RZ 3825 (#Z1105-002) 05/01/2011
Computer Science 16 pages

Research Report

Edges, Structure, and Constraints: The Layout of Business Process Models

Thomas Gschwind[1], Jakob Pinggera[2], Stefan Zugal[2], Hajo A. Reijers[3], and Barbara Weber[2]

[1]IBM Research – Zurich, 8803 Rüschlikon, Switzerland
E-mail: thg@zurich.ibm.com

[2]University of Innsbruck, Austria
E-mail: {jakob.pinggera, stefan.zugal, barbara.weber}@uibk.ac.at

[3]Eindhoven University of Technology, The Netherlands
E-mail: h.a.reijers@tue.nl

Edges, Structure, and Constraints: The Layout of Business Process Models

Thomas Gschwind[1], Jakob Pinggera[2], Stefan Zugal[2], Hajo A. Reijers[3], and Barbara Weber[2]

[1] IBM Research - Zurich, Switzerland
`thg@zurich.ibm.com`
[2] University of Innsbruck, Austria
`jakob.pinggera|stefan.zugal|barbara.weber@uibk.ac.at`
[3] Eindhoven University of Technology, The Netherlands
`h.a.reijers@tue.nl`

Abstract. How a business process model is laid out will affect the ease of making sense of it. Existing layout features in process modeling tools often rely on generic insights on graph representation, but do not take the specific properties of business process models into account. In this paper, we propose an algorithm that is based on a set of constraints which are specifically identified towards establishing a readable layout of a process model. By exploiting the structure of the process model, the proposed algorithm allows for the computation of the final layout in linear time. An empirical evaluation shows that the proposed algorithm indeed provides support beyond what moderately experienced process modelers undertake to lay out their process models.

1 Introduction

Business process models serve a wide variety of purposes. Thereby, a distinction can be made between models that are to be read by *humans* versus those to be read by *machines* [1]. In the latter case, one may think of *workflow specifications*, as enacted by process-aware information systems, or *simulation models*, which are used to estimate a certain measure's effect. Our concern in this paper is with the former category, i.e., those models that are studied by humans to make sense of how organizational operations are related to one another. It is crucial that such models are understandable to end users from very different backgrounds [1].

The *graphical layout* of a process model has been named as affecting the ease with which a human reader can access the information in such a model [2, 3]. When one is concerned with simplifying the task of reading a process model, layout seems a highly attractive angle. After all, in many situations, the graphical positioning of model elements is at the discretion of the modeler, who is creating the model, or even the user, who wants to read the model. Also, layout, as part of what is known as a model's *secondary notation* [4], does not affect the formal meaning of the model. In this way, focusing on a model's layout allows for a separation of concerns with respect to other quality aspects (e.g., a model's semantic quality).

A considerable body of knowledge exists on how to layout graphical models in order to improve their readability [5, 6]. What is currently missing is a clear understanding of how the specific characteristics of business process models should be taken into account when applying these insights. This hampers the transfer of such knowledge to its practical application. In that respect, it is telling that despite a huge availability of advanced process modeling tools (e.g., IBM Blueworks, Oryx, BPM|one), none of these provide automated layout features beyond elementary alignment operations.

This paper presents an efficient algorithm for laying out business process models. Our approach has been to identify a set of favorable layout constraints from literature, which have formed the basis for the development of an algorithm that enforces these constraints. The process structure tree as presented in [7] is an important basis, as it provides the hierarchical abstraction of the semantics of the underlying business process. Furthermore, we have validated in an empirical setting whether and to what extent the identified constraints are followed by moderately experienced business process modelers. This insight is valuable to establish whether the proposed algorithm provides effective support beyond what modelers already do.

The contribution of the proposed algorithm is that it ensures the application of state-of-the-art knowledge to favorably display business process models for a human readership. The approach has been developed for BPMN models that are modeled from left-to-right, but the presented insights can be easily transferred to other flow-oriented languages and other linear model orientations. Our vision is that the algorithm can be implemented in many electronic modeling environments, in this way improving the quality of the models being created without putting an extra load on the modelers themselves. In fact, the provided support may even alleviate a modeler's task, specifically in situations when her modeling experience is limited. When process models are being used within an electronic environment, which is an increasingly realistic option [8], the proposed algorithm directly supports the reader herself in arranging the model in a highly readable form (regardless of the original layout).

The remainder of this paper is structured as follows. Section 2 starts with examining related work. Then, Section 3 discusses the constraints that seem wise to follow when laying out business process models. Section 4 introduces the algorithm that takes into account these constraints to provide automated layout support. Subsequently, Section 5 reports our evaluation of the use of the constraints and discusses the algorithm's complexity. Finally, Section 6 concludes the paper with a summary and outlook.

