BUSINESS PROCESS VISUALIZATION – USE CASES, CHALLENGES, SOLUTIONS*

Stefanie Rinderle[1], Ralph Bobrik[1], Manfred Reichert[2] and Thomas Bauer[3]
[1]*Dept. DBIS, University of Ulm, Germany, {rinderle, bobrik}@informatik.uni-ulm.de*
[2]*Information Systems Group, University of Twente, The Netherlands, m.u.reichert@cs.utwente.nl*
[3]*DaimlerChrysler Research Center Ulm, Germany, thomas.tb.bauer@daimlerchrysler.com*

Keywords: Business Processes Visualization.

Abstract: The proper visualization and monitoring of their (ongoing) business processes is crucial for any enterprise. Thus a broad spectrum of processes has to be visualized ranging from simple, short–running processes to complex long–running ones (consisting of up to hundreds of activities). In any case, users shall be able to quickly understand the logic behind a process and to get a quick overview of related tasks. One practical problem arises when different fragments of a business process are scattered over several systems where they are often modeled using different process meta models (e.g., High–Level Petri Nets). The challenge is to find an integrated and user–friendly visualization for these business processes. In this paper we discover use cases relevant in this context. Since existing graph layout approaches have focused on general graph drawing so far we further develop a specific approach for layouting business process graphs. The work presented in this paper is embedded within a larger project (Proviado) on the visualization of automotive processes.

1 INTRODUCTION

The proper visualization and monitoring of their (ongoing) business processes is crucial for any enterprise. Thus a broad spectrum of processes has to be visualized ranging from simple, short-running workflows to complex long-running processes (consisting of hundreds of activities). In the automotive sector, for example, this includes e-procurement and change management processes whereas the latter may be long-running car engineering or release management processes. In any case, users shall be able to quickly understand the logic behind a process and to get a quick overview of their tasks. In practice, business process data are often scattered over several application systems; i.e., a business process is splitted into different more or less explicit fragments which are kept and executed within different systems (Bobrik et al., 2005). One consequence is that the definition and control of these fragments are often based on different *process meta models*[1]. One first important challenge arising from the visualization of business processes is to extract data about process fragments from the different source systems and to provide an integrated model of the overall process. Further if business process data is completely available the challenge is to find a user-friendly layout of that process. Business processes are often very complex as we have learned from case studies in the automotive domain (Bobrik et al., 2005). A typical process consists of dozens up to hundreds of activities and comprises numerous additional information (e.g., about process data elements, actors, and resources, cf. Fig. 2).

So far a lot of layout approaches for all kinds of graphs (e.g., trees, DAGs, planar graphs, etc.) have been presented in the literature (Eades et al., 1993; Sugiyama, 2002). There are few approaches dealing with the layout of business process graphs as well (Fleischer and Hirsch, 2001; Kikusts and Rucevskis, 1995; Six and Tollis, 2002; Wittenburg and Weitzman, 1996a; Wittenburg and Weitzman, 1996b; Yang et al., 2004). However, to our best knowledge, most of them do not exploit the semantics of business process graphs and only deal with graphs consisting of untyped nodes and edges, i.e., graph nodes (edges) cannot be distinguished. Fig. 1 depicts an example

*This work was conducted during a post doc stay at the University of Twente which was fundend and supported by DaimlerChrysler Research Ulm, Germany.

[1]A meta model defines the constructs available for modeling the process. Examples include Petri Nets, Activity Diagrams, and Statecharts.

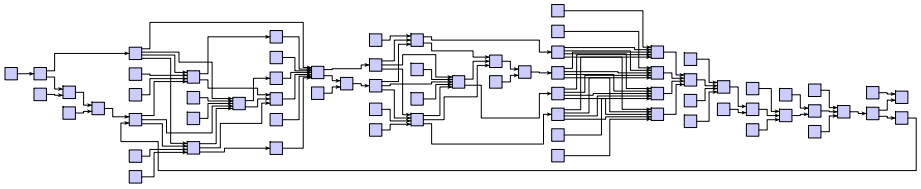

Figure 1: Change Management Process with Untyped Nodes and Edges.

for this case.

