclude that HONG provides a general framework for 2.5D-visualizations of directed graphs. However, for our application domain of business process models, the given algorithms do not comply with the requirements of 2.5D-visualizations for business process models. None of the algorithms given in (Hong, Nikolov, and Tarassov 2007) fulfills all requirements **C1-C4**. For business process models, the requirements can be discriminated as follows:

- **C1** (equal distribution of nodes) is not desired for business process models. In PSL, we use metadata of the process models, e.g. organizational semantics from swimlanes, and encapsulate them into constraints for the 2.5D-visualization by creating a partition. The partition reflects the metadata of swimlanes and follows aesthetics *FLOW*.
- **C2** (minimum number of inter wall edges) attempts to reduce occlusion. When analyzing the given figures from HONG, this requirement still seems to be an issue. In business process models, we argued above that the number of occlusions is limited by the number of splitting gateways  $|G|$ . And for real-world process models, it holds that  $|G| \ll |V|$ . Therefore, occlusion is a small threat for 2.5D-visualizations for business process models. Also, the number of layer edges is induced by the choice of the perspective, see Section 5.3.1.
- **C3** (minimum number of crossings of inter wall edges) is tackled only for two walls by HONG. In our case, business process models are sparse. Crossings between layer edges are very rare and the reduction of crossings has not highest priority.
- **C4** (minimum number of total edge length of inter wall edges) is also a requirement for layer edges in 2.5D-visualizations of business process models. With our approach ILP from Section 6.3, we attempted to find an optimal solution for 2.5D-visualizations with respect to total edge length of all edges. Also, PSL explicitly attempts to minimize total edge lengths by assigning cells to nodes in the partition according to node order in BFS and assigning optimized final horizontal coordinates.

Concluding, the algorithms of HONG do not suffice to fulfill the requirements of 2.5D-visualizations of business process models. However, they provide a useful collection of algorithms that tackle NP-hard problems for partitioning into subsets with constraints. The algorithms are simple and provide fast heuristics.

In the next chapter, we will analyze and benchmark our three approaches SLS, ILP and PSL. We will compare performance and layout quality. Also, we will compare them against a force–feedback approach for 2.5D.

## Analysis and Benchmarks

After the description of the approaches for *2.5D*-visualizations of business process models in the last chapter, we will now present the analysis of the approaches and the results of benchmark tests.

We will at first describe the properties of the data set of business process models that is used for the benchmark tests, and then, we evaluate the approaches empirically with respect to performance and layout quality. Finally, we will discuss the strengths and weaknesses of the approaches in detail.

### 7.1. Data set

For our test data set, we decided not to use random graphs or collections of random graphs, e.g. Rome graphs (Di Battista et al. 1997) or Graph Catalog (also known as the AT&T Graphs) (Di Battista et al. 2000). Instead, we preferred a collection of real industrial business process models because our approaches are targeted to work with process models. A collection of random graphs might not reflect the graph structure of industrial process models and might produce misleading results when used in benchmark tests.

The real-world business process models, which we used, stem from a collection created for soundness checking of industrial business process models (Fahland et al. 2009). The origins of the data are manifold:

Quotation from (Fahland et al. 2009):

These data mostly resulted from modeling activities in customer projects within a SOA context, i.e., processes were captured with the final goal of implementing them in a Service-Oriented Architecture. The models covered various industry domains such as financial services, automotive,

telecommunications, construction, supply chain, health care, and customer relationship management. We also looked at large collections of reference processes that were created for the insurance and banking domain by users who explored different modeling styles, i.e., different ways of capturing data and control-flow at varying level of granularity. All models were available in the IBM WebSphere Business Modeler tool [...]

Many process models are in fact quite small, as good modeling practice suggests an appropriate structuring of processes into subprocesses, and are therefore not a challenge for our soundness-checking approaches. Others, in particular those created in other tools, might not have been created with the appropriate notion of soundness or might have been created by non-experts and consequently turned out to be syntactically incomplete and therefore flawed in such a way that it made no sense to consider them further. In the course of our experimental studies, we therefore reduced our initial test set of approx. 3000 models to 5 libraries of 735 different models in total from the insurance, banking, customer relationship, as well as construction and automotive supply chain domains.

The data files with the process models were provided on the web and, eventually, contained 748 process models. The models were completely anonymized, i.e., description of tasks or gateways were relabeled to  $t_1, t_2, \dots$ , and  $g_1, g_2, \dots$  respectively. The anonymization had no drawback for our purpose of visualizations because we are interested only in the structural information of the underlying graphs and in predefined node types for the mapping *vertex\_type*, which were not removed by the anonymization procedure.

For the mapping *swimlane*, we have to create an assignment of the nodes to swimlanes because the mapping is not given by the data set. Since we want to prevent a purely random assignment, we apply Algorithm 12 that sequentially assign swimlanes to nodes depending on the type and the neighbourhood. For the maximum number of swimlanes that are available for the assignment, we analyzed a repository of existing processes (Kunze et al. 2011) and we found that the number of swimlanes  $|S|$  ( $= |L|$ ) grows sublinear to  $|V|$ , i.e.,  $|S| < o(|V|)$ . Therefore, we set the maximum number of swimlanes for the assignment to  $|S| = o(\sqrt{|V|})$ .

The test set of 748 real process models (Fahland et al. 2009) had the following graph properties: graphs are very sparse with an average node degree of 2.2. The distribution of the number of nodes and the number of edges is depicted in Figure 7.1.

In order to increase the validity of our benchmark results, we use an implementation of Fruchterman-Reingold (FR) (Fruchterman and Reingold 1991) for comparison. The implementation of FR was adapted for 2.5D by restricting forces to act only in the layers ( $z$ -coordinates of nodes are fixed). In the following, we abbreviate this implementation by *Spring*.



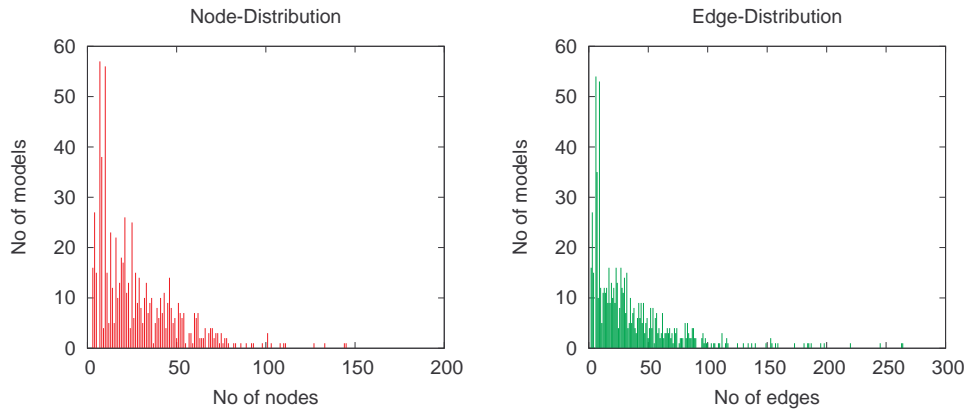
**Algorithm 12:** Swimlane assignment for models in test data set

---

**Input:** Map *vertex\_type*, Graph *model(V, E)*

- 1 *RandGen* = random integer generator for  $[0, \dots, \sqrt{|V|}]$ ;
- 2  $N_u = \emptyset$ ; // set of unassigned nodes
- 3 **foreach**  $v \in V$  **do**
- 4     **if**  $\text{vertex\_type}(v) == \textit{task} \vee \text{vertex\_type}(v) == \textit{subprocess}$  **then**
- 5          $\text{swimlane}(v) = \text{RandGen}()$ ;
- 6     **else**
- 7          $N_u \leftarrow N_u \cup \{v\}$ ;
- 8
- // handle unassigned nodes (gateways/events first)
- 9 **foreach**  $v \in |N_u|$  **do**
- 10     **if**  $\text{vertex\_type}(v) == \textit{gateway} \vee \text{vertex\_type}(v) == \textit{event}$  **then**
- 11          $N_n \leftarrow$  neighbourhood of  $v$ ;
- 12         **foreach**  $n \in N_n$  **do**
- 13             // if a node  $n$  in neighbourhood of  $v$  is assigned
- 14             **if**  $\text{swimlane}(n) \in [0, \dots, \sqrt{|V|})$  **then**
- 15                 // swimlane of  $v$  is set to swimlane of  $n$
- 16                  $\text{swimlane}(v) \leftarrow \text{swimlane}(n)$ ;
- 17                  $N_u \leftarrow N_u \setminus \{v\}$ ;
- 18                 **break**;
- 19
- // handle remaining unassigned nodes
- 20 **while**  $|N_u| > 0$  **do**
- 21     **foreach**  $v \in |N_u|$  **do**
- 22          $N_n \leftarrow$  neighbourhood of  $v$ ;
- 23         **foreach**  $n \in N_n$  **do**
- 24             // if a node  $n$  in neighbourhood of  $v$  is assigned
- 25             **if**  $\text{swimlane}(n) \in [0, \dots, \sqrt{|V|})$  **then**
- 26                 // swimlane of  $v$  is set to swimlane of  $n$
- 27                  $\text{swimlane}(v) \leftarrow \text{swimlane}(n)$ ;
- $N_u \leftarrow N_u \setminus \{v\}$ ;
- break**;
- 

---



**Figure 7.1.:** Graph properties of test set data. Industrial business process models have limited size in terms of elements ( $\max(|V|) = 145$ ) and connections ( $\max(|E|) = 264$ ).

The four approaches are compared quantitatively (analyzing running times) and qualitatively (layout quality). Note that all our new approaches from Chapter 6 also consider *FLOW*. However, the reference implementation Spring does not consider edge orientations.

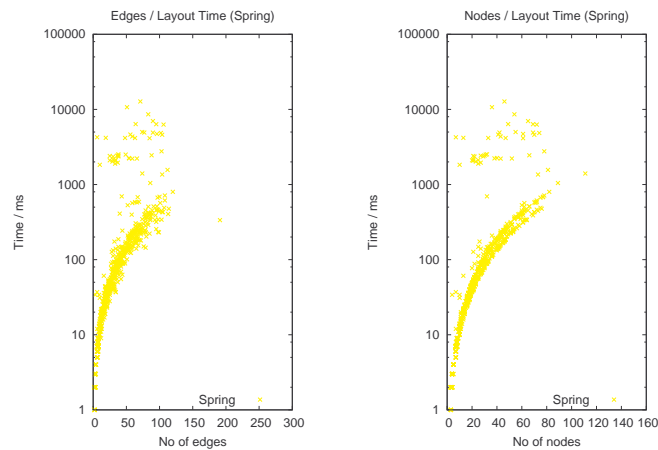
All tests were run on a Intel® Core™2 Quad CPU Q9300 with 2.50GHz and 3GB of RAM. The operating system is *Ubuntu 10.04 LTS - Lucid Lynx*. The algorithms were implemented in JAVA™ 1.5. For solving the linear programs of the ILP approach, we used the solver SCIP (Achterberg 2009). The weights for the objective function are set as follows:  $c_e = 5.0$ ,  $c_{le} = 10.0$ ,  $c_s = 0.25$  and  $c_{dir} = 120.0$ . The values represent the priority of the objectives stated in Section 6.3. In SLS, we ran 5 iterations of sweeps, e.g. sweep orientation was flipped five times. Thus, the number  $k$  of Section 6.2 was set to 5. In pre-experiments, we evaluated that changes of node positions  $\Delta$  was close to 0 for all test process models after five flips of direction in the sweep algorithm. For each model and approach, we ran 3 iterations to eliminate variance due to limited computing hardware resources and parallel calls of the underlying operating system. The test runs resulted in total number of  $\approx 9000$  data sets.

## 7.2. Performance

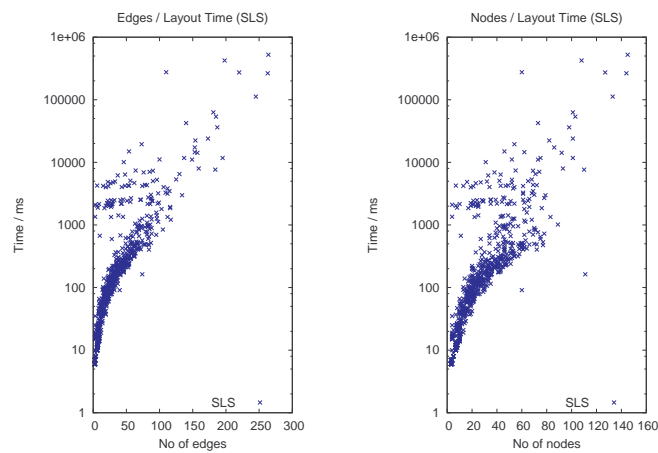
Execution times were recorded per model instance and the total time of each algorithm was measured, including the pre- and post-processing stages. We depict the times in relation to the number of nodes and in relation to the number of edges, see Figures 7.2 - 7.5. We remark that the data structures of the algorithms of Spring did not support multi-edges. Multi-edges were removed in a pre-processing stage and, therefore, and the maximum number of edges of the test graphs might differ in the following diagrams.

The aggregated diagram for execution times of all approaches is depicted in Figure 7.6. ILP was aborted after 120 seconds if no optimal solution was found, or after 20.000 LP iterations without improvement of the objective function. For models with more than 180 edges, this abortion was critical as Figure 7.4 indicates.

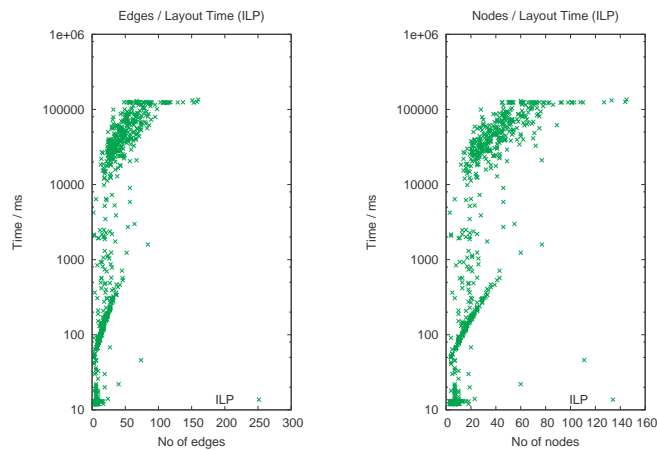
It is clearly obvious that Partition supported layering (PSL) outperforms the other approaches, see Figure 7.6. Execution times of Sequential layer sweep (SLS) is ranked between ILP and Fruchterman-Reingold (Spring), see also discussion in Section 7.4.



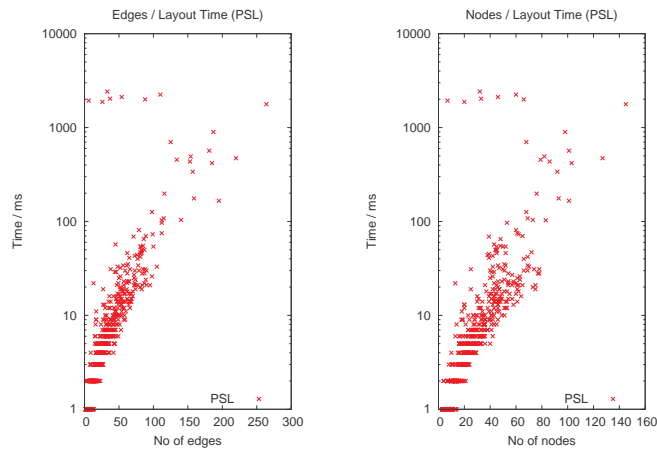
**Figure 7.2.:** Execution times for the reference implementation of Fruchterman-Reingold (FR).



**Figure 7.3.:** Execution times for SLS.



**Figure 7.4.:** Execution times for ILP.

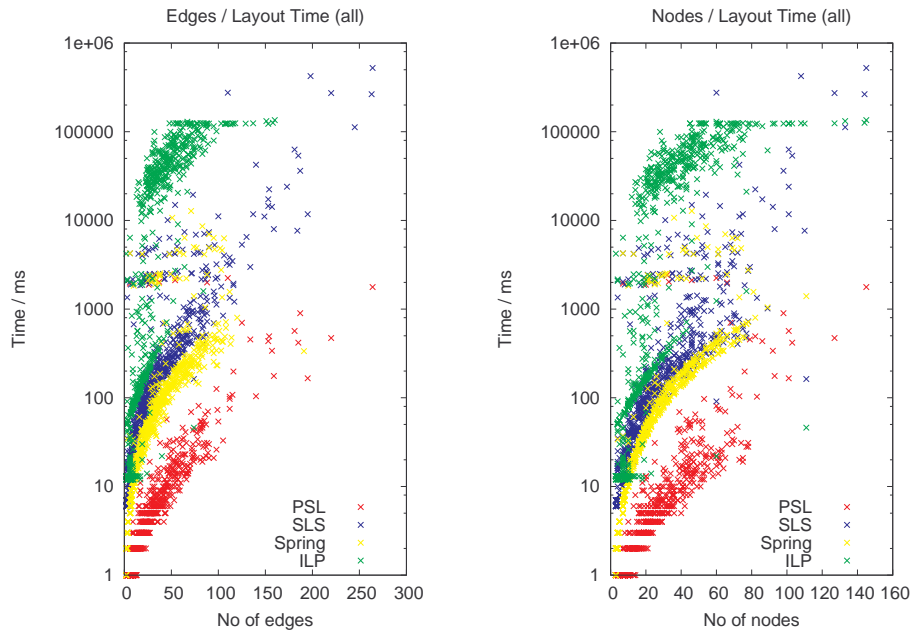


**Figure 7.5.:** Execution times for PSL.

### 7.3. Layout Quality

For evaluating the quality of the approaches according to **OBJ1-OBJ3**, we consult the edge lengths and the area size of the resulting layouts. In Figures 7.7 - 7.10, we depict maximum and average edge lengths for each of the approaches and, in Figure 7.11, we present the aggregated diagram for the data set of all four approaches.