2 Related Work

The importance of laying out business process models has been identified in [2]. While this work confirms the importance of a good layout of business process models for their understandability, we present an algorithm for automatically laying out business process models and verify that the underlying layout characteristics improve over the modeling behavior of process modelers, hence complementing that work.

A simple algorithm to automatically lay out business process models has been presented in [9]. This is done by arranging elements in a topological order and using a grid that allows to insert new elements by inserting rows and hence pushing other elements aside to make room for new elements. This approach by itself may create space-consuming layouts, hence heuristics are used to improve on this situation. The runtime complexity of this approach has not been assessed.

Also related to our approach, although not directly related to business process models, are general-purpose graph layout algorithms. In this context, graphviz, utilizing the dot algorithm [10], is one of the most popular open-source tools. The dot algorithm uses a four stage approach for laying out graphs. First, ranks are assigned to the nodes and the vertex order is determined for minimizing edge crossings. Afterwards, positions are assigned to the nodes before computing splines for laying out the graph's edges. However, the usage of splines (i.e., curved edges) is not desirable for business process models which are typically laid out with Manhattan layout (i.e., orthogonal edges). Additionally, the dot algorithm tries to minimize the length of edges, another characteristic not necessarily desirable for laying out business process models (cf. Section 3).

In [11], Di Battista presents a set of algorithms for plane representations of acyclic digraphs, supporting visibility graphs, grid drawings, and straight line drawings ranging from $O(n)$ to $O(n \log n)$. The grid drawings are similar to our approach, but do not support cyclic or non-planar graphs. Similar approaches can be found in [12, 13].

A problem related to laying out business process models is the layout of trees. In [14], Hasan presents an algorithm to draw a tree in a compact form. We use this algorithm for laying out the spanning tree of unstructured fragments and extend it to support the layout of non-tree edges (cf. Section 4.2).

The ILOG JViews Component Suite is a set of components for Web-based user interfaces, including a component specifically designed for building graphical user interfaces for workflow applications [15]. It is a general-purpose graph drawing tool based on a constraint system allowing, among others, the definition of relations between node positions and the placement of incoming and outgoing edges. Similarly, the layout algorithm presented in this paper relies on a set of pre-defined constraints specifically tailored for the needs of business process models (cf. Section 3).

yFiles is an extensive Java class library that provides algorithms and components enabling the analysis, visualization, and the automatic layout of graphs, diagrams, and networks [16]. yFiles is used by WebSphere Business Modeler for laying out business process models, even though it does not provide a specific algorithm for business process models. The underlying layout algorithm is proprietary and its runtime behavior is unknown.

3 Constraints

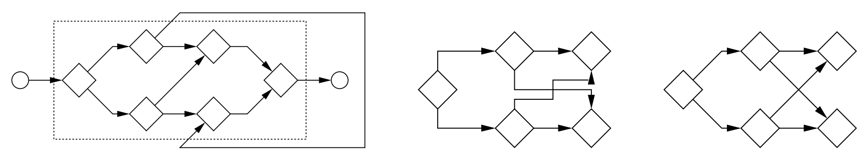
An important consideration when laying out business process models is the set of constraints that should be followed. A small set of desirable properties for laying out business process models has already been presented in [5, 2], which will be extend subsequently. Additionally, we considered constraints which have been discussed in the context of laying out trees [17], but are equally desirable for business process models. Our layout algorithm is based on the following constraints.

C1. Edges should be drawn in the direction of the process's flow. Edges running in the opposite direction of the flow of the process make it hard for humans to understand the process, since it is no longer sufficient to scan the process in one direction. One exception to that rule are edges that are part of a cycle. In such a case, a single edge of that cycle, the back edge, has to flow backwards. This constraint is implicitly present in [2, 5].

C2. Incoming and outgoing edges must be separated. Incoming and outgoing edges can be grasped more easily if they are separated, especially if the diagram is large and has been zoomed out to such a degree that the arrowheads can no longer be identified clearly. Commonly, incoming edges are drawn on the left side of a node, whereas the outgoing edges are drawn on its right side. This constraint has been implemented in commercial modeling tools such as [18] and serves to support the flow of the business process.

C3. Edge crossings should be minimized. The number of edge crossings should be minimized because edge crossings are known to have a significant impact on the understandability of business process models [5, 2]. Specifically, upward planar graphs should be drawn in a planar way. Upward planar graphs are similar to planar graphs (i.e., graphs that can be drawn without crossing edges), but allow edge crossings if these edges would have to encircle the entire process to be drawn in a planar way (cf. Fig. 1(a)).

C4. Bendpoints of edges should be minimized. Edges should be drawn with as little bendpoints as possible [5, 2]. Each bendpoint changes the flow of an edge and makes it harder to trace the source and destination of an edge, especially, if edges have to cross other edges.