However, untyped nodes and edges do not reflect the "nature" of business process graphs. Usually, respective process graphs comprise nodes with (partially) different semantics. This includes, for example, nodes representing activities (i.e., process steps) and nodes representing process data elements. Very similar, edges of different type and semantics (e.g., control flow and data flow edges) have to be distinguished. Consider the process depicted in Fig. 2: Activity nodes are represented as rectangles whereas data element nodes are depicted as trapezoids. This graphical distinction reflects the different roles these elements possess with respect to the overall business process. The challenge is to exploit such semantic information in order to build up an adequate layout for business process graphs.

However, drawing business process graphs is not a one-time task. In fact, the layout of business process graphs is highly dynamic. As an example consider the dynamic generation of business process views. Such views on process graphs may vary from user to user (Bobrik et al., 2005). Furthermore, especially long-running business processes frequently have to be changed due to several reasons (e.g., to adapt to new laws or process optimizations) (Rinderle et al., 2004; Reichert and Dadam, 1998). As a consequence the process graph layouts have to be adapted accordingly. In this context we need adequate approaches for relayouting business process graphs after changes. This imposes several challenges including the provision of automatic and efficient algorithms for maintaining the user's "mental map" when a process change is performed.

In this paper we summarize and describe several use cases. We start with the layouting of business processes graphs. Then we focus on relayouting business process graphs after dynamic changes. Thereby one goal is to maintain the user's mental map of the process. After this, we consider the visualization and layouting of process instances (i.e., concrete business cases created from a business process model). Finally, we shortly discuss how to display organizational structures of enterprises and end up with the definition of certain *views* on business processes (e.g., only displaying the steps worked on by people of a certain group).

Our approach exploits knowledge about the semantics of graph nodes and edges in order to find an adequate process layout. We proceed in different steps and allow users to specify which process constructs shall be prioritized most when layouting a process graph. As we know from our case studies, in most cases, users want to start with layouting the control flow (i.e., the work tasks and the order in which they are carried out). Therefore, we illustrate our approach for layouting *control flow skeletons*. In general, however, starting with the layouting of other process perspectives (e.g., data flow) is conceivable as well. For the control flow layouts we impose several esthetic criteria (e.g., mimizing the number of edge crossings (Purchase, 2002)) which we meet by applying a method based on preprocessing and permutation. In order to complete the process graph layout additional steps are discussed which show how to enhance the control flow skeleton layout with the other process elements (e.g., process data elements or actors nodes).

In Section 2 we present use cases for visualizing business process graphs. Related work is discussed in Section 3. Section 4 presents our approach for layouting business process graphs. We close with a summary and an outlook.

2 USE CASES

In order to come to a sophisticated approach for business process graph visualization we first have to consider several use cases.

2.1 Business Process Graphs

The basis for business process visualization is to find an adequate approach for layouting process graphs. As mentioned we may be confronted with process fragments scattered over different information systems and being based on different process meta models. Actually an integrated and understandable visualization of the whole business process is desired. In order to achieve this we have to extract process knowl-

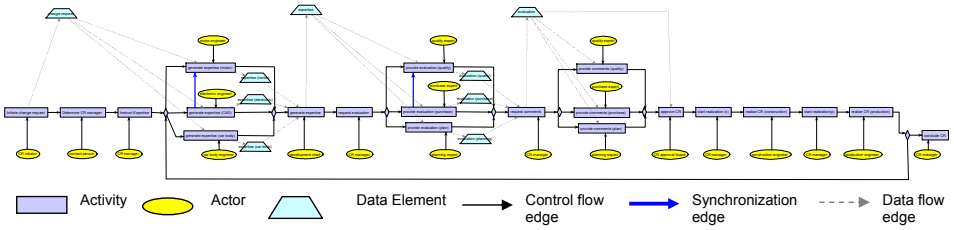

Figure 2: Change Management Process with Typed Nodes and Edges (Simplified).