We decided to consult maximum edge lengths and average edge lengths as metrics for the benchmark measurements for the following reason: Aesthetics *FLOW* (**OBJ1**) is fulfilled if an edge  $e \in E$  of a graph  $G = (V, E)$  is oriented in the orientation of the flow of the process model. Also, *FLOW* implies that the nodes incident to an edge are positioned such that the edge is easy to follow. Therefore, the length of an edge  $e$



**Figure 7.6.:** Comparison of aggregated execution times.

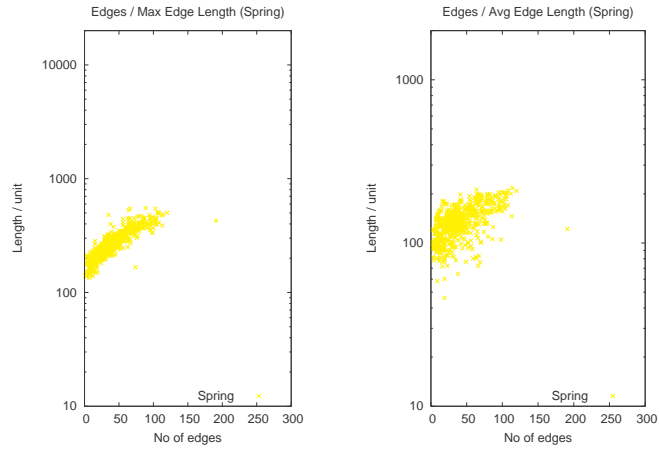
should as short as possible. Thus, average edge length of a process model is desired to be as low as possible. However, the average edge length does not suffice to prevent the following: many short edges  $E_s \subseteq E$  but a few single edges  $E_l \subseteq E \setminus E_s$  that result to be very long in the final layout, i.e., layer edges, because routing of the short edges  $E_s$  causes unfortunate routing of long edges  $E_l$ . This case is observed in our benchmarks by measuring the maximum edge length for each process model. The *good* case in terms of layout quality is a low average edge length but not at the expense of high maximum edge length (**OBJ2**).

Although SLS produced high maximum edge lengths, average edge length was only outperformed by PSL for small graphs. Note that ILP did not produce measurable results for larger graphs ( $|E| > 180$ ) due to timeouts. Remember that Spring could not handle multi-edges between two nodes which explains the missing points of results for  $|E| > 170$ .

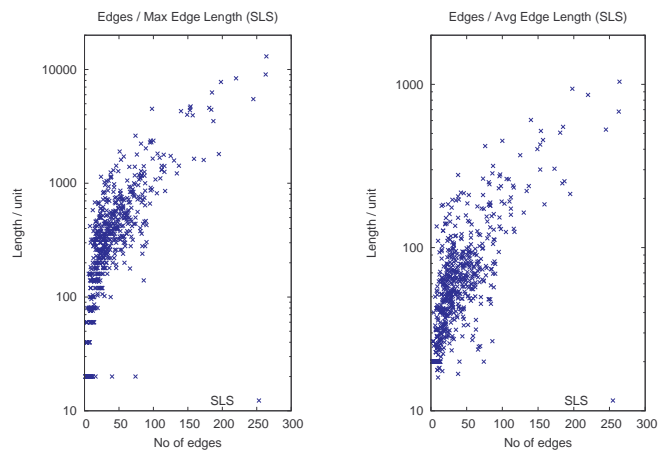
In Figure 7.12, we present the aggregated data for area consumption of the four approaches (**OBJ3**). ILP produced few outliers for graphs with  $|V| < 100$  with very area size. This is attributed to the fact that not all process model instances could be solved optimally within given time constraints, as discussed in Section 7.4.2.

For a visual presentation of the resulting visualizations of our approaches, we refer to an example in Figure 7.16. Also, we provide a high-resolution video which enables the reader to analyze the results interactively in an animated fashion in the environment of *BPMN-Layouter*. The video can be found online at:

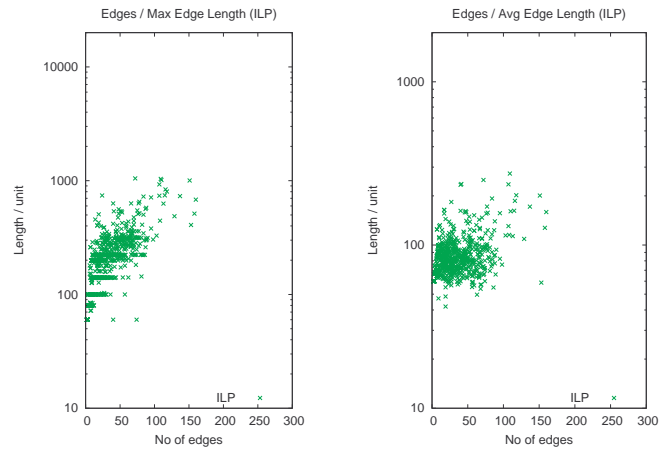
<http://algo.inf.uni-tuebingen.de/?site=forschung/graphenzeichnen/bpmn-layouter>



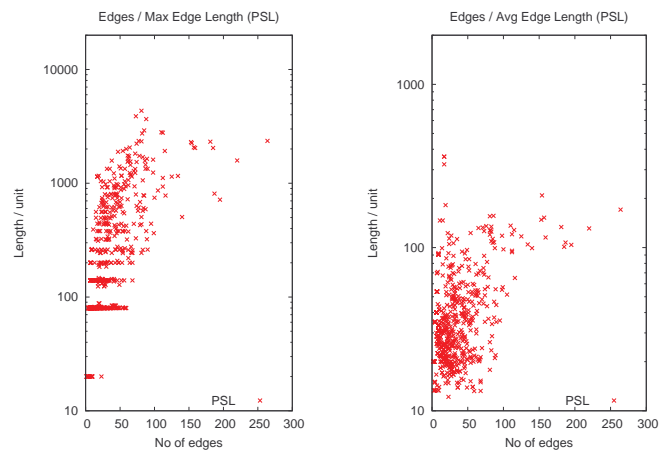
**Figure 7.7.:** Maximum/Average edge lengths for SLS.



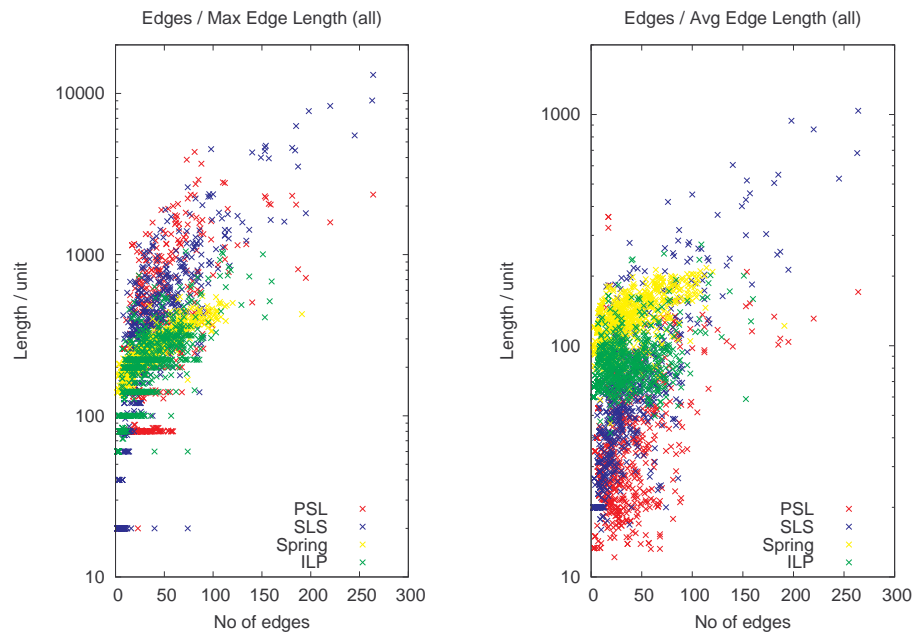
**Figure 7.8.:** Maximum/Average edge lengths for SLS.



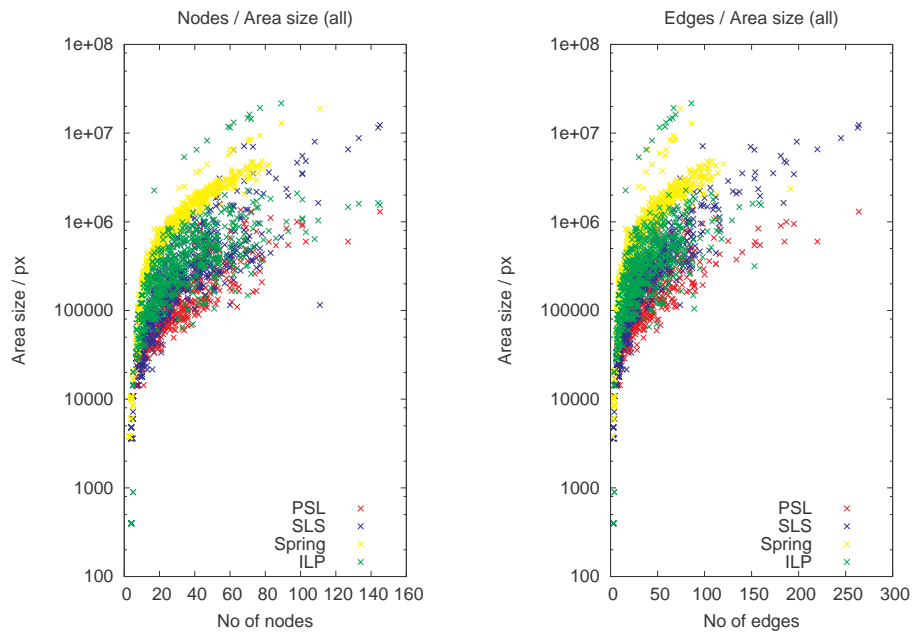
**Figure 7.9.:** Maximum/Average edge lengths for ILP.



**Figure 7.10.:** Maximum/Average edge lengths for PSL.



**Figure 7.11.:** Aggregated data for maximum/average edge lengths of all four approaches.



**Figure 7.12.:** Aggregated data for area consumption of all four approaches.



## 7.4. Discussion

In the following, we will discuss the results of the benchmarks and algorithmic details of the three approaches SLS, ILP and PSL.

### 7.4.1. Sequential layer sweep (SLS)

A drawback of SLS is that it may end up in a state where no changes are performed when sweeping through the layer although optimality (in terms of edge lengths) is not reached. This case occurs if the dummy nodes (that are set in (a) the update layer stage and (b) layer layout stage) block the layout stage of a layer to utilize the area space for regular nodes. The issue of space squandering by inserted dummy nodes is a major drawback for SLS and can be observed in the resulting high (maximum and average) edge lengths in Figure 7.8, especially for graphs with  $|E| > 50$ , and the area consumption in Figure 7.12. Remember that the number of dummy nodes  $|D|$  is  $|D| \leq 3 \cdot |LE| \cdot |L|$  (dummy nodes and principal are represented by at most three inserted nodes per layer and per layer edge).

Another issue of SLS is that a node  $n$  (incident to a layer edge) may oscillate between two positions: caused by connected dummy node(s) and the update layer stage that sets the dummy node(s) in the neighbourhood of  $n$ , a call of the layout stage in layer  $layer(n)$  may reset the position  $n$  and therefore render the previous update layer to be without effect on the position of  $n$ . This was solved by setting  $k$ , the number of sweeps, and breaking oscillating dependencies of nodes incident to layer edges. However, it cannot be guaranteed that the final position of  $n$  is optimal.

Executions times of SLS are high, see Figure 7.3 due to multiple calls of the layer layout algorithm for each layer (including dummy nodes). The layer layout algorithm in SLS is called in total  $|L| \cdot k$  times for each process model with layers  $L$ .

### 7.4.2. Integer linear programming (ILP)

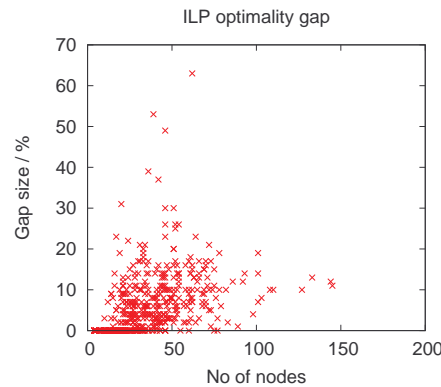
The ILP approach has a very high running time for larger process model instances ( $|E| > 180$ ); for smaller examples, the rise in running time with increasing number of edges is still steep, see Figure 7.4. ILP accomplishes best the task of reducing the maximum edge lengths. However, average edge lengths are mediocre compared to SLS and PSL, see Figure 7.11.

Of 748 tested models, 323 (= 43%) were solved optimally, the remaining 425 process model instances caused an LP solver interruption due to the above mentioned constraints: solving time was limited to 120 seconds and solving was interrupted after 20.000 LP iterations without improvement on the objective function. Although, the

ratio of aborted ILP calls is high (57%), the average optimality gap for these calls is low (9.59%), the maximal optimality gap was 63.18%, see Figure 7.13. The gap is defined by the current (relative) gap given by

$$gap = |primal - dual| / \min(|dual|, |primal|)$$

where  $|primal|$  and  $|dual|$  are the values of the objective function in the solving process of the primal linear program and the dual linear program, respectively. This shows that, although 57% of the process models could not be solved optimally within solving constraints, the remaining steps in the linear program to reach an optimal solution of the ILP layout problem for these models were small. The resulting layouts of ILP are also satisfying for larger process models, despite an aborted solving process.



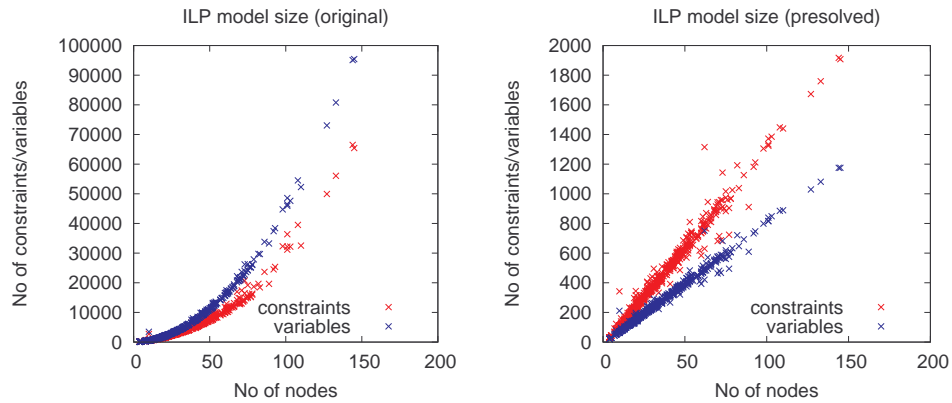
**Figure 7.13.:** Optimality gap of ILP solving processes. Gap values  $> 0\%$  stem from aborted LP solver processes due to time or improvement constraints.

Remember that the ILP model contained relaxed constraints with respect to node overlaps. In tests, we found that removing the overlap relaxation resulted in process models that were not solvable in less than 2 hours. Therefore, the relaxation produces a significant speed-up and allows us to compute layouts in practically acceptable time constraints.

The post-processing stage for node overlap removal was called in special cases only: for small process models ( $|V| \leq 50$ ) the overlap removal stage counted an average number of overlaps of 2. For larger process models, the maximum number of overlaps in a process model was 6. Note that node overlaps cause high costs in the objective function as described in Section 6.3.

Remember that an ILP model for a process model instance has  $O(|E| + |V| + |LE|)$  constraints, see Section 6.3. In Figure 7.14, we depict the sizes of the test model instances for the original models (as described in Section 6.3) and the presolved model sizes. Presolving is a step of SCIP that is performed before the solving process is initiated.

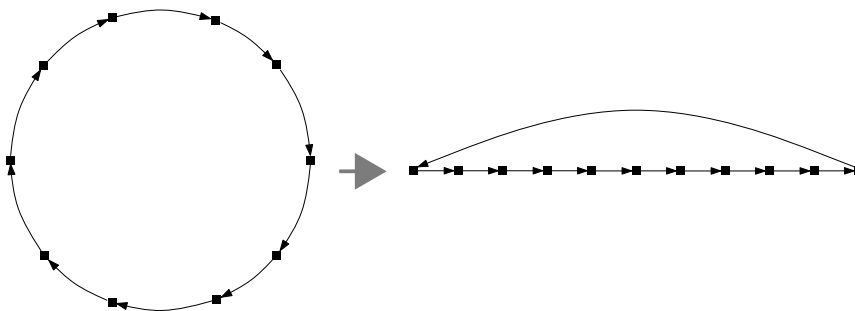
Presolving attempts the following goals: domain tightening, coefficient modification, deletion of redundant constraints, constraint upgrading (Achterberg 2009). From Figure 7.14, we observe that presolving has a significant effect on our ILP model instances in terms of deletion of redundant constraints.



**Figure 7.14.:** Sizes of the ILP models for the process model instances in the benchmarks. The models were created according to description in Section 6.3.

### 7.4.3. Partition supported layering (PSL)

From Figure 7.6, we can observe that PSL has the lowest execution times and outperforms the four approaches. This is a major benefit of PSL. In terms of layout quality, PSL produces low average edge lengths and consumed lowest area space, see Figure 7.12. However, PSL creates higher maximal edge length for graphs with  $|V| \leq 100$  compared to Spring and ILP. This can be attributed to the upward layering. Edges that are reversed for cycle removal can result in a very long at the benefit of short edges in the remaining edges of the cycle, see example in Figure 7.15.



**Figure 7.15.:** Layering of a cycle causes (at least) one long edge in the layer assignment.

The resulting layouts of PSL are very pleasing. A benefit of PSL is that parallelisms in the process model are visualized in a manner that they are easy to perceive. This is achieved by using the BFS for the assignment of partition cells: tasks that are executed in parallel are assigned the identical BFS number (when initiated by the same gateway). Therefore, they are assigned the identical column  $j$  in partition  $p$  and placed in parallel in the resulting layout.

The reason that PSL outperforms the other approaches in terms of performance and layout quality is based on the fact that PSL exploits the graphs' structure of the business process models: the underlying graphs are sparse and contain few cycles. Thus, the assignment into a partition is a pre-processing stage that allows the underlying layering algorithm to be fast and simple, i.e., compared to the model of linear programming. We also use PSL in our project *Flight Navigator* that is presented in Section 8.3.