(a) Avoid to Route Edges Around the Process (b) Manhattan Layout (c) Relaxed Manhattan Layout

Fig. 1. Constraints

C5. A Manhattan layout of edges must be used. Edges should preferably use an orthogonal layout that consists of horizontal and vertical lines. This makes it easier to follow edge crossings. However, strictly following the Manhattan layout can lead to edges with an unnecessary high number of bendpoints (cf. Fig. 1(b)), contradicting constraint C4 (i.e., bendpoints of edges should be minimized). Hence, we slightly relax this constraint and allow the first or the last segment of an arrow to be drawn diagonally (cf. Fig. 1(c)). This constraint is commonly present in business process modeling layout algorithms such as [9, 18].

C6. Minimality should be applied. The business process should consume as little space as necessary [9]. In most cases, this constraint additionally achieves symmetry—another desired property—by putting branches together as close as possible from all sides. However, symmetry may be violated while minimizing bendpoints, which also is considered more important in [2]. Hence, we do not list symmetry explicitly as a constraint.

There is a small number of prominent constraints that we do not consider explicitly. *Clustering* is generally accomplished through SESE decomposition (cf. Section 4). However, the actual position of a modeling element frequently depends on its semantics. Consequently, we did not include *clustering* in the set of constraints underlying the layout algorithm. *Edge length minimization* is intentionally not considered by our algorithm, because of its undesirable effects on business process models. In particular, adherence to it would cause decisions to be pushed away from related elements. For example, assume that elements *a* to *c* in Fig. 2 belong together. To minimize edge length, however, decision *c* and the following elements would have to be pushed towards the right, in this way separating elements *a* and *b* from *c*.

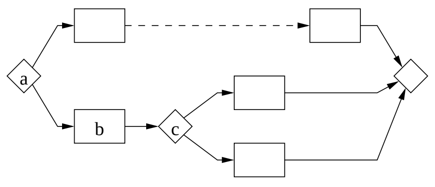


Fig. 2. Edge length minimization

4 Layout Algorithm

So far, we have introduced desirable constraints for the layout of business process models. In this section, we describe how these constraints are taken into account by the proposed layout algorithm. Roughly speaking, the algorithm follows a three-phase approach. First, the process model to be laid out is pre-processed. Second, the resulting process model is decomposed into Single-Entry Single-Exit (SESE) fragments [7]. Third, based upon the SESE fragment's structure, the layout is computed. In the following, we will discuss each phase in more detail.

Pre-Processing. In order to facilitate the model's decomposition into SESE fragments, the algorithm pre-processes gateways with multiple incoming and outgoing edges. More specifically, respective gateways are split into two gateways, one having all the originally incoming edges and the other covering all the originally outgoing edges. Similarly, multiple incoming and multiple outgoing edges of non-gateways (e.g., activities) are separated into an additional gateway. To illustrate the pre-processing step, Fig. 3 shows two exemplary pre-processing transformations. In Fig. 3 (a) an additional gateway is introduced to have *either* multiple incoming *or* multiple outgoing edges. Similarly, in Fig. 3 (b), two gateways are introduced to make the join gateways and split gateways explicit.

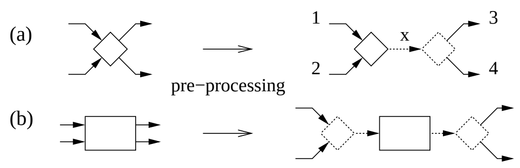


Fig. 3. Pre-Processing Steps Applied to the Business Process Model

By applying these transformations, the business process model remains semantically equivalent to the original model, but each node has either multiple incoming edges or multiple outgoing edges. This pre-processing step ensures constraint C2 (i.e., incoming and outgoing edges must be separated). The gateways introduced during this pre-processing step are used for the computation of the edge ordering only; they can be removed from the final layout of the business process model, i.e., the algorithm does not change the structure of the model.

Due to the split, the only correct edge orderings for the left gateway are all permutations of $\{1, 2, x\}$ and for the right $\{x, 3, 4\}$. This constrains the permutations allowed for edges 1 to 4. to the permutations of $\{1, 2, \{3, 4\}\}$ which makes sure that edges 1 and 2 and edges 3 and 4 stay together. Since edges are drawn in a clock-wise order, orderings $\{1, 2, 3, 4\}$ and $\{2, 3, 4, 1\}$ are the same.

Decomposition into Single-Entry Single-Exit (SESE) Fragments. The algorithm uses the Refined Process Structure Tree (RPST) algorithm to decompose the business process into SESE fragments (for a detailed description we refer to [7]). To illustrate this, Fig. 4 shows an exemplary SESE decomposition with SESE fragments A, B, C and D. As the name suggests, each SESE fragment has *exactly one* incoming and *exactly one* outgoing edge, irrespective of the internal structure of the fragment. For instance, the internal structure of fragment C is a sequence of activities, whereas fragment D consists of a branched construct. Furthermore, SESE fragments can be embedded in other SESE fragments: Note how fragment B aggregates fragment C and fragment D to a SESE fragment.