edge from different logfiles and application systems and must then transfer the obtained process fragments into the notion of a canonical process meta model. Doing so one has to preserve potentially existing process meta model properties (e.g., information about the structuring of related process models) and existing layout information. This step is then followed by layouting the whole business process using the graphical notion of the canonical meta model. Thereby the challenges are to (1) find an algorithm for layouting business processes graphs, (2) exploit the particular semantics of the different process elements, and (3) use available meta model properties and already existing layout information in order to optimize the process layout algorithm

2.2 Dynamic Changes

Business processes change over time (e.g., by adding/deleting process steps or dependencies between them) (Rinderle et al., 2004; Reichert and Dadam, 1998). There are several approaches supporting such changes (Rinderle et al., 2004). When changing the structure (or logic) of a business process it is important that this is also reflected by adapting the layout of the business process graph accordingly. The challenge is to avoid that users lose their *mental map* when changing the process. Note that this could be the case if we relayout the process "from scratch" as depicted in Fig. 3.

The problem of maintaining the mental map when conducting changes of the process logic has been recognized in the literature and different algorithms have been presented in this context (e.g., Force-Scan or incremental algorithms (Yang et al., 2004; Diguglielmo et al., 2002)). However, all these approaches have been applied to graphs with untyped nodes so far. For this reason, it is also very interesting to analyze their applicability on business processes.

2.3 Business Process Instances

Based on a process model, new process instances can be created and executed during runtime. Since process instances represent concrete executions of the process model, the latter have to be enriched with state information (e.g., node and edge markings) in order to reflect the respective instances. The challenge is to display business process models together with additional information (e.g., state markings or instance-specific changes). An orthogonal aspect is the way of visualizing process instances. They can be displayed in a static way (displaying the instance and its current state) or by using a dynamic layout (i.e., by replaying the previous execution history of respective instances along a time line).

2.4 Business Process Views

In practice process graphs are often very big and complex ("wallpapers"). Thus users are overwhelmed with information. Therefore a challenge for the visualization of business process graphs is to be able to (dynamically) build up (dynamic) views on business process graphs, i.e., to choose process objects along certain criteria and to compose them in an appropriate way. Criteria based on which process views can be built may be (1) *object types* (we only display objects of a certain type, e.g., only nodes of type "activity" are displayed whereas nodes of types "data element" or "actor" are hidden), (2) *static attribute values* (e.g., only manual activities are displayed whereas automatic activities are hidden), and (3) *dynamic attribute values* (e.g., displaying only those activities which have not been worked on so far).

This technique is called *graph reduction* (Sadiq and Orlowska, 2000). Another approach is *graph aggregation* (Liu and Shen, 2003). Aggregation means to nest certain objects (e.g., activities) to a complex object (e.g., activity with underlying subprocess) and to adapt activity attributes accordingly.

In order to provide a suitable framework for business process layout it is extremely important to deeply understand all these use cases and to provide appropriate approaches. In this paper we focus on the first use case (i.e., business process layout) to build up the basis for the other use cases.