## 7.5. Conclusion and Summary

Concluding the analysis and benchmarks of the proposed approaches for 2.5D-visualizations of business process models, we can observe that our approaches SLS, ILP and PSL are feasible and produce satisfying layouts.

From Figures 7.6, 7.11 and 7.12 and the discussion in the previous section, we can conclude that

1. PSL computes the resulting layouts with the best performance.
2. PSL consumes the lowest amount of area for the resulting layouts.
3. PSL has the lowest average edge lengths.
4. PSL achieves lowest average edge lengths at the expense of higher maximum edge lengths.
5. PSL exploits the graphs' structure of business process models best.
6. SLS is unable to cover the issue of space usage by inserted dummy nodes.
7. ILP produces optimal results (with respect to the objectives given in Section 6.3).
8. ILP solves only 43% of the instances in the test data set within a given time frame of 120 seconds, despite the relaxation of node overlaps.

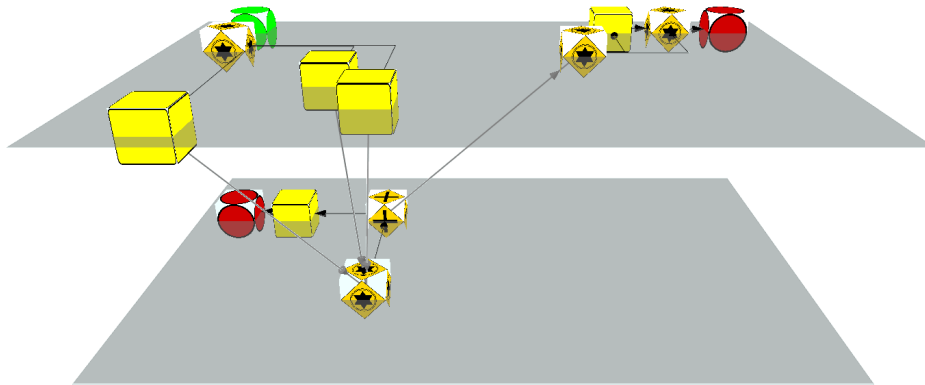
Furthermore, we presented perspectives for the projection of business process models into the 3D-space. These perspectives are a requirement for semantically expressive 2.5D-visualizations of business process models. The perspectives are:

- Organization perspective
- Functional perspective
- Control flow/Data flow perspective

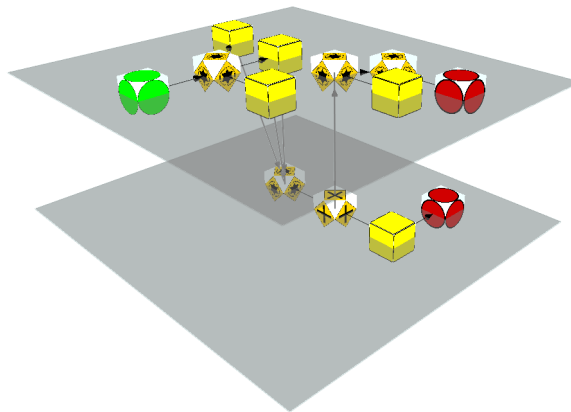
We employ the perspectives in our three approaches for 2.5D-visualizations.

---

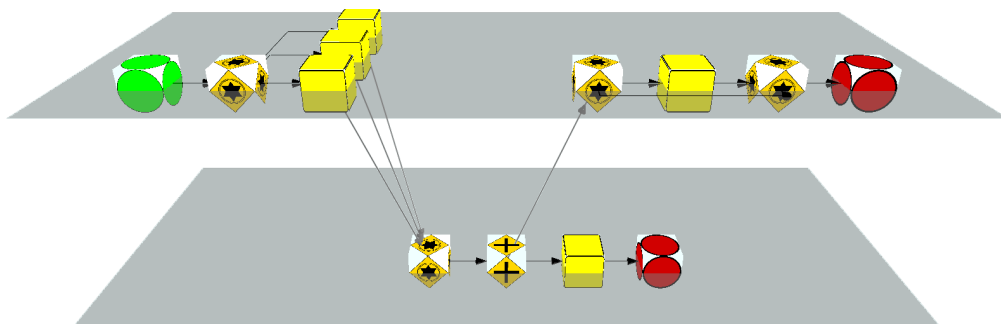
The algorithms for *2.5D*-visualizations are integrated into *BPMN-Layouter*. Therefore, we developed a *3D*-framework for the presentation and analysis of *2.5D*-visualizations. This so-called *3D-Navigator* provides highly interactive user interfaces with navigation support in *3D* and real-time presentation of *2.5D*-visualizations. A use case of the *3D-Navigator* is the *Flight-Navigator* for business process models that is presented in Section 8.3.



(a) SLS visualization. It is obvious that, in this example, SLS is not able to guarantee *FLOW*, see lower layer. Also, placement of nodes is far from optimal considering area consumption which is due to the temporarily inserted dummy nodes.



(b) ILP visualization. The approach has a good area consumption. Also, layer edge lengths are optimal in this example. Note that the approach considers the minimum node distance which is given as input parameter. (The view is rotated by  $45^\circ$  around the  $z$ -axis.)



(c) PSL visualization. *FLOW* is highlighted best for this example. Note that the concurrency of the three parallel tasks is easily identifiable. Layer edge lengths are not optimal but *FLOW* is stressed further because layer edges are oriented in flow direction.

**Figure 7.16.:** Examples of 2.5D-visualizations with our approaches. The process from Figure 2.16 on page 47 is used as input model.

**Part III.**

**Epilog**





## Applications and Projects

In this chapter, we would like to present additional projects that were undertaken during the time of this thesis. All projects are related to visualization and were published and/or presented at conferences or workshops in the field of graph drawing or process visualization. Among the projects are attempts to visualize business process models, either in an online-application on the web, see Section 8.2, or in an interactive browsing mode in *3D*, see Flight-Navigator in Section 8.3. Also, there is a project which the graph drawing community benefits from: an online archive for graphs (GraphArchive) is presented in Section 8.4. I would like to start with a project of personal interest, the visualization of the relations between the most important composers since the 10th century.

### 8.1. Contribution to GraphDrawing2011 Contest

In parallel to the annual International Symposium of Graph Drawing, the organization committee calls for participation in the GraphDrawing Contest. The tasks given to participants are of the following two categories:

(a) problems in graph drawing that are known to be hard to solve optimally, e.g. layout of a graph with given restricted area size, or (b) visualize given data in a sense that the underlying data becomes perceptible, e.g. if nodes represent cities, they might be placed according to location on the globe.

In 2011, the following task was published in category (b):

The composers graph is a large directed graph, where the nodes represent Wikipedia articles about composers, and the edges represent links between these articles. The graph has too many nodes and edges to be effectively presented in a straightforward way. The task is to combine graph

drawing algorithms with appropriate techniques for complexity reduction (such as filtering and varying the graphical attributes) to create an illuminating visualization (one or more images, possibly with commentaries, or a movie). It is by no means a requirement to present the entire data set.

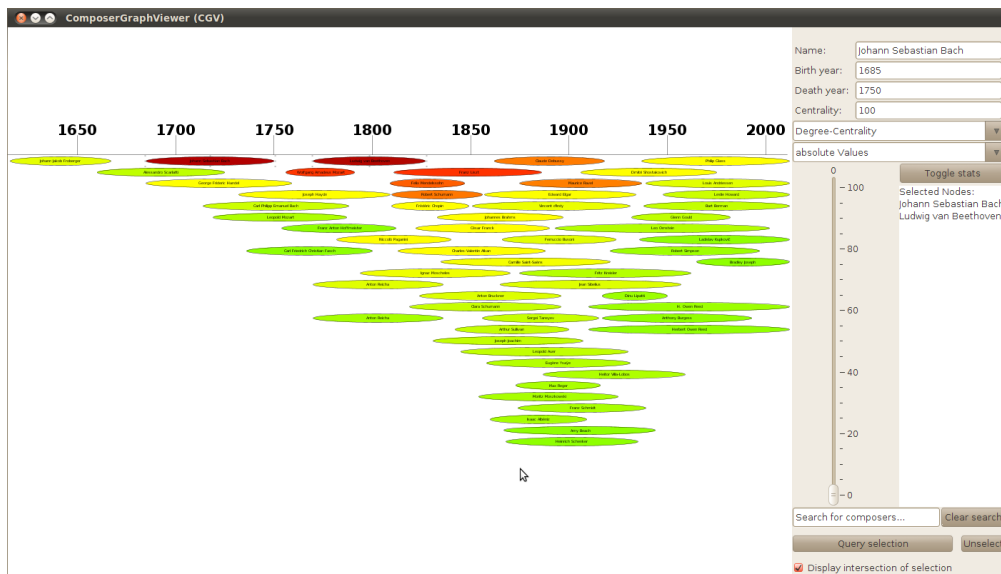
Source: homepage of GraphDrawing Symposium 2011:

<http://graphdrawing.de/contest2011/topic2-2011.html>, 2012–09–30.

Our contribution to the GraphDrawing Contest 2011 consists of a dedicated application (ComposerGraphViewer, CGV) that allows interactive browsing and filtering of the composers graph. CGV is also mentioned on the homepage of the contest results, see <http://graphdrawing.de/contest2011/results.html>, 2012–09–30.

Furthermore, we created a video for the presentation of the interactive features, see <http://www.youtube.com/watch?v=8b5z4nLL8KM&hd=1>, 2012–09–30.

In the following summary, we present the ideas and features that are incorporated in our contributed tool CGV.



**Figure 8.1.:** A screenshot of the graphical user interface for the CGV. The timeline is drawn on top of the composers nodes. Colors represent centrality values of nodes in the composers graph (from 'red' nodes that denote high centrality to 'green' nodes with low centrality).

### 8.1.1. Preprocessing of the graph

The following steps were applied to the graph prior to a visualization in CGV. The main goal was to reduce the huge amount of indistinguishable data contained in the graph.

- **Component analysis:** Analysis revealed 623 components (622 components with size  $\leq 3$ ). These small components were removed and the connected remaining graph (with approx. 2750 nodes) was processed further.
- **Retrieval of birth dates:** For the remaining nodes, composer names were used to search for birth dates of composers in wikipedia articles. Birth dates were found for 2684 composers.
- **Computation of centrality values:** Centrality values of nodes are computed and visualized using colors in the range from red (high centrality) to green (low centrality). Centrality computations can be done with two centrality measures: degree- or node-betweenness-centrality.
- **Assign default coordinates:**  $x$ -coordinates of nodes are given by birth dates. Composer node length is set relatively to the years of a composer's life.  $y$ -coordinates are initially set to 0, see layouter in next section
- **Edge removal:** Due to performance reasons, edges are prevented from being displayed but will be considered in filtering features, see Section 8.1.3.

### 8.1.2. Layout implementation

Our layout algorithm applies a layered layout to the nodes. Nodes are inserted sequentially in descending order of centrality. Initiating with the timeline and a single empty layer, the algorithm searches for the next (to top, and to timeline respectively) unused area space in the given life span of the composer. If no layer contains free space, a new layer is appended to the layout.

It is possible that a node  $a$  with lower centrality appears higher in the display than a node  $b$  with high centrality since the algorithm looks for an area next to the time-line and if  $a$  has a shorter life span than  $b$ , it might be inserted in a higher layer.

The number of layers is bound by the maximum of composers that live at a time in any period of time. In our case, the maximum numbers of layers is occupied by alive composers in the present.

### 8.1.3. Filtering features

Filtering is a powerful feature when displaying massive amount of data. Our visualization approach supports three different filters:

**Algorithm 13:** Layering layout algorithm

---

```

1 Let  $V_u$  be the set of nodes;
2  $V_s = \text{Sort } V_u$  in descending order of (degree-) centrality value;
3 Let  $L = \emptyset$  be the ordered list of layers;
4 for Node  $v$  in  $V_s$  do
5   let  $t_v = (\text{birth}, \text{death})_v$  be the life time period of  $v$ ;
6   for Layer  $l$  in  $L$  do
7     if  $l$  has free space for  $t_v$  then
8       insert  $v$  to  $l$ ;
9       break;
10    // if no layer contained free space
11    create new layer  $l_n = \{v\}$ ;
12     $L = L : l$ ; // append new layer to list of layers
13 return  $L$ ;
```

---

- **Incidence selection:** incident edges of selected nodes are taken into account: only neighbours of selected nodes are displayed. When multiple nodes are selected, the displayed nodes can be set to be the union or intersection of the set of neighbour nodes.
- **Centrality:** a lowest centrality bound can be set by using an interactive slider in the application. The slider defines the lowest bound for the centrality of a node to be displayed. Distribution of centrality can be linear (same number of nodes in each slider unit) or absolute (ranges according to centrality values). Degree- and node-betweenness-centrality are available to interactively swap between the two types of centrality.
- **String search:** a string filter enables the search for composers' names.

The filters can also be combined. Using the combinations lead to many interesting use-cases that allow to inspect the graph in detailed manner, e.g., the following analysis: what composer(s) has interacted with what other composer(s) that have at least the importance (in the sense of centrality) of a given value?

#### 8.1.4. Interactive application

Our application comes with a small graphical interface 'ComposerGraphViewer' (CGV) for user interactivity, see Figure 8.1. Important parts of the interface are the following:

- **Statistics:** a window for statistics can be enabled. It displays the total number of composers currently displayed, as well as the chronological predecessors,

contemporary colleagues and successors of one or more selected composers.

- Switch of applied centrality type (degree- or node-betweenness).
- Update after slider changes with layout computation in real-time.
- Text search for names in real-time.
- Selection box for union or intersection when applying the selection filter.

The user can discover surprising connections between the composers and analyze them further since the filters can interactively be changed and update the current graph immediately. By not displaying the huge number of edges (> 13.000), the application runs smoothly and allows real-time interaction with the user. A switch of centrality type needs, as a one-time pre-processing step, between 2-7 seconds per computation for the whole graph.

## 8.2. Business Process Modeling using Web2.0

The second project<sup>1</sup> provides automatic layout algorithms that are incorporated in a Web2.0 modeling platform. The project was a joint work with Dr. Gero Decker, a former PhD student at Hasso-Plattner-Institut (HPI) Potsdam, Germany. He later founded Signavio GmbH<sup>2</sup>, a company providing software tools for business process modeling.

Modern business process modeling can be facilitated using automatic layout techniques. Business process modeling accomplishes the task of designing processes in a graphical manner. With Oryx, we have a open-source modeling tool at hand that supports collaborative and web-based modeling of BPMN diagrams. Here, we show how automatic layout of diagrams can support the designer when starting to model a process in BPMN. We provide an automatic layout approach integrated into Oryx that computes a new layout for a given BPMN diagram. When constructing a new layout, BPMN drawing conventions have to be considered, e.g. orthogonal edges, hierarchical structures, partitions, etc.

### 8.2.1. Oryx – A Web2.0-based collaborative graphical editor

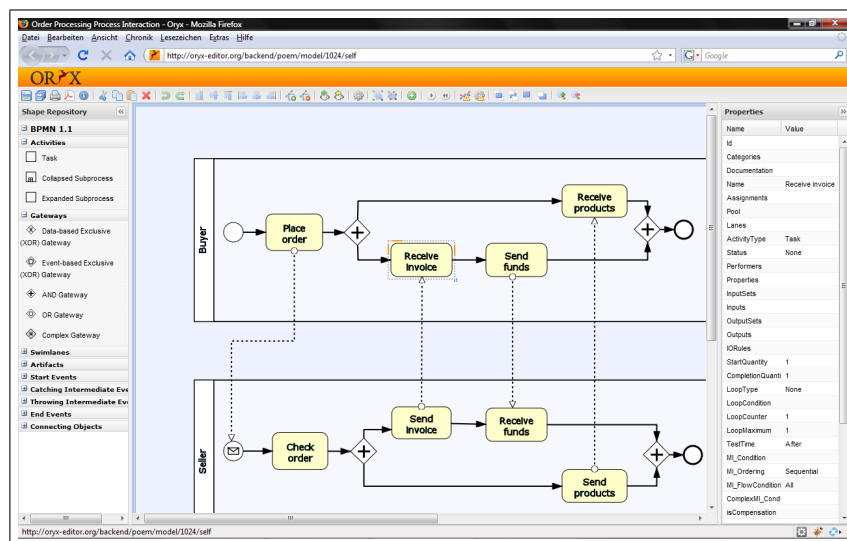
Tooling plays an increasing role in academic research. This is mainly due to two reasons. Firstly, theoretical concepts can benefit from exploration using prototypical implementation. Secondly, the practical applicability of the research work can be demonstrated, which is important to raise awareness of results of academic research to practitioners.

---

<sup>1</sup>Parts of this section are published in (Effinger and Decker 2010).

<sup>2</sup>see company website: <http://www.signavio.com/de>, 2012-09-30.

In academic research groups, researchers tend to implement small-scale prototypes that can do exactly what the particular researcher is interested in. Typically each project is started from scratch. If results from collaborators are reused, then re-use is done in a non-structured way, by copying and pasting program code. As a result, the wheel is reinvented many times, and valuable resources are wasted. Motivated by this observation, the business process technology research group at the Hasso-Plattner-Institute has decided to develop an open and extensible framework for graphical modeling<sup>3</sup> called *Oryx*.



**Figure 8.2.:** A BPMN process diagram modeled by a human process designer. The Oryx user interface is completely browser-based and allows user-friendly drag-and-drop usability.