Layout Computation. After the algorithm has pre-processed and analyzed the process model's structure, the actual computation of the layout is carried out. In

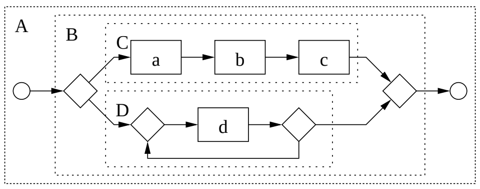


Fig. 4. A Business Process and its Major Sub-Fragments

particular, the algorithm makes use of the SESE's tree-structuredness to compute the layout recursively and bottom-up:

1. Categorize SESE fragment as (i) structured, (ii) unstructured or (iii) non-planar unstructured
2. According to this classification, compute the layout and size for the SESE fragment (as detailed in Sections 4.1, 4.2 and 4.3)
3. Use the SESE fragment's size to position it within the tree of SESE fragments

Consider the process model depicted in Fig. 4, which illustrates the algorithm. As discussed above, the SESE decomposition partitions the process model into fragments A, B, C and D. According to the nesting of SESE fragments, the algorithm starts by classifying and laying out fragments C and D. Since the size of fragments C and D is now known, it can be used for positioning the fragments within fragment B. Finally, fragment A is laid out—in turn using the spatial information about fragment B.

This concludes the description of the algorithm's overall structure. In the following we detail how the internal structure of SESE fragments is laid-out.

4.1 Structured Fragments

Structured fragments can be classified as *sequences*, *branching fragments*, and *structured loops*.

Sequences consist of a sequence of activities or fragments (cf. fragments A and C in Fig. 4). Sequences are laid out as straight lines, ensuring in this way that the exit of one fragment is on the same height as the entry of the next fragment.

Branching fragments consist of a diverging (i.e., splitting) gateway as entry and a converging (i.e., joining) gateway as exit. To illustrate a branching fragment, we refer to fragment B in Fig. 4. Branching fragments are laid out by first arranging the individual branches vertically. Then, the diverging node is put to the left of the branches and the converging node is put to the right, as shown in Fig. 5. When laying out the branches, they can be optimized by using the actual shape of the branches and pushing them vertically together as shown in Fig. 5 to minimize the fragment's area.

Structured loops consist of a converging gateway followed by an optional fragment in turn followed by a diverging gateway. The latter has an exit branch and one or more branches that loop back (cf. fragment D in Fig. 4). The structure

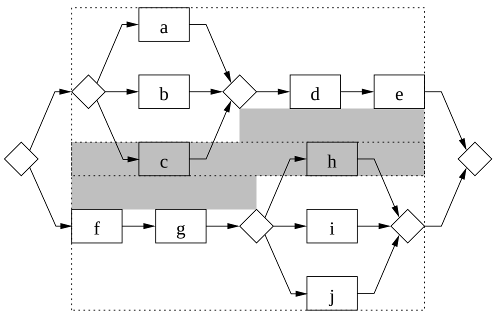


Fig. 5. Layout of Branching Fragments

of a loop fragment and a sequence are almost identical, hence we use a similar strategy for computing the layout. In particular, we lay out the converging gateway, the optional body and the diverging gateway like a sequence. In addition, the loop-back branches are laid out like the branches of a branching fragment.

4.2 Unstructured Fragments

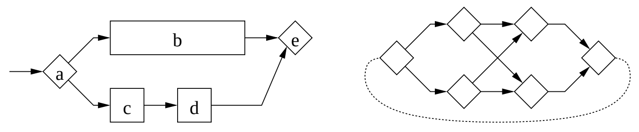
In contrast to structured fragments, the layout for unstructured fragments cannot solely rely on structural properties. To cater for the lack of structure, the following steps are performed:

1. Identify Back Edges
2. Perform Topology Sort
3. Determine Longest Path Spanning Tree
4. Order Edges
5. Compute Preliminary Layout
6. Compact Layout

S1. Identify Back Edges. Back edges have to be identified as they are exempted from fulfilling constraint C1 (i.e., edges should be drawn in the direction of the process's flow). This is accomplished using depth-first search.

S2. Perform Topology Sort. In the second stage, the topology order of the graph (without considering back edges) is computed to ensure that nodes depending on other nodes are drawn on the right hand side of the prerequisite nodes [19]. This ensures that nodes are ordered based on their dependencies. This is a prerequisite for fulfilling constraint C1 (i.e., edges should be drawn in the direction of the process's flow).