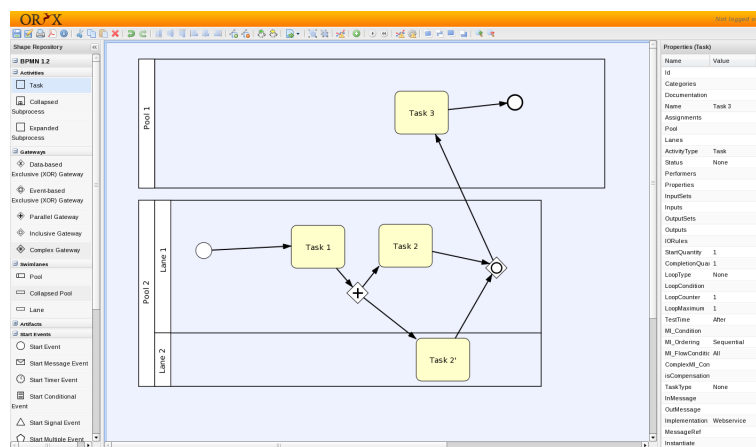
Oryx is an extensible framework for graphical modeling in the web browser. Using JavaScript and Scalable Vector Graphics (SVG), Oryx uses modern web technologies that realize a similar user experience like a classical modeling tool that runs on the desktop. The application is loaded into the browser whenever a graphical model is opened for editing.

In Oryx, each artifact is identified by a URL, so that models can be shared by passing references, rather than by exchanging model documents as email attachments. Oryx follows the Representational State Transfer (REST) architectural style, using the HTTP verbs GET, PUT, POST and DELETE for reading and updating models. This enables a highly scalable architecture, allowing for caching mechanisms at the protocol level.

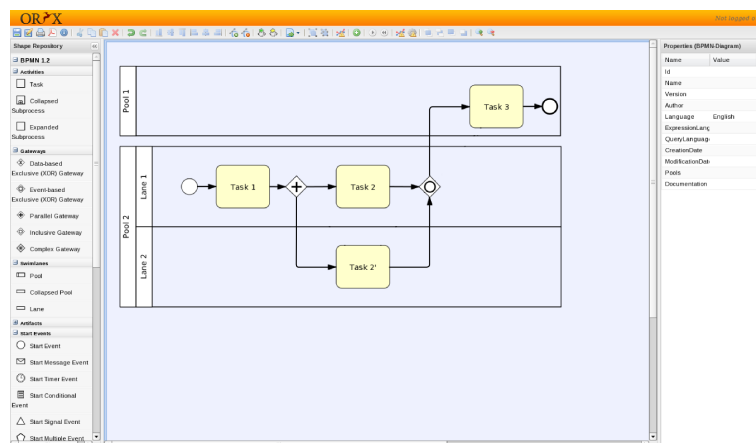
<sup>3</sup>see <http://oryx-project.org>, 2012-09-30.

The Oryx source code is available under an Open Source license and has become a widely used technology platform, especially in the Business Process Management (BPM) community. Here, process modeling using languages such as the Business Process Modeling Notation (BPMN) is a central activity.

The plugin infrastructure of Oryx allows to easily add (1) new modeling notations in a declarative way, including syntax rules such as containment and connection constraints, (2) new client- and server-side functionality, e.g. language transformations or advanced model verification techniques and (3) UI mashups on top of graphical models. The latter can be compared to the Google Maps API: graphical models serve as read-only canvas that can be enriched with UI elements that provide additional functionality on top of model elements.



(a)



(b)

**Figure 8.3.:** (a) A BPMN example process diagram. (b) The resulting process diagram after our approach is applied. The drawing conventions of BPMN are fulfilled.

### 8.2.2. The automatic layout algorithm and integration into Oryx

Our layout approach for support of automatic layout in Oryx is developed by extending previous works on layout techniques adoptable for BPMN (Effinger, Kaufmann, and Siebenhaller 2009; Effinger, Siebenhaller, and Kaufmann 2009a) which were rehased in Section 2.4. The layout is computed on BPMN diagrams that are based on a graphs according to the definition of a BPMN-Graph in Section 2.

An automatic layout approach for BPMN-Graphs has to support the drawing conventions that represent specific layout requirements of the BPMN notation. In the case for the online modeling platform, we use the following specific layout requirements:

- Elements have different sizes.
- We have to consider partitions, e.g. (collapsed/expanded) pools and swimlanes.
- Handle labels of pools, swimlanes, elements and edges.

Since BPMN-graphs are usually drawn using orthogonal routes for edges in Oryx, we use our orthogonal layout approach for computing a layout of a given BPMN-graph.

Our layout approach employs the implementation described in (Effinger, Kaufmann, and Siebenhaller 2009; Effinger, Siebenhaller, and Kaufmann 2009a; Siebenhaller and Kaufmann 2006a) that incorporates different constraints needed for the automatic layout of activity diagrams which are related to business process diagrams. The supported constraints include partitions (a generalization of swimlanes), clusters (subprocesses/groups) as well as a common workflow direction of edges which is especially important for such diagrams. Remember that the techniques in use are based on Sugiyama's algorithm (Sugiyama, Tagawa, and Toda 1981) and the *Topology-Shape-Metrics* (TSM) approach (Tamassia 1987). All above mentioned layout requirements and drawing conventions required by BPMN models can be satisfied.

For the integration of the layout implementations into Oryx, a wrapper was implemented in JAVA that offers interfaces for the connection to the JavaScript-based BPMN-editor Oryx and support of the Oryx-internal diagram model.

An integration of the layout feature into the productive branch of Oryx is yet to be performed. Then, users of Oryx can apply the layout algorithm by pushing a single button in the web-interface of Oryx.

### 8.3. Flight Navigator for Business Process Models

There are countless tools for modeling business processes. They serve the purpose of creating models from new or existing business processes. However, in many cases, a model might already exist but has to be understood and/or analyzed.



The understanding of a business process model depends on the preferred method of presentation. In this project<sup>4</sup>, we present an alternative method called '*Flight Navigator*' for presenting and analyzing models.

We exploit the freedom of an additional dimension for displaying models (*3D*) and offer keyboard/mouse interaction methods to the user for simple navigation in *3D*.

For sequential analysis/presentation of models, we support 'flights' through the process model using smooth animations. We aim at supporting the user to keep his/her mental model while browsing in *3D*. Then, the amount of information perceived by the user is higher than compared to *2D*-diagrams (Ware and Franck 1994). For preservation of a user's mental model, we keep the number of animation motions per flight step steady and set the orientation in the *3D*-environment as fixed, e.g. the up-direction of the viewing angle in *3D* is set to be constant.

In the following, we specify the method that Flight Navigator uses for projection of an input model to its *3D*-environment and describe the navigation and flight features of our tool.

### 8.3.1. Presentation of Flight Navigator

In this section, we present details of our Flight Navigator, a software with features providing interactive browsing in a *3D*-environment. The goal is to support presentation, inspection and analysis of business process models, also with the help of Business Process Flights (BPFs).

For depicting the business process models, we use BPMN. More specifically, we use the part of *collaboration diagrams* of BPMN which represent interactions between roles and responsibilities and their corresponding tasks. In the following, we describe used techniques for projection of process models into *2.5D*, supported navigation features and flight planning of Flight Navigator.

#### Perspectives and Projection onto *2.5D*

For the creation of our BPMN-models in *3D*, we use two techniques: organizational perspective and *2.5D*-projection. In Section 5.3.1, the following set of perspectives on a general business process is presented:

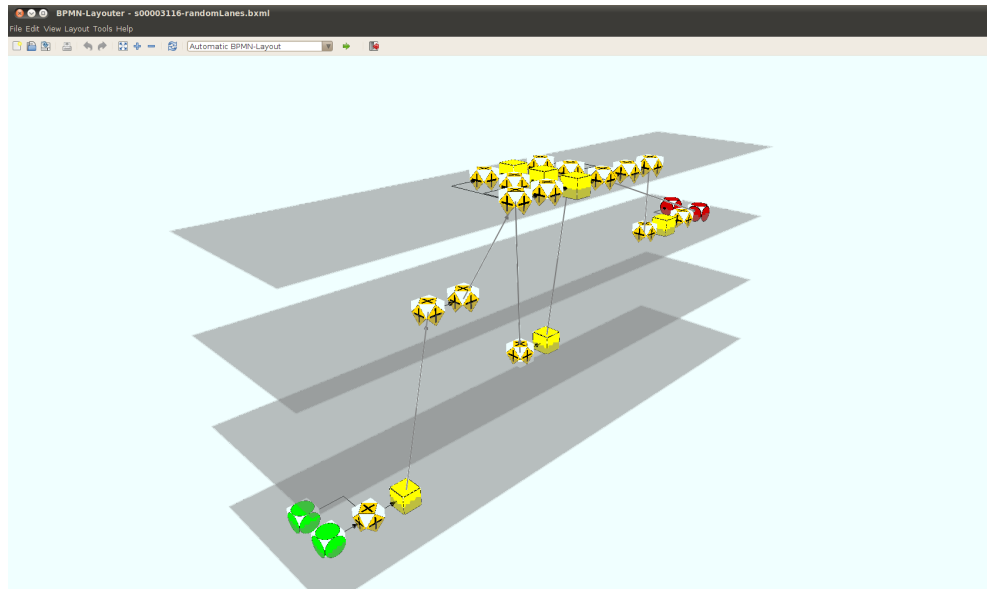
- Functional perspective
- Data (flow) perspective
- Operational perspective
- Organizational perspective
- Behavioural perspective

---

<sup>4</sup>Parts of this project were published in (Effinger 2012).

For our purpose, we employ the organizational perspective. We derive the roles and responsibilities from the structure of swimlanes/pools in the original BPMN model. For instance, swimlanes and pools represent departments or single executives in companies which are assigned specific tasks in a process. If not given by the input model, we compute element positions in  $2.5D$  using layers by applying techniques of automatic layout algorithms, see Section 6.4.

After having derived the view from the organizational perspective of a model, we project the view into  $3D$ , using the concept of two-and-a-half dimensions ( $2.5D$ ), see Section 5.3. Remember that, in  $2.5D$ , elements of the model are assigned to layers. Layers are planes in  $3D$ , with a individual but fixed value in  $z$ -axis direction ( $z$  defines the depth values of layers). The layers can be considered as a stack of planes with equal size. The final view of a process model after projection is depicted in Figure 8.4.



**Figure 8.4.:** Overview of a complete process model in the  $3D$ -navigator. The layout of the model considers the flow orientation (left to right) of the process.

### Navigation support

For navigating in Flight Navigator, we aimed to support high user interactivity and smooth transitions when moving in the  $2.5D$ -model. Flight Navigator offers mouse-actuated navigation on the viewing plane and rotation of that plane, as well as changing the viewing height to accommodate viewing of different layers. The navigation features of our  $3D$ -Navigator, presented in Section 5.3, are also available in Flight Navigator.

### Navigating Flights

The idea of flights in process models is the following: (a) start at an element that represents an entry point of the process; (b) then, follow the process by considering the predetermined directed sequence flow; (c) analyse/present the sequence flow, e.g., check if the order of tasks in this flight was correct and as expected. A jump (a step in the flight) from an element  $a$  to an element  $b$  is allowed only if  $b$  is a successor of  $a$  in the process' sequence flow, or if  $b$  is a predecessor of  $a$ . A jump is performed by a smooth animated transition of the view perspective in Flight Navigator.

Since an element  $a$  may have several predecessors and/or successors, we use a heads-up-display (HUD) to preview miniature images of the possible destination elements. The destination elements are updated after every jump since the destination candidates (the sets of predecessors and successors) depend on the current element. In Figure 8.5, the HUD is depicted for a node with two predecessors and one successor. The placing of the HUD miniature preview images is derived from the keyboard numpad shortcuts for the destination node, e.g. Key '8' represents the top center in the HUD, Key '1' for the left bottom position of the HUD, see Figure 8.5 for an example. The keyboard shortcuts are chosen for good support of a intuitive user interface: going 'forward' is assigned to keys '7' to '9' on the numerical pad for successors; going 'backwards' is assigned to keys '1' to '3' for predecessors while '2' (backwards) is also used to keep a history of the last visited nodes. Keys '4' to '6' are dynamically assigned if the set of neighbours of either direction (forward, backward) grows larger than 3. However, this is a rare case. The maximum size of predecessors/successors in our test set, taken from (Fahland et al. 2009), was low ( $\leq 6$ ).

In activated flight mode, the user can simply select a random node from the model to start a flight. The HUD is then updated immediately. Also, we implemented to switch to start/end nodes using shortcut keys 'HOME' / 'END'.

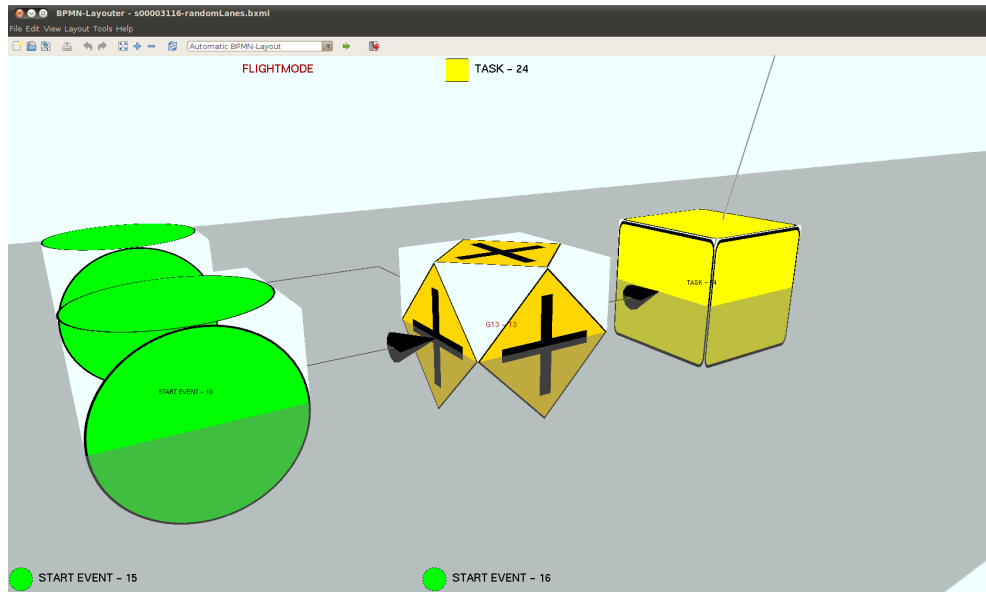
### Implementation

The Flight Navigator is implemented using standard technology. We use JAVA<sup>TM</sup> 1.5 and JOGL<sup>5</sup>, an open-source interface for binding OpenGL® into a JAVA program. For graph analysis and basic data structures, we employ the graph library yFiles developed by yWorks<sup>6</sup>. Flight Navigator is also integrated into our academic BPMN modeling tool *BPMN-Layouter*<sup>7</sup>.

<sup>5</sup><http://download.java.net/media/jogl/www/>, 2012-09-30.

<sup>6</sup><http://www.yworks.de>, 2012-09-30.

<sup>7</sup> see homepage of *BPMN-Layouter* for a video demonstrating interactivity and animations in Flight-Navigator:  
<http://algo.inf.uni-tuebingen.de/?site=forschung/graphenzeichnen/bpmn-layouter>, 2012-09-30.



**Figure 8.5.:** Display of the HUD (head-up-display) in the Flight Navigator. From the current node (Gateway 'G13'), a single successor ('TASK - 24') is reachable by pressing keyboard shortcut '8', see top center HUD position; the two predecessors ('START EVENT - 15', 'START EVENT - 16') are reachable by pressing '1', or '2' respectively, see bottom HUD positions.

### 8.3.2. Summary

We presented our project with the software tool 'Flight Navigator', a 3D-navigator for analyzing and presenting business process models. It supports strong interactivity with the user using keyboard/mouse input and real-time 3D-presentation. For the presentation, we use the concept of 2.5D for projection of business process models into 3D. Future development might comprehend adding more features: The Flight Navigator software is ready to be extended, e.g. for editing business process models in 3D or for activating an 'autopilot' for automatic process presentations.

## 8.4. GraphArchive

The work on *GraphArchive*, an online database for graphs, was initiated during Dagstuhl seminary 11191<sup>8</sup> in May 2011. There, the need for a central store for graphs become obvious when performing a survey among the seminary participants. The survey asked for needs and habits when handling graphs which, i.e., were used in publications for

<sup>8</sup>for more details on Dagstuhl seminary 11191, see <http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=11191,2012-09-30>.

testing or evaluation and should be made open to the public for further investigations. Our survey results are made accessible by the Dagstuhl organizers at: <http://www.dagstuhl.de/mat/Files/11/11191/11191.SpoenemannMiro.Slides1.pdf> The fruitful discussions at the seminary and the subsequent development of a web–platform resulted in several publications where our approach GraphArchive was described and used. Among these publications are (Bachmaier et al. 2011a,b; Effinger, Kaufmann, Meinert, et al. 2011). The details of the code base development are presented in (Stegmaier 2011). In (Effinger, Kaufmann, Meinert, et al. 2011), a history of GraphDB is presented. GraphDB is the predecessor of GraphArchive which dates back to 2004 and had several drawbacks that required a full redesign and new implementation. However, the overall goal is already valid for both, GraphArchive and GraphDB: to enable researcher to share graphs as easy as possible for a fast distribution of data sets. On the 19th International Symposium on Graph Drawing in September 2011, GraphArchive was officially launched and it was presented to its target audience.

#### 8.4.1. Features of the new GraphArchive

In the following, we will provide a list of the main features of our new approach and present our system architecture as presented in (Effinger, Kaufmann, Meinert, et al. 2011). Then, we will present selected key features in more detail.

##### Main features of GraphArchive

All features are chosen supporting the guideline that our major goal is to provide an open and easily accessible system. In the following, we present the main features of the new system:

- **web–based user interface:** All user interaction is done online via a browser. A web portal offers all functionality that is needed to handle a graph from uploading data, inspection of existing graphs and search for others and, finally, downloading a found graph.
- **automated registration (email opt–in):** Registration is performed online using a registration form, which is handled automatically. The system sends immediately a registration link via email after submitting the form.
- **limited rights management:** There are no groups of users that define rights for small circles of users. Licenses for graphs limiting usage are not encouraged in our open approach, thus, if necessary, a license can be attached to individual graphs only.

- **open access to all graphs after registration:** After confirming registration by fulfilling the email opt-in process, a user has full access to all graphs and can initiate queries without restrictions.
- **categorization of graphs (e.g., fields of application):** For search, graphs can be assigned to the field(s) of application that they derive from. This enables researchers from different fields to use GraphArchive as a common platform.
- **automatic graph analysis after upload (for graphs with <100.000 nodes):** After upload, graphs are analyzed in order to provide consistent data. The consistency is very important for search queries on graph properties. Also, automatic analysis might reveal more properties than manual assignment.
- **search for graphs using multiple criteria:** Search queries can be executed on multiple parameters, among them are graph properties, categories, author, name and upload date. Also, search parameters can be combined to further narrow down the result set.
- **support of user-defined tags attachable to graphs:** Users can define individual tags to identify special attributes of graph(s). All user-defined tags are made fully searchable.
- **support of grouping of graphs:** Graphs can be grouped to mark their relation, e.g., graphs that stem from a specific test data set. Graphs that are uploaded as a single zip file are also grouped using one distinct tag, e.g. the zip file's name.
- **support of graph layouts to create visualizations (images) of graphs:** An image of a graph is valuable if a user quickly wants to inspect visually a graph's properties. Layouts are computed automatically in the background and also can be changed after upload.
- **support for creating comments and references:** Commenting on graphs might initiate discussions on certain graphs. Also, descriptions can be stored as comments. References can be assigned to a graph in order to highlight publications and/or websites that made use of this graph in any kind.
- **unique links to a graph (URI) for referencing in publications:** A URI allows for a permanent reference in publications. Stating the URI in a publication enables the reader to quickly find the used graph data set.
- **'multiview' for comparing multiple graphs on a single page:** For quickly comparing multiple graphs at a time, we support the presentation of various graphs at a time. Properties are displayed for all graphs. Boolean properties, e.g., directed/undirected, are presented visually on a scale (property can be fulfilled by (a) no graph, (b) a subset of the displayed graphs or (c) all graphs).
- **support of various graph file formats:** Since it is impossible to decide on a specific file format when supporting many fields of applications, we aim at

providing support for as many formats as possible. Our system allows to add further formats in the future.

- **support of graph file format conversion for downloads:** For downloading graphs, a user can choose the format that fits best to his/her work environment. We provide cross conversion (the users can select any supported format and the system starts the conversion automatically).
- **support of zipped files for import/export of multiple graphs:** When handling a test data set of graphs, we allow to upload/download several graphs at a time using zip compression. In an upload process, each file in the compressed file can optionally be processed individually (for properties analysis and layout computation). When downloading several files, the system automatically creates a compressed file containing all selected graphs.
- **graph authorship management featuring *my graphs* for graph authors:** An author of graphs can easily manage his/her graphs using the view 'my graphs' where inspections and actions, e.g., deletions of multiple graphs, are quickly accessible.
- **guest access for non-registered users:** If a user wants to check a specific graph, he/she can access a detailed view on the graph using the URI. All properties and attributes of the graph are made visible entering via the guest account. However, actions, e.g., commenting, changing properties or download, are disabled in this view.

## Architecture

Our system architecture is built similar to a common web-browser application including a couple of necessary extensions for handling of graphs. The application is written in PHP5<sup>9</sup> using Apache2<sup>10</sup> for online presentation. For graph analysis and layout computation, we make use of the java graph library yFiles<sup>11</sup>, which is handled in the background via PHP/JAVA Bridge<sup>12</sup>. Data storage is provided by a PostgreSQL database<sup>13</sup>. A schema of the system architecture is depicted in Figure 8.6.

<sup>9</sup>see project homepage: <http://www.php.net>, last accessed 2011-07-12

<sup>10</sup>see project homepage: <http://www.apache.org>, last accessed 2011-07-12

<sup>11</sup>developed and maintained by yWorks GmbH: <http://www.yworks.com>, last accessed 2011-07-29

<sup>12</sup>Online source to the SourceForge project available at:

<http://php-java-bridge.sourceforge.net/pjb/index.php>, last accessed 2011-07-12

<sup>13</sup>see project homepage: <http://www.postgresql.org/>, last accessed 2011-07-12

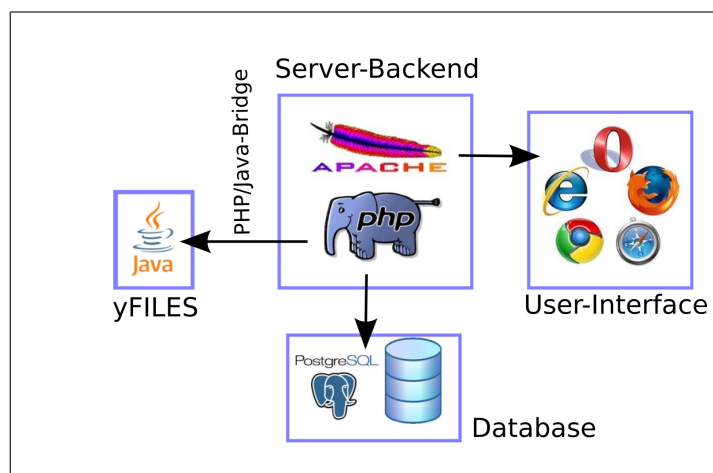


Figure 8.6.: Architecture of GraphArchive.

#### Presentation of selected key features:

**Rights management** In the former approach GraphDB, many graphs were not public by default. Thus, rights handling was a major issue. A hierarchy of rights was integrated, involving group rights and user rights for a graph. Download of a graph was allowed only if a user was granted the appropriate rights. If a user was not specifically granted the right for a graph, and the user was also not assigned to a group that had access to the graph, access and download of the graph was denied.

Our system pushes rights for access and downloads towards an open-access approach. After registration, all graphs are accessible and may be inspected, e.g., to analyze graph properties. When uploading a graph to GraphArchive, the author of a graph needs to confirm that he/she holds the rights to publish the graph. Also, the author agrees that the graph is shared in GraphArchive.

Since some graphs come with usage limitations and/or demands, it is possible to integrate a license to a graph when uploading as graph author. In this case, a later graph download demands confirmation of the license before file transfer is started. Due to the fact that GraphArchive is intended to be an open-access platform we restrict licenses to be assignable on a 'per-graph' base only.

**Tagging** For assigning properties to graphs, we use the principle of tags. Tags consist of a (key, value)-pair, whose value is of type boolean, integer or double. Since the principle of tags is general, we can use it for several purposes:

Tags allow ...

- ... graph categorization, i.e., assignment to field(s) of application (e.g., metabolic networks, electrical circuit, class diagrams and many others).



- ... assigning of graph properties, e.g., acyclic, directed or degree, etc.
- ... graph grouping, e.g., graphs of one group have the same boolean tag set to true.
- ... user-defined properties, e.g., user can create new tags and assign them to graphs.

**Graph analysis and visualization** Graph analysis can be a tedious task when done manually, it even may prevent users to upload graphs. However, graph properties are of essential importance when it comes to search for specific properties. To provide a valuable query mechanism, an archive depends on sufficient assigned properties. To free the user from this task, we perform an automatic graph analysis on the graphs after upload. Graphs are analyzed for a pre-defined default set of properties. The set comprises the following:

node count	edge count	biconnected	bipartite
connected	cyclic	forest	multiple edge free
planar	rooted tree	self loop free	simple
strongly connected	tree	component count	minimum degree
maximum degree	average degree	median degree	

In parallel to the graph analysis, layouts of the graph are created and stored as images for later presentation on the graph's detail page. The images are created using a standard layout algorithm provided in yFiles, a Java library to work with graphs. The default layout is computed by a spring layout algorithm (Di Battista, Eades, et al. 1999). The library is integrated into the system with the help of the PHP/Java Bridge, which allows to connect JAVA classes to PHP scripts. The layout algorithm can be changed later on the graph detail page where new layouts can be created (e.g. orthogonal/hierarchical/spring/circular layout).

Since the computation for some properties and layouts is very time consuming, we perform a complete analysis only for graphs with < 100.000 nodes. The analysis is done in the background to not disturb the user while browsing in GraphArchive; this also holds for the computation of layouts.

**Referencing graphs** Often, researchers use sets of graphs to perform experiments. In order to render such experiments repeatable for other researchers, it is preferable that these data sets are referenced in the corresponding publication. To allow this, we introduced the possibility to add references to a graph in our system. A reference consists of a description and an optional link to the relating publication. For each graph, multiple references are possible. The references are also searchable to be found easily via the main page.

Additionally, we create a unique description for each graph (URI). Given the URI of a graph, it can be reached online by adding the URL of our system, e.g.,

```
http://algo.inf.uni-tuebingen.de/forschung/graphdb/graphs/showgraph.php?graph=bdc3639a
```

where *bdc3639a* represents a graph's URI. URIs are considered static such that they are not supposed to undergo changes even in case that the underlying system is modified heavily. Also, given the URI, one can view the corresponding graph as a non-registered user. Thus, readers of a publication given a URI of our GraphArchive can have a look at the graph. This is provided by our guest access. The guest access is entered by browsing to a URL as described above. Major differences between a guest and registered users are: guests can only access a single graph, they have no access to the main page; guests are not allowed to perform actions, e.g., search, upload or download.

**Search for graphs** The query mechanism of our system allows to search for graphs by selecting and specifying query parameters. The parameters can be combined. A picture of the search form is given in Figure 8.7. Main search criteria are:

- **graph properties:** when searching for a graph property, e.g., number of nodes, a distinct value is supported as well as a given range or upper/lower bound, e.g., graphs with more than 10 nodes but less than 100 nodes.
- **graph categories:** the categories are stored using tags. Thus, graphs with a specific field of application carry the name of their field as an attributed keyword.
- **author/graph name:** search for graphs uploaded by a specific user or named by a specific name, e.g., *Metro map*.
- **upload date:** search for graphs according to an upload date. We provide search for specific dates but also for periods of dates, if the exact date is unknown.
- **additional keywords (tags):** user-defined keywords are treated as tags and are searchable by selecting the appropriate keyword in the search form.
- **references:** graphs can also be found by a lookup according to the references that are connected to them or their specific URI.

**File formats** In the field of Graph Drawing, there are numerous tools with very different file formats. The reasons for the usage of a distinct file format can be multifaceted, e.g., text graph format (.tgf) can be favoured for its simplicity whereas xml-based format *graphml* (.gml) (Brandes, Eiglsperger, et al. 2001) might be preferred due to its extensibility.

**Figure 8.7.:** Query form of the free search: retrieving graphs by giving a range of upload dates is also among the possible search queries.

The reasons why one file format is preferred over others depends on the field of application. One aim of GraphArchive is to become a central graph repository for all domains of interest. Therefore, we do not favor one of the file formats, but we try to achieve support of as many file formats as possible. We are convinced that limitation to a few file formats might prevent people to use the GraphArchive. We also support conversion between our supported formats when the user wants to download a graph. As we continuously improve our approach, we are open for source code contributions to enlarge our set of supported file formats. Currently, the following formats are supported:

Description	Abbreviation
Text graph file format	.tgf
GraphML	.graphml
Compressed GraphML	.graphmlz
Graph Markup Language	.gml
Graph Markup Language (XML)	.xgml
Y Graph Format	.ygf

If a graph is uploaded in an unknown format, it is left unprocessed and stored as a binary file. Then, graph analysis and layout computation as well as conversion for export is not possible for this graph.

For ease of import/export and the handling of graph libraries with numerous graphs, we also support zipped files. When uploading a zipped file, the compressed file is optionally extracted and each contained file is processed individually as a graph file. Downloading several graphs (without format conversion) is facilitated by compressing these graphs using zip compression before download.

**GraphArchive Main Menu**

EBERHARD KARLS UNIVERSITÄT TUBINGEN

Logout  
Account administration  
My graphs  
Administration

Search for graphs:

**Upload graphs**

**All graphs currently in the database:**

Graphs per page: [5](#) [10](#) [20](#) [50](#) Page 1 of 3 | 2 | 3 | Next > | Last >>

<input type="checkbox"/>	ID	Name	Author	Upload date	URI
<input type="checkbox"/>	384	Krugsche Testgraphen	Robert Krug	01.02.2011	<a href="#">cdda111</a>
<input type="checkbox"/>	385	kottler-graphs	Stephan Kottler	01.02.2011	<a href="#">7ac72015</a>
<input type="checkbox"/>	386	Krugsche Testgraphen	Robert Krug	01.02.2011	<a href="#">a3e16318</a>
<input type="checkbox"/>	387	kottler-graphs	Stephan Kottler	01.02.2011	<a href="#">14fca21c</a>
<input type="checkbox"/>	388	graph3	Stephan Kottler	01.02.2011	<a href="#">a741ed24</a>
<input type="checkbox"/>	389	graph4-kottlers	Stephan Kottler	01.02.2011	<a href="#">1e5c2c20</a>
<input type="checkbox"/>	390	graph3	Robert Krug	01.02.2011	<a href="#">cd6dfcd0</a>
<input type="checkbox"/>	391	graph4-krug	Robert Krug	01.02.2011	<a href="#">7a793dd4</a>
<input type="checkbox"/>	405	3D-Nodes	Philip Effinger	08.02.2011	<a href="#">hdc3639a</a>
<input type="checkbox"/>	455	Edge Bundling David Auber	Philip Effinger	11.05.2011	<a href="#">c8235e0d</a>

[download selected](#) | [view selected](#)  
[download all](#) (Warning: This may take a while!)

Page 1 of 3 | 2 | 3 | Next > | Last >>

**Figure 8.8.:** Screenshot of the GraphArchive main page.

## 8.4.2. Presentation of the new system

In this section, we want to give an impression of the design and online appearance of GraphArchive by taking a virtual walk through a typical use case. The reader is encouraged to make a tour on his own by browsing to the current GraphArchive via our institute entry page:






<http://algo.inf.uni-tuebingen.de/?site=forschung/graphdb/grapharchive>

In Figure 8.8, the main page is depicted, graphs are displayed in a table. The table is sortable ascending/descending in any of the columns. For a quick overview on the main page, the user may show detailed information of the displayed graphs, where key facts of the graphs are given, see Figure 8.9.

The detailed single graph page is shown in Figure 8.10 including an image for visualization and a list of the attributes of the graph. On the details page, comments or references can be updated and users may add additional tags. The default set of tags that is analyzed automatically and the insertion for user-defined keywords is presented in Figure 8.11.

**All graphs currently in the database:**

Graphs per page: [5](#) [10](#) [20](#) [50](#)    << First | < Prev | 1 | Page 2 of 5 | 3 | 4 | 5 | Next > | Last >>

<input type="checkbox"/>	ID	Name	Author	Upload date	URI
<input type="checkbox"/>	389	graph4-kottlers	Stephan Kottler	01.02.2011	<a href="#">1e5c2c20</a>
			<ul style="list-style-type: none"> <li>node count: 300</li> <li>edge count: 500</li> <li>not connected</li> </ul>	<ul style="list-style-type: none"> <li>cyclic</li> <li>minimum degree: 0</li> <li>maximum degree: 8</li> </ul>	<ul style="list-style-type: none"> <li>component count: 13</li> <li>multiple edge free</li> <li>not planar</li> </ul>
<input type="checkbox"/>	390	graph3	Robert Krug	01.02.2011	<a href="#">cd6dfcd0</a>
			<ul style="list-style-type: none"> <li>node count: 110</li> <li>edge count: 150</li> <li>not connected</li> </ul>	<ul style="list-style-type: none"> <li>cyclic</li> <li>minimum degree: 0</li> <li>maximum degree: 7</li> </ul>	<ul style="list-style-type: none"> <li>component count: 5</li> <li>multiple edge free</li> <li>not planar</li> </ul>
<input type="checkbox"/>	391	graph4-krug	Robert Krug	01.02.2011	<a href="#">7a703dd4</a>
			<ul style="list-style-type: none"> <li>node count: 300</li> <li>edge count: 500</li> <li>not connected</li> </ul>	<ul style="list-style-type: none"> <li>cyclic</li> <li>minimum degree: 0</li> <li>maximum degree: 8</li> </ul>	<ul style="list-style-type: none"> <li>component count: 13</li> <li>multiple edge free</li> <li>not planar</li> </ul>
<input type="checkbox"/>	405	3D-Nodes	Philip Effinger	08.02.2011	<a href="#">fde3639a</a>
			<ul style="list-style-type: none"> <li>node count: 6</li> <li>edge count: 6</li> <li>connected</li> </ul>	<ul style="list-style-type: none"> <li>cyclic</li> <li>minimum degree: 1</li> <li>maximum degree: 3</li> </ul>	<ul style="list-style-type: none"> <li>component count: 1</li> <li>multiple edge free</li> <li>planar</li> </ul>
<input type="checkbox"/>	455	Edge Bundling David Auber	Philip Effinger	11.05.2011	<a href="#">c8235e0d</a>
			<ul style="list-style-type: none"> <li>node count: 1715</li> <li>edge count: 6529</li> <li>not connected</li> </ul>	<ul style="list-style-type: none"> <li>cyclic</li> <li>minimum degree: 1</li> <li>maximum degree: 231</li> </ul>	<ul style="list-style-type: none"> <li>component count: 28</li> <li>multiple edge free</li> <li>not planar</li> </ul>

[download selected](#) | [view selected](#)    << First | < Prev | 1 | Page 2 of 5 | 3 | 4 | 5 | Next > | Last >>

[download all](#) (Warning: This may take a while!)

**Figure 8.9.:** Expanded view of the main page with quick facts of graphs.

### 8.4.3. Summary

In this section, we presented our project *GraphArchive*. It enables the community to exchange graphs in a central online storage. Also, it provides a data store for archiving graphs, e.g., graphs that are used in test suites.

With our new approach GraphArchive, we tackle the identified weak spots of the predecessor GraphDB. The new application is developed as an online tool supporting and exploiting modern web technologies. The portal is fully accessible via a common browser. The goal was to provide an easy-to-use and powerful yet simple graph data platform.

GraphArchive enables interested researchers to find, share and store graphs of various fields of applications, e.g., social networks, road networks, class diagrams or metabolic networks. Additionally, it provides a persistency mechanism, which allows for storing data sets and permanently referencing them by a URI. This allows to reference data sets in future publications, which makes experiments more transparent, repeatable and reliable.

Also, the automated analysis of graphs increases usability and data consistency. Integrated layout computations provide visualizations for quickly grasping mental maps of graphs.