S3. Determine Longest Path Spanning Tree. A spanning tree, representing the longest path to each individual node is computed. This is necessary for identifying the parts of the graph that might be drawn in parallel to each other and for determining the leftmost position a node might occupy. Using the topology sort alone would be insufficient because it would simply return the ordering a,



(a) Longest Path Computation (b) Artificial Edge Added to Each Fragment

Fig. 6. Laying Out Unstructured Fragments

b, *c*, *d*, *e* without indicating which nodes may be drawn in parallel of each other (cf. Fig. 6(a)).

The longest path spanning tree allows us to identify that node *e* has to be placed after node *b* and not after node *d*, despite the fact that node *d* is more nodes away from the entry node *a*. The longest path is measured by taking the width of a node and the gap inserted between the node and its successor node. It is based on the topology sort. This step ensures that constraint C1 (i.e., edges should be drawn in the direction of the process's flow) is satisfied.

S4. Order Edges. To fulfill constraint C3 (i.e., edge crossings should be minimized), a planar order for the edges of the graph is computed using a left-right planarity checker [20]. This checker computes a spanning tree of the business process's underlying undirected graph and partitions the remaining edges into left and right partitions such that they do not generate conflicts (i.e., edge-crossings).

Before passing a fragment to the planarity checker, it is pre-processed such that edges are no longer permitted to be drawn around the entire process (cf. Fig. 1(a)). We accomplish this by adding an artificial edge from a fragment's entry node to the fragment's exit node (cf. Fig. 6(b)) which prohibits an edge on either the upper or the lower side of a fragment to be drawn around the entry or exit nodes since it would need to cross the artificial edge.

The planarity checker returns a planar order for edges indicating how to draw them in order to avoid edge crossing. In this stage all edges (i.e., including back edges that were omitted from the previous two phases) are considered. The planar order will be used in subsequent stages. If a graph is non-planar we perform some extra processing to find an optimal edge order. This modification will be discussed in Section 4.3.

S5. Compute Preliminary Layout. This stage computes a preliminary tree layout on the basis of the longest path spanning tree computed earlier. Each branch is put into its own horizontal section such that the branches are not overlapping each other (cf. Fig. 7). While generating the preliminary tree layout, this stage computes how much closer a given branch might be moved to the branch drawn directly above it, ensuring constraint C6 (i.e., minimality should be applied).

For computing the preliminary layout back edges are treated like cross and forward edges. All these non-tree edges are considered to be branches of zero height that run up to the point where they merge with another branch. For

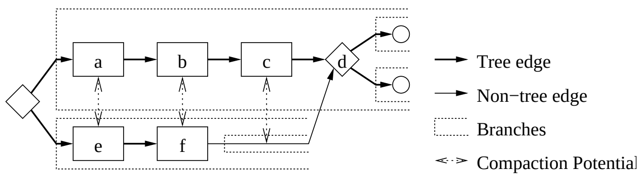


Fig. 7. Layout of the Branches of the Spanning Tree

instance, the branch with e and f in Fig. 7, extends up to the converging gateway d due to the cross edge.

Edges are laid out with the relaxed Manhattan layout (i.e., constraint C5). Tree edges are laid out with at most one bendpoint. All other edges have a maximum of two bendpoints ensuring C4 (i.e., bendpoints of edges should be minimized). Depending on the modeler's personal aesthetic preferences, back edges may be laid out with more than two bendpoints.

S6. Compact Layout In the final stage, the tree is compacted to ensure constraint C6 (i.e., minimality should be applied), by using the information calculated in Step S5 (i.e., compute preliminary layout), stating how much closer two branches of the tree might be drawn. In Fig. 7, this stage would shift nodes e and f closer to nodes a to c since the lower branch ends before the upper branch requires more space to fit in the final nodes.

4.3 Non-Planar Unstructured Fragments

Non-planar unstructured fragments are laid out similarly to their planar counterparts, except that stage S4 (i.e, order edges) adds a step to minimize edge crossings once a non-planar graph is identified. Since the general problem of minimizing edge crossings is known to be NP complete [21], we use a heuristic that is based on the planarity checker. When the planarity checker identifies a conflict, we ignore it and continue to sort edges eliminating crossings of the remaining edges.

To verify that this heuristic has a positive effect on the number of edge crossings, we compared it to a random order of the edges using a collection of insurance processes. For this process model collection even the random edge ordering performed well, since unstructured fragments in business processes are relatively rare and typically rather small. This finding has also been observed in [22]. Still, our heuristic manages to remove many of the unnecessary edge crossings present when using a random edge order.

Due to the positive results we have received so far with our simple heuristic, we have not invested more efforts in further improving it. Possible extensions would be to identify the $K_{3,3}$ and K_5 subgraphs (cf. Fig. 8), which are responsible for the non-planarity. The planarity tester can be extended to compute these in linear time and to minimize edge crossings introduced by these subgraphs. Another approach would be to use the heuristics presented in [23], which also produce attractive results [10].