**Details of graph with ID 405**

[new layout](#)

[Download graph](#)  
Choose format:

**Name**  
3D-Nodes

**Author**  
Philip Effinger

**Upload date**  
08. February 2011

**Appears in**  

- [Journal for Graphs](#)
- (add new reference...)

**URI**  
<https://kandinsky.informatik.uni-tuebingen.de/forschung/graphdb/graphs/showgraph.php?graph=bdc3639a>

**Comments**  

- kann man die Schatten der Knoten sehen? posted on 08. February 2011 by you
- ja, super! posted on 08. February 2011 by you
- das möchte ich lesen, nochmal! posted on 08. February 2011 by you
- (add new comment...)

**Keywords**  

- node count:
- edge count:
- biconnected:
- bipartite:
- connected:
- cyclic:
- forest:

**Figure 8.10.:** Detailed graph page; downloads can be initialized on this page and layout calculation can be selected. On the right, attributes are listed; at the bottom, user-defined tags can be added, as shown in Figure 8.11.

**Details of graph with ID 405**

[new layout](#)

[Download graph](#)

**Keywords**  

- node count:
- edge count:
- biconnected:
- bipartite:
- connected:
- cyclic:
- forest:
- multiple edge free:
- planar:
- rooted tree:
- self loop free:
- simple:
- strongly connected:
- tree:
- component count:
- minimum degree:
- maximum degree:
- average degree:
- median degree:

**Add keyword manually:**  
Name of keyword:   
Type:   
Value:

**Figure 8.11.:** Complete display of default graph tags, the user-defined tags can be set at the bottom and will be added immediately to the tags.

---

As a matter, development of the tool is not completed. In the future, we will keep improving the running system and adding new features to GraphArchive. We will use the homepage at <http://www.graph-archive.org> as a platform to post news and development progress of our system. We hope that our system succeeds in providing a helpful service and is hopefully being promoted and supported by the community to establish a central place to go for sharing graphs. The rise and fall of the system depends on user acceptance and its regular usage. What renders us to be optimistic is the number of  $\approx 170$  registered users (last checked on August 1st, 2012) less than one year after the official launch.





## Conclusion

In this work, we addressed the challenge of computing visualizations for business process models. We presented various novel algorithmic approaches to compute visualizations in *2D* and *2.5D/3D*. The approaches are based on the projection of business process models on graphs. Using graphs as abstracts for process models, we were able to employ graph drawing techniques, combined with results of research on layout aesthetics in order to provide visualizations of business process models.

In the following, we will briefly summarize our results and contributions. Also, we point out relevant publications of the author which were published during the research for this work. We will close with an outlook on possible future topics.

### Results

#### Groundwork: study of layout aesthetics

For the modeling of business process models, we use the standard notation language *BPMN*. For the computation of visualizations for models in *BPMN*, we conducted a study on user preferences of layout aesthetics in *BPMN* models. This was necessary preparatory work to create a starting point for our algorithmic approaches which are to fulfill these layout aesthetics. Due to the study, we could define a set of layout aesthetics, the *Standard Layout Aesthetics*, to rely on for the remainder of the work. The algorithms, which we developed for *BPMN*, aimed at considering the *Standard Layout Aesthetics* and were able to fulfill these constraints.

Also, in the study, we evaluated the usability and user friendliness of our integrated software suite *BPMN-Layouter* which allows to model and visualize *BPMN* in models in *2D* and *2.5D*.

Publications: (Jogsch et al. 2010), (Seiz et al. 2010), (Effinger, Jogsch, and Seiz 2010), (Effinger, Seiz, and Jogsch 2011).

### Algorithms for visualizations of business process models in 2D

For visualizations of business process models in the two-dimensional plane, we presented the following contributions:

- Sketch-Driven-Layout (SDL) for BPMN: we presented an approach for dynamic visualizations that considers a given embedding of a graph when computing a layout. The approach SDL is realized as an extension to the TSM approach which computes visualizations for graphs in Kandinsky model. The extended approach of SDL is able to consider the partition of a business process models, which is prerequisite to support BPMN models, because swimlanes and pools are basic structures in BPMN models.  
Publications: (Effinger, Kaufmann, and Siebenhaller 2009), (Effinger, Siebenhaller, and Kaufmann 2009b), (Effinger, Siebenhaller, and Kaufmann 2009a).
- Layout Patterns: three novel patterns for visualizations of business process were presented. They can be applied to SDL and are targeted at semantic considerations of business process models in visualizations. The patterns and their goals are:
  - Geometry pattern (GeoP): reduce visual cluttering of process model diagrams.
  - Gateway Pattern (GaP): highlight the logical structure of process models.
  - Start-End-Pattern (SEP): enforce aesthetics *FLOW* on start and end event. We provided two variants, dynamic SEP or locked SEP.

We gave details on the algorithmic extensions that are necessary to integrate the patterns into SDL. We also analyzed the effects of the patterns on layout aesthetics. Therefore, we computed visualizations using SDL enriched by patterns and compared the results against a static approach. The effects of the patterns on performance and layout aesthetics are manifold and vary for different single patterns or combination of patterns and different layout aesthetics. However, this is a new approach for layout algorithms which promises more satisfiable visualizations by targeting at specific semantic properties of business process models. Publication: (Effinger 2011).

- Visualization of BPEL models: we presented a complete algorithm for the computation of visualizations for BPEL processes. A transformation from XML-based BPEL files to graphs was introduced, and shapes and structures were integrated in the visualizations to highlight the structure and semantics of BPEL process elements. The layout approach stresses the hierarchical and sequential structure of BPEL processes, by defining paths in the layout that are to be visualized in a straight fashion. The algorithms are integrated into the Sugiyama

framework. The necessary modifications of the steps in the Sugiyama framework towards the visualization of BPEL processes were elaborated in detail. Also, we provided comparisons to other (commercial) visualizations of BPEL models which clearly indicated the supremacy of our visualization approach. The approach was integrated into the BPEL modeling platform HOBBS. Publications: (Albrecht, Effinger, Held, Kaufmann, and Kottler 2009), (Albrecht, Effinger, Held, and Kaufmann 2010).

### **Algorithms for visualizations of business process models in 2.5D**

The second part comprehends contributions to visualizations in 3D. We applied the concept of two-and-a-half-dimensions (2.5D) to business process models by creating perspectives for BPMN models in 3D. The perspectives are: (a) organizational perspective, (b) functional perspective and (c) control flow/data flow perspective.

We presented our software tool 3D-Navigator which extended *BPMN-Layouter* to support and display BPMN models in 3D, integrated in an interactive user interface with real-time 3D-rendering.

For the computation of 2.5D-visualizations, we presented three different algorithmic approaches: (a) SLS: an approach which is SDL-based and operates on single layers in 2.5D, (b) ILP: an ILP formulation for 2.5D-visualization of BPMN models, and (c) PSL: a TSM-based approach, with extended Sugiyama framework and usage of topological orderings, for the projection of BPMN models into 3D-space.

Summarizing the thorough analysis and benchmarks of the proposed approaches for 2.5D-visualizations of business process models, we observed that our approaches SLS, ILP and PSL are feasible and produce satisfying layouts.

We can draw the following conclusions from the benchmarks on SLS, ILP, PSL and a reference implementation of Fruchterman-Reingold (FR) (Fruchterman and Reingold 1991):

1. PSL computes the resulting layouts with the best performance.
2. PSL consumes the lowest amount of area for the resulting layouts.
3. PSL has the lowest average edge lengths.
4. PSL achieves lowest average edge lengths at the expense of higher maximum edge lengths.
5. SLS is unable to cover the issue of space usage by inserted dummy nodes.
6. ILP produces optimal results. However, ILP solves only 43% of the instances in the test data set within a given time frame of 120 seconds.

### Additional projects involving visualization

Furthermore, we could also present additional visualization projects that produced further results during the research on this work:

- *3D-Navigator* provides a highly interactive user interface with navigation support in *3D* and real-time presentation of *2.5D*-visualizations. A use case of *3D-Navigator* is the newly introduced *Flight-Navigator* for business process models that enables a user to interactively browse through a process model by following the sequence flow.  
Publication: (Effinger 2012).
- *ComposerGraphViewer (CGV)* : an interactive tool for the visualization of relations between composers with birth dates from the year 1000 A.D. to the present. The tool provides extensive features for filtering and search and visualizes the data in an animated fashion.
- *Layout in Web2.0-modeling with BPMN*: an extension for the ORYX modeling platform for BPMN models was presented which incorporated layout features to ORYX by integrating the visualization approach of Section 2.4 into the backend of the online Web2.0-based modeling platform.  
Publication: (Effinger and Decker 2010).
- Community project *GraphArchive*: design and implementation of an online portal for a graph database which was provided to the graph drawing community. The feature list of the online database is evolving further until today and *GraphArchive* has about  $\approx 170$  registered users (date: August 2012).  
Publications: (Effinger, Kaufmann, Meinert, et al. 2011), (Bachmaier et al. 2011a).

We think that the results of our visualization approaches for *2D* and *2.5D* are able to ignite the influence of visualization in business process models. Also, we hope that our findings contribute to the rising importance of visualizing processes during the modeling and analysis phase of process management.

## Future topics

Before closing, we would like to point to possible future topics on visualization of business process models.

### **Improvements in pattern-based layout:**

It is an interesting task for the future to conduct a user study in order to receive feedback on the benefits/drawbacks of the patterns from the user's perspective. Also, statements from users might lead to more patterns in the future that incorporate BPMN semantics in layouts for BPMN diagrams.

Furthermore, user studies might lead to findings for higher interactivity when using the patterns, e.g. automatic recommendations which patterns might be suited well for a specific BPMN model.

### **User experience in 2D vs. 2.5D:**

Although we provided the algorithmic foundations for visualizations of business process models in 2.5D in this work, we did not examine if a human process modeler might prefer 2.5D- over 2D-visualizations of a particular BPMN model. User studies in this field are necessary to analyze the possible benefits and typical use-cases of 2.5D-visualizations.

### **Support of BPMN 2.0:**

In this work, we focused on collaboration diagrams of BPMN. With BPMN2.0, there are now two new diagrams types, conversations and choreographies. It would be worth to include these diagrams in a tool which supports visualization for all diagram types of BPMN. From an algorithmic perspective, the two diagram types require less complex solutions because their necessary set of layout aesthetics is smaller than our Standard Layout Aesthetics for collaboration diagrams, i.e., conversations and choreographies do not use partitioning.

We had several requests from developers of commercial modeling suites for the techniques and algorithms used in our visualization. However, during this work, we were not able to spend more time on placing our approaches in a widely used modeling software, apart from Oryx. Full support of BPMN2.0, including conversations and choreographies, might ease this future task.



# List of Figures

1.1. Rules of rock, paper, scissors as a graph. . . . .	2
2.1. Example for a GT ordering . . . . .	16
2.2. Example for an L-triangle and a T-triangle . . . . .	19
2.3. Network flow model of a node $v$ for the Kandinsky model. . . . .	21
2.4. Rectangular decomposition . . . . .	22
2.5. Connecting objects in BPMN. . . . .	27
2.6. Artifacts in BPMN. . . . .	28
2.7. Hierarchic structure of a pool with three lanes in BPMN. . . . .	28
2.8. Example of a BPMN process model . . . . .	30
2.9. Ratings of the categories for total group . . . . .	38
2.10. Comparison for total group. . . . .	40
2.11. Comparison for subset groups of 'Gender'. . . . .	40
2.12. Comparison for subset groups of 'Experience'. . . . .	41
2.13. Comparison for subset groups of education. . . . .	42
2.14. Layout example of tool with highest score. . . . .	43
2.15. Structure of partition cells . . . . .	46
2.16. Example drawing of 2D-approach for BPMN models. . . . .	47
3.1. Modification of a vertex-node in the Kandinsky network for SDL. . . . .	53
3.2. Modifications for bends in Kandinsky network. . . . .	54
3.3. Structural edges. . . . .	55
3.4. Steps of layout example with SDL. . . . .	57
3.5. Determination a route for a cut in a BPMN-graph. . . . .	58
3.6. Visualization of cluttering (densities) using circles. . . . .	62
3.7. Visualization of cluttering with blocks. . . . .	62
3.8. Schematic view of the algorithm for the Geometry Pattern. . . . .	63

3.9. Insertion of skeleton edges to SDL when applying GaP. . . . .	65
3.10. Dynamic Start–End–Pattern. . . . .	67
3.11. Locked Start–End–Pattern. . . . .	67
3.12. Example results for layout patterns . . . . .	70
3.13. Performance of the layout patterns. . . . .	71
3.14. Effects of layout patterns on crossings and bends. . . . .	71
3.15. Effects of layout patterns on area size. . . . .	72
3.16. Effects of layout patterns on edge lengths. . . . .	72
3.17. Example layout by a simple approach for BPEL layout. . . . .	76
3.18. Workflow layout by the ActiveBPEL® Designer. . . . .	78
3.19. Workflow layout by the Eclipse BPEL Designer. . . . .	79
3.20. Workflow example constructed with our layout algorithm. . . . .	79
3.21. Split of pathways. . . . .	80
3.22. Modifications of the computation of the horizontal coordinates. . . . .	82
3.23. Input structure of a BPEL workflow. . . . .	84
3.24. Result of the extension for pathway construction. . . . .	85
3.25. Result for the layout of $G$ . . . . .	86
3.26. BPEL–specific shapes. . . . .	87
3.27. Result of the final layout for running BPEL model example. . . . .	88
3.28. Table containing the BPEL model meta–data. . . . .	89
3.29. Layout example of a larger BPEL process. . . . .	91
5.1. Example for a visualization in $2.5D$ . . . . .	98
5.2. Example for a $2.5D$ –layout of a stratified graph. . . . .	100
5.3. Rendering of a business process model in the $3D$ –framework. . . . .	103
5.4. Rendering of single BPMN elements in $3D$ . . . . .	103
5.5. Example process and $2.5D$ –perspectives. . . . .	107
5.6. Screenshot of the $2.5D$ –visualization display in $2.5D$ –Navigator. . . . .	108
5.7. Screenshot of the GEOMI–framework. . . . .	109
5.8. Application of the perspectives to an example process. . . . .	111
6.1. Layer stack in $3D$ –space and (alternating) sweep line direction. . . . .	115
6.2. Replacement step of a layer edge. . . . .	116
6.3. Placing heuristic for the ranking of positions. . . . .	118
6.4. Placing heuristic for overlaps. . . . .	124
6.5. Example for the removal of an overlap. . . . .	124
6.6. Insertion of temporary nodes and edges. . . . .	130
6.7. Shortest path routing in the (modified) dual graph. . . . .	133
6.8. Steps of PSL . . . . .	136
6.9. Example for layout with HONG. . . . .	138



---

6.10. Example for zig-zag wall partition and dominating-wall partition. . . . .	140
7.1. Graph properties of test set data. . . . .	146
7.2. Execution times for the reference implementation. . . . .	147
7.3. Execution times for SLS. . . . .	147
7.4. Execution times for ILP. . . . .	148
7.5. Execution times for PSL. . . . .	148
7.6. Comparison of aggregated execution times. . . . .	149
7.7. Maximum/Average edge lengths for SLS. . . . .	150
7.8. Maximum/Average edge lengths for SLS. . . . .	150
7.9. Maximum/Average edge lengths for ILP. . . . .	151
7.10. Maximum/Average edge lengths for PSL. . . . .	151
7.11. Aggregated data for maximum/average edge lengths. . . . .	152
7.12. Aggregated data for area consumption. . . . .	152
7.13. Optimality gap of ILP solving processes. . . . .	154
7.14. Sizes of the ILP models for the process model instances. . . . .	155
7.15. Layering of a cycle. . . . .	155
7.16. Examples of $2.5D$ -visualizations with our approaches . . . . .	158
8.1. A screenshot of the graphical user interface for the CGV. . . . .	162
8.2. A BPMN process diagram in Oryx. . . . .	166
8.3. Layout example in Oryx. . . . .	167
8.4. Overview of a complete process model in the $3D$ -navigator. . . . .	170
8.5. Display of the heads-up-display in the $3D$ -navigator. . . . .	172
8.6. Architecture of GraphArchive. . . . .	176
8.7. Query form of the free search. . . . .	179
8.8. Screenshot of the GraphArchive main page. . . . .	180
8.9. Expanded view of the main page. . . . .	181
8.10. Detailed graph page. . . . .	182
8.11. Complete display of tags. . . . .	182



# Bibliography

## Publications of the Author

- Albrecht, Benjamin, Philip Effinger, Markus Held, and Michael Kaufmann. “An Automatic Layout Algorithm for BPEL Processes”. In: *SoftVis '10: Proceedings of the 5th ACM Symposium on Software visualization*. Salt Lake City, Utah, USA: ACM, 2010.
- Albrecht, Benjamin, Philip Effinger, Markus Held, Michael Kaufmann, and Stephan Kottler. “Visualization of Complex BPEL Models”. In: *Proc. of the 17th International Symposium on Graph Drawing (GD '09)*. LNCS. Springer, 2009, pp. 421–423.
- Bachmaier, Christian, Franz-Josef Brandenburg, Philip Effinger, Carsten Gutwenger, Jyrki Katajainen, Karsten Klein, Miro Spönemann, Matthias Stegmaier, and Michael Wybrow. “The Open Graph Archive: A Community-Driven Effort”. In: *Proc. of the 19th International Symposium on Graph Drawing (GD '11)*. Vol. 7034. 2011, pp. 435–440.
- Bachmaier, Christian, Franz-Josef Brandenburg, Philip Effinger, Carsten Gutwenger, Jyrki Katajainen, Karsten Klein, Miro Spönemann, Matthias Stegmaier, and Michael Wybrow. “The Open Graph Archive: A Community-Driven Effort”. In: *CoRR abs/1109.1465* (2011), pp. 1–10.
- Effinger, Philip. “A 3D Navigator for Business Process Models”. In: *Intl. Workshop on Theory and Applications of Process Visualization (TAProViz12)*. LNBIP. Springer Berlin Heidelberg, 2012.
- Effinger, Philip. “Automatisches Layout von Geschäftsprozessen”. (german). Diploma thesis. Arbeitsbereich Algorithmik, Wilhelm Schickard Institut, Eberhard Karls Universität Tübingen, 2008.

- Effinger, Philip. "Layout Patterns with BPMN Semantics". In: *3rd International Workshop on Business Process Model and Notation (BPMN)*. LNBIP. Springer Berlin Heidelberg, 2011, pp. 130–135.
- Effinger, Philip and Gero Decker. "Layout techniques coupled with Web2.0-based Business Process Modeling". In: *Proc. of the 17th International Symposium on Graph Drawing (GD '09)*. Vol. 5849. LNCS. 2010, pp. 417–418.
- Effinger, Philip, Nicole Jogsch, and Sandra Seiz. "On a Study of Layout Aesthetics for Business Process Models Using BPMN". In: *Business Process Modeling Notation (BPMN2010)*. Vol. 67. LNBIP. Springer, 2010, pp. 31–45.
- Effinger, Philip, Michael Kaufmann, Sascha Meinert, and Matthias Stegmaier. *Graph-Archive - An Online Graph Data Store*. Technical report WSI-2011-03. Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 2011.
- Effinger, Philip, Michael Kaufmann, and Martin Siebenhaller. "Enhancing Visualizations of Business Processes". In: *Proc. of the 16th International Symposium on Graph Drawing (GD '08)*. LNCS. 2009, pp. 437–438.
- Effinger, Philip, Sandra Seiz, and Nicole Jogsch. "Evaluating single features in usability tests for modeling tools". In: *3rd Workshop Methodische Entwicklung von Modellierungswerkzeugen (MEMWe 2011) at Informatik 2011*. LNI. Berlin, 2011.
- Effinger, Philip, Martin Siebenhaller, and Michael Kaufmann. "An Interactive Layout Tool for BPMN". In: *Proc. of the IEEE International Conference on Commerce and Enterprise Computing (CEC2009)*. IEEE Computer Society, 2009, pp. 399–406.
- Effinger, Philip, Martin Siebenhaller, and Michael Kaufmann. *Improving Business Process Visualizations*. Technical report WSI-2009-02. Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 2009.
- Effinger, Philip and Johannes Spielmann. "Lifting Business Process Diagrams to 2.5 Dimensions". In: *IS&T/SPIE Electronic Imaging, Visualization and Data Analysis, (VDA2010)*. Vol. 7530. Proc. SPIE. San Jose, 2010.
- Jogsch, Nicole, Sandra Seiz, Philip Effinger, and Tamara Wehrstein. "Softwareauswahl - auch eine Frage der Usability". In: *Wirtschaftsinformatik und Management 03* (2010), pp. 26–30.
- Seiz, Sandra, Philip Effinger, Nicole Jogsch, and Tamara Wehrstein. *Forschungsprojekt: Usability-Evaluation von BPMN-konformer Geschäftsprozessmodellierungssoftware*. Arbeitsberichte zur Wirtschaftsinformatik 35. (german). Lehrstuhl für Wirtschaftsinformatik, Universität Tübingen, Apr. 2010.

## Publications on Graph Drawing and 2D–Visualization

- Ahmed, Adel, Tim Dwyer, et al. “GEOMI: GEOMETRY for Maximum Insight”. In: *Proc. of the 13th International Symposium on Graph Drawing, GD’05*. 2005, pp. 468–479.
- Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.
- Angelini, Patrizio, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Vincenzo Roselli. “Beyond Clustered Planarity”. In: *CoRR* abs/1207.3934 (2012).
- Bachmaier, Christian, Franz-Josef Brandenburg, Wolfgang Brunner, and Gergö Lovász. “Cyclic Leveling of Directed Graphs”. In: *Proc. of the 16th International Symposium on Graph Drawing, GD’08*. 2008, pp. 348–359.
- Bannister, Michael J. and David Eppstein. “Hardness of Approximate Compaction for Nonplanar Orthogonal Graph Drawings”. In: *Proc. of the 19th International Symposium on Graph Drawing, GD’11*. 2011, pp. 367–378.
- Barth, Wilhelm, Petra Mutzel, and Michael Jünger. “Simple and Efficient Bilayer Cross Counting”. In: *Journal of Graph Algorithms and Applications* 8.2 (2004), pp. 179–194.
- Berg, Mark de, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Second edition. Springer, 2000, p. 367.
- Binucci, Carla and Walter Didimo. “Experiments on Area Compaction Algorithms for Orthogonal Drawings”. In: *Proc. of the 17th Canadian Conference on Computational Geometry, CCCG’05*. 2005, pp. 113–116.
- Brandenburg, Franz J., Michael Himsolt, and Christoph Rohrer. “An Experimental Comparison of Force–Directed and Randomized Graph Drawing Algorithms”. In: *Graph Drawing*. 1995, pp. 76–87.
- Brandes, U., M. Eiglsperger, I. Herman, M. Himsolt, and MS. Marshall. “GraphML Progress Report - Structural Layer Proposal”. In: *Proc. of the 9th Symposium on Graph Drawing (GD’01)*. Springer Verlag, 2001, pp. 501–512.
- Brandes, Ulrik. “Layout of Graph Visualizations”. PhD thesis. <http://www.uni-konstanz/kops/volltexte/1999/255/>: University of Konstanz, 1999.
- Brandes, Ulrik, Michael Kaufmann, Dorothea Wagner, and Markus Eiglsperger. “Sketch-Driven Orthogonal Graph Drawing”. In: *Proc. of the 10th International Symposium on Graph Drawing (GD’02)*. Vol. 2528. LNCS. Springer, 2002, pp. 131–148.
- Brandes, Ulrik and Boris Köpf. “Fast and Simple Horizontal Coordinate Assignment”. In: *Proc. of the 9th International Symposium on Graph Drawing, (GD’01)*. Vol. 2265. LNCS. 2001, pp. 31–44.

- Brandes, Ulrik and Dorothea Wagner. "A Bayesian Paradigm for Dynamic Graph Layout". In: *Proc. of the 5th International Symposium on Graph Drawing (GD '97)*. London, UK: Springer-Verlag, 1997, pp. 236–247.
- Bridgeman, Stina S., Giuseppe Di Battista, Walter Didimo, Giuseppe Liotta, Roberto Tamassia, and Luca Vismara. "Turn-regularity and optimal area drawings of orthogonal representations". In: *Computational Geometry* 16.1 (2000), pp. 53–93.
- Bridgeman, Stina S. and Roberto Tamassia. "Difference Metrics for Interactive Orthogonal Graph Drawing Algorithms". In: *Proc. of the 6th International Symposium on Graph Drawing (GD '98)*. London, UK: Springer-Verlag, 1998, pp. 57–71.
- Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.
- Cornelsen, Sabine and Dorothea Wagner. "Completely Connected Clustered Graphs". In: *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003*. Vol. 2880. Lecture Notes in Computer Science. Springer, 2003, pp. 168–179.
- Di Battista, Giuseppe, Ashin Garg, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. "An Experimental Comparison of Four Graph Drawing Algorithms". In: 7.5–6 (1997), pp. 303–325.
- Di Battista, Giuseppe, Ashin Garg, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. "Drawing Directed Acyclic Graphs: An Experimental Study". In: *J. Comput. Geom. Appl.* 10.6 (2000), pp. 623–648.
- Di Battista, Giuseppe, Walter Didimo, Maurizio Patrignani, and Maurizio Pizzonia. "Orthogonal and Quasi-upward Drawings with Vertices of Prescribed Size". In: *Proc. of the 7th International Symposium on Graph Drawing (GD '99)*. Vol. LNCS 1731. Springer, 1999, pp. 297–310.
- Di Battista, Giuseppe, Peter Eades, Roberto Tamassia, and Ioannis Tollis. *Graph Drawing: Algorithms for the visualization of graphs*. Prentice Hall, 1999, p. 432.
- Diguglielmo, Gilles, Eric Durocher, Philippe Kaplan, Georg Sander, and Adrian Vasiliiu. "Graph Layout for Workflow Applications with ILOG JViews". In: *Proc. of the 10th Symposium on Graph Drawing (GD '02)*. 2002, pp. 362–363.
- Eades, Peter, Xuemin Lin, and W. F. Smyth. "A Fast and Effective Heuristic for the Feedback Arc Set Problem". In: *Inf. Process. Lett.* 47.6 (1993), pp. 319–323.
- Eades, Peter, Kozo Sugiyama, Kazuo Misue, and Wei Lai. "Preserving the mental map of a diagram". In: *Proceedings of Compugraphics*. 1991.
- Eades, P. and N.C. Wormald. "Edge crossings in drawings of bipartite graphs". In: *Algorithmica* 11 (1994), pp. 379–403.
- Eichelberger, Holger. "Aesthetics and Automatic Layout of UML Class Diagrams". PhD thesis. Universität Würzburg, 2005.

- Eiglsperger, Markus. “Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach”. PhD thesis. Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 2003.
- Eiglsperger, Markus, Ulrich Fößmeier, and Michael Kaufmann. “Orthogonal graph drawing with constraints”. In: *Proceedings of the 11th annual ACM-SIAM symposium on Discrete algorithms, SODA '00*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000, pp. 3–11.
- Eiglsperger, Markus and Michael Kaufmann. “An Approach for Mixed Upward Planarization”. In: *Proceedings of 7th International Workshop on Algorithms and Data Structures, WADS 2001*. Vol. 2125. LNCS. Springer, 2001, pp. 352–364.
- Eiglsperger, Markus and Michael Kaufmann. “Fast compaction for orthogonal drawings with vertices of prescribed size”. In: *Proc. of the 9th International Symposium on Graph Drawing (GD '01)*. LNCS. Germany: Springer, 2002, pp. 124–138.
- Eiglsperger, Markus, Martin Siebenhaller, and Michael Kaufmann. “An efficient implementation of Sugiyama’s algorithm for layered graph drawing”. In: *Proc. of the 12th International Symposium on Graph Drawing (GD '04)*. Vol. 3383. LNCS. Springer, 2005, pp. 155–166.
- Eiglsperger, M., C. Gutwenger, M. Kaufmann, J. Kupke, M. Jünger, S. Leipert, K. Klein, P. Mutzel, and M. Siebenhaller. “Automatic layout of UML class diagrams in orthogonal style.” In: *Information Visualization 3.3* (2004), pp. 189–208.
- Erten, Cesim, Philip J. Harding, Stephen G. Kobourov, Kevin Wampler, and Gary V. Yee. “GraphAEL: Graph Animations with Evolving Layouts”. In: *Proc. of the 11th International Symposium on Graph Drawing, GD2003*. Vol. 2912. LNCS. Springer, 2003, pp. 98–110.
- Forster, Michael. “Applying Crossing Reduction Strategies to Layered Compound Graphs”. In: *Proc. of the 10th International Symposium on Graph Drawing (GD '02)*. Vol. 2528. LNCS. Springer, 2002, pp. 276–284.
- Forster, Michael. “Crossings in clustered level graphs”. PhD thesis. Fakultät für Informatik und Mathematik, Universität Passau, 2005.
- Fortune, Steven. “A sweepline algorithm for Voronoi diagrams”. In: *Algorithmica* 2 (1 1987). 10.1007/BF01840357, pp. 153–174.
- Fößmeier, Ulrich. “Orthogonale Visualisierungstechniken für Graphen”. german. PhD thesis. Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 1997.
- Fößmeier, Ulrich and Michael Kaufmann. “Drawing High Degree Graphs with Low Bend Numbers”. In: *Proc. of the 4th Symposium on Graph Drawing (GD '95)*. Vol. 1027. LNCS. Springer, 1995, pp. 254–266.
- Fruchterman, Thomas M. J. and Edward M. Reingold. “Graph Drawing by Force-directed Placement”. In: *Software - Practice and Experience* 21 (11 1991), pp. 1129–1164.

- Gansner, Emden R., Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. "A Technique for Drawing Directed Graphs". In: *IEEE Transactions on Software Engineering* 19.3 (1993), pp. 214–230.
- Garey, M. R. and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- Garey, M. R. and D. S. Johnson. "Crossing Number is NP-Complete". In: *SIAM Journal on Algebraic and Discrete Methods* 4 (3 1983).
- Garg, Ashim and Roberto Tamassia. "A new Minimum Cost Flow Algorithm with Applications to Graph Drawing". In: *Graph Drawing*. Vol. 96. Graph Drawing. 1996, pp. 201–216.
- Goldschmidt, O. and A. Takvorian. "An efficient graph planarization two-phase heuristic". In: *Networks* 24 (1994), pp. 69–73.
- Healy, Patrick and Nikola S. Nikolov. "How to Layer a Directed Acyclic Graph". In: *Proc. of the 9th International Symposium on Graph Drawing (GD'01)*. 2001, pp. 16–30.
- Herman, Ivan, Guy Melançon, and M. Scott Marshall. "Graph Visualization and Navigation in Information Visualization: A Survey". In: *IEEE Trans. Vis. Comput. Graph.* 6.1 (2000), pp. 24–43.
- Jünger, Michael and Petra Mutzel. "2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms". In: *J. Graph Algorithms Appl.* 1.1 (1997), pp. 1–25.
- Kaufmann, Michael and Dorothea Wagner, eds. *Drawing Graphs: Methods and Models*. Vol. 2025. LNCS. Springer, 2001.
- Kitzmann, Ingo, Christoph König, Daniel Lübke, and Leif Singer. "A Simple Algorithm for Automatic Layout of BPMN Processes". In: *Proc. of the IEEE International Conference on Commerce and Enterprise Computing (CEC2009)*. IEEE Computer Society, 2009, pp. 391–398.
- Klau, Gunnar W., Karsten Klein, and Petra Mutzel. "An Experimental Comparison of Orthogonal Compaction Algorithms (Extended Abstract)". In: *Proc. of the 8th International Symposium on Graph Drawing, GD'00*. 2000, pp. 37–51.
- Matuszewski, Christian, Robby Schönfeld, and Paul Molitor. "Using Sifting for k - Layer Straightline Crossing Minimization". In: *Proc. of the 7th International Symposium on Graph Drawing (GD '99)*. Vol. 1731. LNCS. Springer, 1999, pp. 217–224.
- Mutzel, Petra and Gunnar W. Klau. *Quasi-orthogonal drawing of planar graphs*. Tech. rep. MPI-I-98-1-013. MPI Saarbrücken, 1998.
- Nöllenburg, Martin and Alexander Wolff. "A Mixed-Integer Program for Drawing High-Quality Metro Maps". In: *Graph Drawing*. 2005, pp. 321–333.



- Pach, János and Géza Tóth. "Monotone Crossing Number". In: *Proc. of the 19th International Symposium on Graph Drawing (GD'11)*. 2011, pp. 278–289.
- Patrignani, Maurizio. "On the complexity of orthogonal compaction". In: *Computational Geometry* 19.1 (2001), pp. 47–67.
- Pelsmajer, Michael J., Marcus Schaefer, and Daniel Stefankovic. "Crossing Numbers and Parameterized Complexity". In: *Proc. of the 15th International Symposium on Graph Drawing (GD '07)*. 2007, pp. 31–36.
- Resende, Mauricio and Celso C. Ribeiro. "A grasp for graph planarization". In: *NETWORKS: Networks: An International Journal* 29 (1997), p. 3.
- Rinderle, Stefanie, Ralph Bobrik, Manfred Reichert, and Thomas Bauer. "Business Process Visualization - Use Cases, Challenges, Solutions". In: *Proceedings of the Eighth International Conference on Enterprise Information Systems (ICEIS2006)*. 3. 2006, pp. 204–211.
- Sander, Georg. "Graph Layout for Applications in Compiler Construction". In: *Theoretical Computer Science* 217.2 (1999), pp. 175–214.
- Siebenhaller, Martin. "Orthogonal Drawings with Constraints: Algorithms and Applications". PhD thesis. Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 2009.
- Siebenhaller, Martin. "Partitioned Drawings". In: *Proc. of the 14th International Symposium on Graph Drawing (GD '06)*. Vol. 4372. LNCS. Springer, 2006, pp. 252–257.
- Siebenhaller, Martin and Michael Kaufmann. *Drawing Activity Diagrams*. Technical report WSI-2006-02. Eberhard Karls Universität Tübingen: Wilhelm-Schickard-Institut, 2006.
- Siebenhaller, Martin and Michael Kaufmann. "Drawing Activity Diagrams". In: *Proc. of ACM 2006 Symposium on Software Visualization, SoftVis 2006*. ACM, 2006, pp. 159–160.
- Siebenhaller, Martin and Michael Kaufmann. "Mixed upward planarization - fast and robust". In: *Proc. of the 13th Symposium on Graph Drawing (GD '05)*. Vol. 3843. LNCS. Springer, 2005, pp. 522–523.
- Six, Janet M. and Ioannis G. Tollis. "Automated Visualization of Process Diagrams". In: *Proc. of the 9th International Symposium on Graph Drawing (GD '01)*. Springer, 2002, pp. 45–59.
- Sugiyama, K., S. Tagawa, and M Toda. "Methods for visual understanding of hierarchical system structures". In: *IEEE Transactions on Systems, Man, and Cybernetics SMC-11,2* (1981), pp. 109–125.
- Tamassia, Robert. "On Embedding a Graph in the Grid with the Minimum Number of Bends". In: *SIAM Journal on Computing* 16(3) (1987), pp. 421–444.

- Tamassia, Roberto, Giuseppe Di Battista, and Carlo Batini. "Automatic graph drawing and readability of diagrams". In: *IEEE Trans. Syst. Man Cybern.* 18.1 (1988), pp. 61–79.
- Tunkelang, Daniel. *A practical approach to drawing undirected graphs*. Tech. rep. Carnegie Mellon, 1994.
- Wiese, Roland, Marcus Eiglsperger, and Michael Kaufmann. "yFiles: Visualization and Automatic Layout of Graphs". In: *Proc. of the 9th International Symposium on Graph Drawing (GD '01)*. LNCS 2265. Springer, 2001, pp. 453–454.
- Yang, Yun, Wei Lai, Jun Shen, Xiaodi Huang, Jun Yan, and Lukman Setiawan. "Effective Visualisation of Workflow Enactment". In: *Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference (APWeb04)*. 2004, pp. 794–803.
- Zhao, Xin, Jun Han, and Yaling Huang. "An Automatic Layout Function in BPEL Visual Modeling Tool". In: *SNPD '09: Proc. of the 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 14–17.