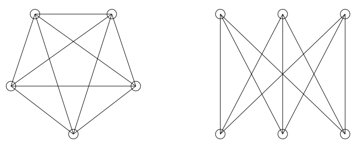


Fig. 8. K_5 and $K_{3,3}$ graphs

5 Evaluation

This section discusses in how far utilizing the layout algorithm provides additional benefits for process modelers beyond their natural way of laying out process models. Moreover, it presents an evaluation of the proposed algorithm in terms of its runtime complexity.

5.1 Constraint Evaluation

In order to evaluate whether the layout algorithm provides benefits for process modelers beyond their natural way of modeling, we conducted an empirical evaluation. In the following we describe the setup, execution and results.

Subjects. The subjects used for this evaluation have to be at least moderately familiar with imperative process modeling languages, preferably with BPMN. Otherwise, the effects would be blurred by subjects struggling with the modeling notation itself.

Objects. The object of our study is a process modeling task, starting from an empty canvas using a subset of BPMN. Subjects received a textual description of the process model[1] to be created, designed to reach a medium level of complexity going well beyond a "toy-example". To ensure that the process model is of appropriate complexity and not misleading, the textual descriptions were refined in several iterations. Additionally, we performed a pre-test involving ten students with a similar background. The gathered feedback allowed us to further refine the modeling task and to improve comprehension of the task at hand.

Instrumentation and Data Collection. To ensure reliable data collection, Cheetah Experimental Platform (CEP) [24] was utilized. In particular, we employed CEP's BPMN modeling editor to provide subjects with a modeling environment that allows them to focus on the task of modeling without being distracted by an abundance of software features. The built-in modeling tutorial informed the subjects on how the modeling editor could be used. Furthermore, CEP's logging mechanism ensured that no data was lost during the evaluation.

[1] Material available at http://pinggera.info/experiment/layout

Execution. The evaluation was conducted in January 2011 at Eindhoven University of Technology as part of a course on business process management. 50 students participated in the evaluation, creating 50 business process models. The evaluation was guided by CEP's experimental workflow engine [24], leading students through an initial questionnaire collecting demographic data, the modeling task, a concluding survey, and a feedback questionnaire. Prior to the evaluation, the students were not informed about the layout constraints that were evaluated.

Data Validation. The utilization of CEP minimizes the danger of misunderstandings and students accidentally disobeying the setup by guiding them through the evaluation [24]. Additionally, all data was automatically transferred and stored on a central database server preventing potential data loss. To ensure that the subjects match the targeted profile, i.e., modelers who are at least moderately familiar with BPMN, we used a questionnaire to assess familiarity with BPMN, confidence in understanding BPMN, and competence in using BPMN. To quantify these factors, we used a seven-point Likert-Scale, ranging from *strongly disagree* (1) over *neutral* (4) to *strongly agree* (7). The subjects have been moderately familiar with BPMN (mean value: 4.28), moderately confident in understanding BPMN (mean value: 4.72) and consider themselves moderately competent in using BPMN (mean value: 4.26). Thus, we conclude that the subjects are reasonable representatives for average process modelers, i.e., the targeted profile.

Data Analysis and Results. The resulting process models were analyzed for their adherence to the constraints described in Section 3 and coded on a binary scale, i.e., *adherence to constraint*, *no adherence to constraint.* In order to ensure coding reliability, the process models were evaluated by two researchers independently, iteratively discussing and revising their results until consensus was reached, thereby following the inter-rater procedure applied in [25].

Table 1 illustrates the adherence of the various process modelers to the previously described constraints. Especially constraint C3 (i.e., edge crossings should be minimized), constraint C4 (i.e., bendpoints of edges should be minimized), and constraint C6 (i.e., minimality should be applied), seemed to be already widely fulfilled by the subjects; at least 80% followed these principles.

Adoption rates for the remaining constraints, however, were considerably lower. Constraint C1 (i.e., edges should be drawn in the direction of the process's flow), and constraint C2 (i.e., incoming and outgoing edges must be separated), were only adopted by about 2/3 of the subjects, i.e., 68% and 66% respectively. The non-compliance with C2 was mostly caused by a lack of separation when modeling the loop contained in the process model; i.e., incoming and outgoing edges are attached to the same side of the activity or gateway making it relatively difficult to grasp incoming and outgoing edges. In addition, C2 was frequently violated when subjects changed the flow of the process model, i.e., one or more edges running opposite to the process model's flow.