## Publications on 3D–Visualization

- Ahmed, Adel and Seok-Hee Hong. "Navigation techniques for 2.5D graph layout". In: *Proc. of the 6th International Asia-Pacific Symposium on Visualization (APVIS07)*. IEEE, 2007, pp. 81–84.
- Balzer, Michael and Oliver Deussen. "Hierarchy Based 3D Visualization of Large Software Structures". In: *IEEE Visualization*. 2004, p. 4.
- Betz, Stefanie, Daniel Eichhorn, Susan Hickl, Stefan Klink, Agnes Koschmider, Yu Li, Andreas Oberweis, and Ralf Trunko. "3D Representation of Business Process Models". In: *Modellierung betrieblicher Informationssysteme (MobIS)*. Vol. 141. LNI. GI, 2008, pp. 73–87.
- Brown, Ross. "Conceptual modelling in 3D Virtual Worlds for Process Communication". In: *Proc. of the 7th Asia-Pacific Conference on Conceptual Modelling, APCCM 2010*. 2010, pp. 25–32.
- Bruß, Ingo and Arne Frick. "Fast interactive 3-D graph visualization". In: *Proc. of the Symposium on Graph Drawing (GD '95)*. Vol. 1027. LNCS. Springer, 1995, pp. 99–110.
- Dwyer, Tim. "Three Dimensional UML Using Force Directed Layout". In: *Proc. of the Australasian Symposium on Information Visualisation (InVis.au '01)*. Vol. 9. CRPIT. Australian Computer Society, 2001, pp. 77–85.
- Dwyer, Tim. "Two and a Half Dimensional Visualisation of Relational Networks". PhD thesis. The University of Sydney, 2004.

- Dwyer, Tim and Peter Eades. "Visualising a Fund Manager Flow Graph with Columns and Worms". In: *Sixth International Conference on Information Visualisation (IV'02)*. 2002, pp. 147–152.
- Dwyer, Tim and Peter Eckersley. "WilmaScope - An Interactive 3D Graph Visualisation System". In: *Proc. of the 9th International Symposium on Graph Drawing (GD '01)*. Vol. 2265. LNCS. Springer, 2001, pp. 442–443.
- Frunk, Alexander, Armin Bruckhoff, and Michael Kern. "3D visualisation of code structures in Java software systems". In: *Proc. of the ACM 2006 Symposium on Software Visualization (SOFTVIS '06)*. ACM, 2006, pp. 145–146.
- Garg, Ashim and Roberto Tamassia. "GIOTTO3D: A System for Visualizing Hierarchical Structures in 3D". In: *Graph Drawing*. 1996, pp. 193–200.
- Hoipkemier, Benjamin N., Nicholas A. Kraft, and Brian A. Malloy. "3D visualization of class template diagrams for deployed open source applications". In: *Proc. of the 18th Intern. Conference on Software Engineering and Knowledge Engineering*. 2006.
- Hong, Seok-Hee. "MultiPlane: A New Framework for Drawing Graphs in Three Dimensions". In: *Proc. of the 13th International Symposium on Graph Drawing (GD '05)*. Vol. 3843. LNCS. Springer, 2005, pp. 514–515.
- Hong, Seok-Hee and Tom Murtagh. "Visualisation of Large and Complex Networks Using PolyPlane". In: *Proc. of the 12th International Symposium on Graph Drawing (GD '04)*. Vol. 3383. LNCS. Springer, 2004, pp. 471–481.
- Hong, Seok-Hee and Nikola S. Nikolov. "Layered drawings of directed graphs in three dimensions". In: *Proc. of the 2005 Asia-Pacific symposium on Information visualisation - Volume 45*. APVis '05. Sydney, Australia: Australian Computer Society, Inc., 2005, pp. 69–74.
- Hong, Seok-Hee, Nikola S. Nikolov, and Alexandre Tarassov. "A 2.5D Hierarchical Drawing of Directed Graphs". In: *J. Graph Algorithms Appl.* 11.2 (2007), pp. 371–396.
- Jainek, Werner. "y25 - Graphs in 2.5 Dimensions". (german). Study thesis. Eberhard Karls Universität Tübingen: Arbeitsbereich Algorithmik, Wilhelm-Schickard-Institut, 2006.
- Mian, A. S., M. Bennamoun, and R. A. Owens. "3D Recognition and Segmentation of Objects in Cluttered Scenes". In: *Applications of Computer Vision and the IEEE Workshop on Motion and Video Computing, IEEE Workshop on 1* (2005), pp. 8–13.
- Munzner, Tamara. "H3: laying out large directed graphs in 3D hyperbolic space". In: *Proc. of the IEEE Symposium on Information Visualization (INFOVIS '97)*. IEEE Computer Society, 1997, pp. 2–10.

- Ostry, Diethelm Ironi. “Some Three-Dimensional Graph Drawing Algorithms”. MA thesis. Department of Computer Science and Software Engineering, University of Newcastle, 1996.
- Pilgrim, Jens von and Kristian Duske. “GEF3D: a framework for two-, two-and-a-half-, and three-dimensional graphical editors”. In: *Proc. of the ACM 2008 Symposium on Software Visualization (SOFTVIS '08)*. ACM, 2008, pp. 95–104.
- Reiss, Steven P. “3-D Visualization of Program Information”. In: *DIMACS International Workshop on Graph Drawing (GD '94)*. Vol. 894. LNCS. Springer, 1994, pp. 12–24.
- Schönhage, Bastiaan, Alex van Ballegooij, and Anton Eliëns. “3D gadgets for business process visualization - a case study”. In: *Web3D Symposium*. 2000, pp. 131–138.
- Spielmann, Johannes. “Werkzeuge für Graphen in 2.5 Dimensionen”. (german). Diploma thesis. Arbeitsbereich Algorithmik, Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 2009.
- Teyseyre, Alfredo R. and Marcelo R. Campo. “An Overview of 3D Software Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), pp. 87–105.
- Ware, Colin. “Designing with a 2.5D attitude”. In: *Information Design Journal* 10.3 (2001), pp. 255–262.
- Ware, Colin, David Hui, and Glenn Franck. “Visualizing object oriented software in three dimensions”. In: *Proc. of the 1993 conference of the Centre for Advanced Studies on Collaborative research (CASCON '93)*. Toronto, Ontario, Canada: IBM Press, 1993, pp. 612–620.
- West, Stephen, Ross A. Brown, and Jan C. Recker. “Collaborative business process modeling using 3D virtual environments”. In: *16th Americas Conference on Information Systems*. Association for Information Systems (AIS), 2010.

## Publications on Aesthetics

- Agarwal, Ritu, Prabuddha De, and Atish P. Sinha. “Comprehending Object and Process Models: An Empirical Study”. In: *IEEE Trans. Software Eng.* 25.4 (1999), pp. 541–556.
- Ambler, Scott W. *The Elements of UML 2.0 Style*. Cambridge University Press, 2005.
- Apfelbacher, R., A. Knopfel, P. Aschenbrenner, and S. Preetz. *FMC Visualization Guidelines*. online. [http://www.fmc-modeling.org/visualization\\_guidelines](http://www.fmc-modeling.org/visualization_guidelines). 2006.
- Bobrik, Ralph, Manfred Reichert, and Thomas Bauer. “View-Based Process Visualization”. In: *Proc. of the 5th International Conference on Business Process Management (BPM '07)*. Vol. 4714. LNCS. Springer, 2007, pp. 88–95.

- Coleman, M. K. and D. S. Parker. "Aesthetics-based Graph Layout for Human Consumption". In: *Software – Practice and Experience* 26.12 (1996), pp. 1415–1438.
- Genero, Marcela, Geert Poels, and Mario Piattini. "Defining and validating metrics for assessing the understandability of entity-relationship diagrams". In: *Data Knowl. Eng.* 64.3 (2008), pp. 534–557.
- Green, Thomas R.G. and Alan F. Blackwell. *A tutorial on cognitive dimensions*. online. <http://www.cl.cam.ac.uk/%7Eaafb21/CognitiveDimensions/CDtutorial.pdf>, last accessed (2010-05-31). 1998.
- Hahn, Jungpil and Jinwoo Kim. "Why are some diagrams easier to work with? Effects of diagrammatic representation on the cognitive intergration process of systems analysis and design". In: *ACM Trans. Comput.-Hum. Interact.* 6.3 (1999), pp. 181–213.
- Huang, Weidong, Peter Eades, and Seok-Hee Hong. "Beyond time and error: a cognitive approach to the evaluation of graph drawings". In: *BELIV '08: Proceedings of the 2008 conference on BEyond time and errors*. Florence, Italy: ACM, 2008, pp. 1–8.
- Huang, Weidong, Seok-Hee Hong, and Peter Eades. "Effects of Crossing Angles". In: *IEEE Pacific Visualization Symposium 2008, PacificVis*. 2008, pp. 41–46.
- Huang, Weidong, Seok-Hee Hong, and Peter Eades. "Effects of Sociogram Drawing Conventions and Edge Crossings in Social Network Visualization". In: *J. Graph Algorithms Appl.* 11.2 (2007), pp. 397–429.
- Maier, Sonja and Mark Minas. "Interactive diagram layout". In: *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*. CHI EA '10. New York, NY, USA: ACM, 2010, pp. 4111–4116.
- Mendling, Jan, Hajo A. Reijers, and Jorge Cardoso. "What Makes Process Models Understandable?" In: *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*. Ed. by M. Rosemann G. Alonso P. Dadam. Vol. 4714. LNCS. 2007, pp. 48–63.
- Petre, Marian. "Cognitive dimensions 'beyond the notation'". In: *J. Vis. Lang. Comput.* 17.4 (2006), pp. 292–301.
- Petre, Marian. "Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming". In: *Commun. ACM* 38.6 (1995), pp. 33–44.
- Purchase, Helen C. "Which aesthetic has the greatest effect on human understanding?" In: *Proc. of the 5th Symposium on Graph Drawing (GD '97)*. Vol. 1353. LNCS. Springer, 1997, pp. 248–261.
- Purchase, Helen C., Jo-Anne Alder, and David A. Carrington. "User Preference of Graph Layout Aesthetics: A UML Study". In: *Proc. of the 8th International Symposium on Graph Drawing (GD '00)*. Vol. 1984. LNCS. Springer, 2001, pp. 5–18.

- Purchase, Helen C., Robert F. Cohen, and Murray I. James. "An Experimental Study of the Basis for Graph Drawing Algorithms". In: *ACM Journal of Experimental Algorithmics* 2.4 (1997), p. 4.
- Purchase, Helen C., Robert F. Cohen, and Murray I. James. "Validating Graph Drawing Aesthetics". In: *Graph Drawing*. 1995, pp. 435–446.
- Schrepfer, Matthias, Johannes Wolf, Jan Mendling, and Hajo A. Reijers. "The Impact of Secondary Notation on Process Model Understanding". In: *PoEM*. 2009, pp. 161–175.
- Sun, Dabo and Kenny Wong. "On Evaluating the Layout of UML Class Diagrams for Program Comprehension". In: *IWPC*. 2005, pp. 317–326.
- Ware, Colin. *Information Visualization: Perception for Design*. Second Edition. Morgan Kaufmann, 2004.
- Ware, Colin and Glenn Franck. "Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions". In: *ACM Trans. Graph.* 15.2 (1996), pp. 121–140.
- Ware, Colin and Glenn Franck. "Viewing a Graph in a Virtual Reality Display is Three Times as Good as 2D Diagram". In: *IEEE Conference on Visual Languages, (VL1994)*. 1994, pp. 182–183.
- Ware, Colin, Helen C. Purchase, Linda Colpoys, and Matthew McGill. "Cognitive measurements of graph aesthetics". In: *Information Visualization* 1.2 (2002), pp. 103–110.
- Wettel, Richard and Michele Lanza. "Program Comprehension through Software Habitability". In: *Proc. of the 15th International Conference on Program Comprehension (ICPC '07)*. IEEE Computer Society, 2007, pp. 231–240.
- Wittenburg, K. and L. Weitzman. "Qualitative Visualization of Processes: Attributed Graph Layout and Focusing Techniques". In: *Proc. of the 4th International Symposium on Graph Drawing (GD '96)*. Vol. 1190. LNCS. Springer, 1997, pp. 401–408.

## Publications on Business Process Management

- Allweyer, Thomas. *BPMN 2.0*. Books on Demand, 2010, p. 156.
- Alves, Alexandre et al. *Web Services Business Process Execution Language Version 2.0*. Standard. OASIS, Apr. 11, 2007.
- Brown, Ross A. and Jan C. Recker. "Improving the Traversal of Large Hierarchical Process Repositories". In: *20th Australasian Conference on Information Systems*. Monash University, Melbourne, 2009.

- Dumas, Marlon, Luciano García-Bañuelos, and Artem Polyvyanyy. “Unraveling Unstructured Process Models”. In: *Business Process Modeling Notation (BPMN2010)*. Vol. 67. LNBIP. 2010, pp. 1–7.
- Fahland, Dirk, Cédric Favre, Barbara Jobstmann, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. “Instantaneous Soundness Checking of Industrial Business Process Models”. In: *Proceedings of the 7th International Conference on Business Process Management, BPM 2009*. Ed. by Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo Reijers. Vol. 5701. LNCS. Springer, 2009, pp. 278–293.
- Held, Markus and Wolfgang Blochinger. “Collaborative BPEL Design in a Rich Internet Application”. In: *CCGRID '08: 8th International Symposium on Cluster Computing and the Grid*. Lyon: IEEE Computer Society Press, 2008, pp. 202–209.
- Held, Markus and Wolfgang Blochinger. “Structured Collaborative Workflow Design”. In: *Future Generation Computer Systems* 25.6 (2009), pp. 638–653.
- Jablonski, Stefan and Manuel Götz. “Perspective Oriented Business Process Visualization”. In: *Business Process Management Workshops*. Vol. 4928. LNCS. Springer, 2007, pp. 144–155.
- Jensen, Kurt. *Coloured Petri nets: basic concepts, analysis methods, and practical use*. Vol. 2. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1996.
- Kim, Gun-Woo, Seung Hoon Lee, Jae Hyung Kim, and Jin Hyun Son. “An Effective Algorithm for Business Process Mining Based on Modified FP-Tree Algorithm”. In: *Proc. of the Second International Conference on Communication Software and Networks (ICCSN '10)*. IEEE Computer Society, 2010, pp. 119–123.
- Kopp, Oliver, Daniel Martin, Daniel Wutke, and Frank Leymann. “The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages”. In: *Enterprise Modelling and Information Systems Architectures* 4.1 (2009), pp. 3–13.
- Kunze, Matthias, Alexander Luebbe, Matthias Weidlich, and Mathias Weske. “Towards Understanding Process Modeling – The Case of the BPM Academic Initiative”. In: *Third International Workshop on Business Process Model and Notation (BPMN 2011)*. Vol. 95. LNBIP. Springer, 2011, pp. 44–58.
- Polyvyanyy, Artem, Luciano García-Bañuelos, and Marlon Dumas. “Structuring Acyclic Process Models”. In: *Proc. of the 8th International Conference on Business Process Management (BPM '10)*. Vol. 6336. LNCS. 2010, pp. 276–293.
- Polyvyanyy, Artem, Sergey Smirnov, and Mathias Weske. “The Triconnected Abstraction of Process Models”. In: *Proceedings of the 7th International Conference on Business Process Management (BPM '09)*. Vol. 5701. LNCS. Springer, 2009, pp. 229–244.

- Reijers, Hajo and Jan Mendling. “Modularity in Process Models: Review and Effects”. In: *Proc. of the 6th International Conference on Business Process Management, BPM2008*. Vol. 5240. LNCS. Springer, 2008, pp. 20–35.
- Silver, Bruce. *BPMN Method and Style*. Second edition. Code-Cassidy Press, 2011.
- Smirnov, Sergey, Matthias Weidlich, Jan Mendling, and Mathias Weske. “Action Patterns in Business Process Models”. In: *ICSOC/ServiceWave*. 2009, pp. 115–129.
- Smirnov, Sergey, Matthias Weidlich, Jan Mendling, and Mathias Weske. “Object-Sensitive Action Patterns in Process Model Repositories”. In: *Business Process Management Workshops*. 2010, pp. 251–263.
- White, Stephen A. *Introduction to BPMN*. online. online: <http://www.bpmn.org>. United States: IBM Corporation, 2004.
- White, Stephen A. *Process Modeling Notations and Workflow Patterns*. online: <http://www.bpmn.org>. IBM Corporation. 2004.
- White, Stephen A. *Using BPMN to Model a BPEL Process*. online: <http://www.bpmn.org>. IBM Corporation. United States, 2005, p. 18.
- Yongchareon, Sira, Chengfei Liu, Xiaohui Zhao, and Marek Kowalkiewicz. “BPMN Process Views Construction”. In: *Database Systems for Advanced Applications*. Ed. by Hiroyuki Kitagawa, Yoshiharu Ishikawa, Qing Li, and Chiemi Watanabe. Vol. 5981. LNCS. Springer, 2010, pp. 550–564.

## Other Publications

- Achterberg, Tobias. “SCIP: Solving constraint integer programs”. In: *Mathematical Programming Computation* 1.1 (2009), pp. 1–41.
- Gehring, Uwe W. and Cornelia Weins. *Grundkurs Statistik für Politologen und Soziologen*. 5th ed. (german). VS Verlag für Sozialwissenschaften, 2009, p. 46.
- Hogg, Robert V., Joseph McKean, and Allen T. Craig. *Introduction to Mathematical Statistics*. 7th edition. Prentice Hall, 2012.
- Karp, R. M. “Reducibility Among Combinatorial Problems”. In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. Plenum Press, 1972, pp. 85–103.
- Nöllenburg, Martin. *Integer-Programming methods in graph drawing*. Presentation slides, online, <http://bowman.infotech.monash.edu.au/cbldd07/>. 2007.
- Rudell, Richard. “Dynamic variable ordering for ordered binary decision diagrams”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’93)*. 1993, pp. 42–47.
- Stegmaier, Matthias. “GraphDB im Web”. (german). Study thesis. Eberhard Karls Universität Tübingen: Arbeitsbereich Algorithmik, Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 2011.