Interestingly, only 16% of the students fulfilled constraint C5 (i.e., a Manhattan layout of edges must be used). The amount of work necessary for laying

Constraint	Adherence	Adherence %
C1: Edges drawn in the direction of the process's flow	34	68%
C2: Incoming and outgoing edges must be separated	33	66%
C3: Edge crossings should be minimized	46	92%
C4: Bendpoints of edges should be minimized	40	80%
C5: A Manhattan layout of edges must be used	8	16%
C6: Minimality should be applied	42	84%

Table 1. Constraint Evaluation

out every edge manually to comply with the manhattan layout seemed to outweigh its perceived benefit, which is also indicated by 14 process models partially fulfilling C5, e.g., having one or more rectangular edges.

The results of our evaluation show that process modelers who are moderately familiar with BPMN adhered *only to a limited degree* to the constraints underlying the proposed algorithm. Hence, the value of using automated layout is not only found in relieving the modeler of a part of her task, but also supporting her towards the *full* adoption of the layout constraints. As we explained, satisfying these constraints can be expected to foster a better understanding of a process model.

Limitations. Our findings are presented with the explicit acknowledgement of a number of limitations. First of all, our subjects represented a rather homogeneous group. More experienced process modelers may have laid out the process models differently, fulfilling other and, most likely, more constraints than these novice modelers did.

Secondly, all students were working on the same process model that might favor the satisfaction of some constraints. We tried to compensate for this problem by including commonly used workflow patterns [26] (e.g., sequence, parallel split, synchronization, exclusive choice, simple merge and structured loop) when designing the process modeling task.

5.2 Complexity Analysis

The proposed layout algorithm is intended to be used during modeling—the modeler should be empowered to automatically layout the process model anytime. To ensure that the invocation of the automated layout is not perceived as a nuisance, the layout's computation *must* be performed efficiently. In order to validate that our algorithm fulfills this requirement, we perform a complexity analysis and describe the algorithm's runtime behavior using *O-Notation* [27].

Table 2 summarizes the results of the complexity analysis where n is the sum of number of vertices and edges and f_i being the number of vertices and edges of the fragment being processed. Phase 1 (pre-processing) runs in $O(n)$: the preprocessing steps include the introduction of auxiliary vertices and reconnecting edges. This requires to iterate once over all nodes, hence this step runs in $O(n)$. For the complexity analysis of Phase 2 (SESE decomposition), we refer to [7] (again, $O(n)$).

Phase	Runtime
Phase 1: Pre-Processing	$O(n)$
Phase 2: SESE Decomposition	$O(n)$
Phase 3: Computation of Layout	$O(n)$
Structured Fragments	$O(f_i)$
Unstructured Fragments	$O(f_i)$
Non-planar Unstructured Fragments	$O(f_i)$
Total	$O(n)$

Table 2. Complexity Analysis

For the analysis of Phase 3 (layout computation), the complexity of laying out the different types of fragments has to be considered. Laying out structured fragments depends linearly on the number of vertices, as the main task is to position vertices horizontally (sequence and loop fragment) or vertically (branching fragment). Making branching fragments more compact can be compared to compacting unstructured fragments, which will be shown to run in $O(n)$.

Laying out unstructured fragments includes several sub-tasks, as being discussed subsequently. The depth-first search algorithm used for identifying back edges, the topology sort algorithm and the longest path search algorithm in acyclic graphs are all well known to run in $O(n)$ [28]. Edge ordering is performed using the algorithm presented in [20], shown to run in $O(n)$. Additionally, a spanning tree and the spaces between the elements of neighboring branches are computed. For computing the runtime complexity of the compaction algorithm, let us represent the space between neighboring elements as a graph such as the double-headed arrows in Fig. 7. These arrows form a planar graph and a planar graph is know to have a maximum of $3v - 6$ edges, where v is the number of vertices for graphs with three or more vertices [20]. During the compacting phase we iterate over the graph twice, assessing how much each element may be moved (once to compute the minimum for moving the entire branch, and once for the actual compaction). Hence, this part of the algorithm runs in $O(6 * n) = O(n)$ time.

The algorithm we use to minimize edge crossings does not change the algorithmic behavior of the planarity tester and hence also runs in $O(n)$ [20].

The computation of the entire layout is the sum of the computation of each individual fragment. Since $\sum_i^{fragments} f_i = n$, Phase 3 runs in $O(n)$, resulting in an overall runtime complexity for computing the layout of $O(n)$, i.e., in linear time.

6 Conclusions

In this work we presented an automated layout algorithm that is specifically tailored to the characteristics of business process models. In particular, we improved over state-of-the-art generic layout algorithms by following a three-phase approach to pre-process, structure and layout business process models. The proposed algorithm makes use of the Refined Process Structure Tree for partitioning the model into SESE fragments that can be laid out using a bottom-up approach. In addition to the conception and implementation of the algorithm, a complexity analysis showed that the algorithm runs in $O(n)$, i.e., linear time, which ensures that its application is also feasible in real-world modeling settings. Furthermore, we performed an empirical evaluation of the constraints underlying the algorithm. We established that not all constraints are widely followed by modelers with moderate modeling experience. This supports the view that our algorithm can be considered as providing support beyond what modelers intuitively or deliberately take care of with respect to the layout of a process model. In this way, the proposed algorithm may contribute to the creation of process models that are more easily read.

Additional potential benefits that have not been discussed so far are that an automatic layout can be utilized for computer-generated models and will also allow for a consistent layout across models, i.e., all constraints are fully supported. This may be particularly beneficial in the context of large process modeling initiatives where several process modelers create process models, which are then read by many different stakeholders. Establishing the existence and size of such a benefit will be a topic for further research.

With respect to the technical side of our work, there are open issues as well. The current service[2], has a few shortcomings compared to the algorithm presented in this paper. It does not compact branching fragments, back edges are not laid out as discussed in the paper, and the compacting phase may introduce additional bendpoints. We plan to address these issues in future versions of the service, as well as to extend it towards dealing with data items and swim lanes.

References

1. Dehnert, J., Van Der Aalst, W.: Bridging the gap between business models and workflow specifications. IJCIS **13** (2004) 289–332
2. Schrepfer, M., Wolf, J., Mendling, J., Reijers, H.A.: The impact of secondary notation on process model understanding. In: Proc. PoEM '09. (2009) 161–175
3. Reijers, H.A., Mendling, J.: A Study into the Factors that Influence the Understandability of Business Process Models. IEEE Transactions on Systems, Man and Cybernetics, Part A (2010) 1–14
4. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. Comm. of the ACM **38** (1995) 33–44
5. Purchase, H.C.: Which aesthetic has the greatest effect on human understanding? In: Proc. Graph Drawing' 97. (1997) 248–261
6. Purchase, H., Carrington, D., Allder, J.: Empirical evaluation of aesthetics-based graph layout. Empirical Software Engineering **7** (2002) 233–255
7. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. DKE **68** (2009) 793–818
8. Koschmider, A., Song, M., Reijers, H.A.: Social software for business process modeling. JIT **25** (2010) 308–322
9. Kitzmann, I., König, C., Lübke, D., Singer, L.: A simple algorithm for automatic layout of bpmn processes. In: Proc. CEC '09. (2009) 391–398

[2] Freely available through http://business-services.researchlabs.ibm.com.

10. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.P.: A technique for drawing directed graphs. IEEE Trans. Softw. Eng. **19** (1993) 214–230
11. Battista, G.D., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theoretical Computer Science **61** (1988) 175–198
12. Hutton, M.D., Lubiw, A.: Upward planar drawing of single source acyclic digraphs. In: Proc. SODA '91. (1991) 203–211
13. Bertolazzi, P., Battista, G.D., Mannino, C., Tamassia, R.: Optimal upward planarity testing of single-source digraphs. SIAM J. Comput. **27** (1998) 132–169
14. Hasan, M., Rahman, M.S., Nishizeki, T.: A linear algorithm for compact box-drawings of trees. Networks **42** (2003) 160–164
15. Diguglielmo, G., Durocher, E., Kaplan, P., Sander, G., Vasiliu, A.: Graph layout for workflow applications with ilog jviews. In: Proc. Graph Drawing '02. (2002) 257–282
16. yWorks: yFiles for Java (accessed 03/14/2011) http://www.yworks.com/en/products_yfiles_about.html.
17. Stein, B., Benteler, F.: On the Generalized Box-Drawing of Trees: Survey and New Technology. In: Proc. I-KNOW '07. (2007) 408–415
18. IBM: Websphere business modeler (accessed 03/14/2011) http://www.ibm.com/software/integration/wbimodeler.
19. Kahn, A.B.: Topological sorting of large networks. Comm. of the ACM **5** (1962) 558–562
20. Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM **21** (1974) 549–568
21. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM J. on Algebraic and Discrete Methods **4** (1983) 312–316
22. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In: Proc. ICSOC '07. (2007) 43–55
23. Warfield, J.: Crossing theory and hierarchy mapping. IEEE Transactions on Systems, Man, and Cybernetics **7** (1977) 505–523
24. Pinggera, J., Zugal, S., Weber, B.: Investigating the process of process modeling with cheetah experimental platform. In: Proc. ER-POIS '10. (2010) 13–18
25. Recker, J., Safrudin, N., Rosemann, M.: How Novices Model Business Processes. In: Proc. BPM '10. (2010) 29–44
26. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases **14** (2003) 5–51
27. de Bruijn, N.G.: Asymptotic Methods in Analysis. Dover Publications (1958)
28. Diestel, R.: Graph Theory. 4th edn. Springer (2010)