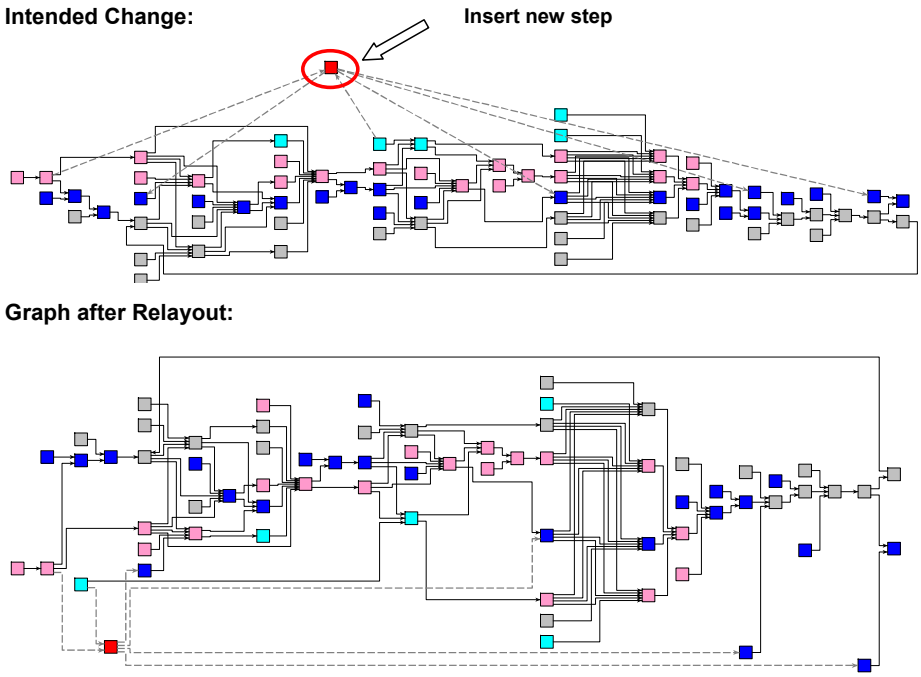

Figure 3: Overall Approach for Layouting Business Processes.

3 RELATED WORK

There are a lot of approaches dealing with graph layout. In general, graph classes having certain properties are identified and layout algorithms based on these properties are provided. These graph classes and the respective algorithms comprise trees (Eades et al., 1993), directed (acyclic) graphs (Sugiyama, 2002), planar graphs (de Fraysseix et al., 1988), and series-parallel graphs (Hong et al., 1998). In the literature there are also approaches dealing with general graphs, e.g., Heavy Duty Preprocessing, Spring Embedder (Fruchterman and Reingold, 1991; Frick et al., 1994), etc., There are also a few approaches dealing with layouting process graphs (Six and Tollis, 2002; Yang et al., 2004; Diguglielmo et al., 2002). In (Six and Tollis, 2002) an approach is introduced which determines a layout of the process in linear time using existing partitioning information (e.g., swimlanes). Yang et al (Yang et al., 2004) address several problems described in connection with the above use cases. In detail, they propose the so called force scan algorithm which maintains the mental map after changes. Furthermore the authors suggest to use the fisheye technique in order to overcome the “wallpaper” problem. (Diguglielmo et al., 2002) show how their tool jViews contributes to layout process graphs. Using the incremental mode the mental map of process graphs is maintained after changes. It is also possible to impose a partitioning on the graphs (e.g., swimlanes). An approach using a 3D representation of business processes including process analyis results (e.g., throughput) is introduced in (Schönhage et al., 2000).

Though all of these approaches are very inspiring they neglect the different semantics of the nodes and edges within a business process. Therefore we will make use of existing ideas and theoretical results but combine and extend them towards an approach especially tailored for the layout of business process graphs. Doing so might open a new interesting application area for general graph drawing approaches.

4 BUSINESS PROCESS LAYOUT

In this section we present our approach for layouting business process graphs which has been implemented in the context the Proviado project on business process visualization[2]. First of all, we introduce a (canonical) process meta model describing the different constructs which can be used for modeling business processes. In order to provide a complete formal basis for our further considerations we simplify the meta model to some extend.

[2]The partners are DaimlerChrysler, University of Ulm, and University of Twente.

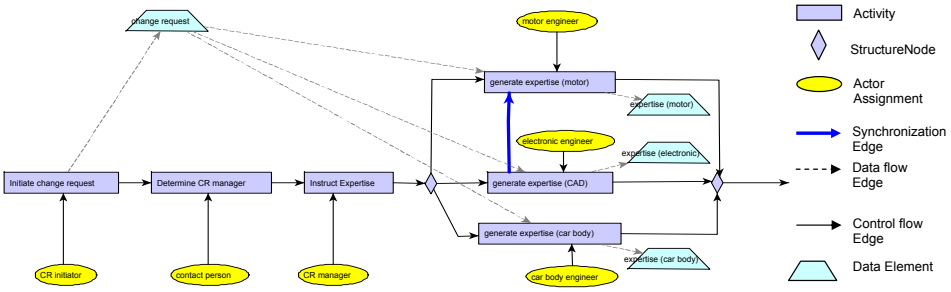

Figure 4: Overall Approach for Layouting Business Processes.

4.1 Fundamentals

Within the canonical process meta model CPM we specify $\mathcal{A}$ as the total set of all process activities, $\mathcal{D}$ as the total set of process data elements, and $\mathcal{W}$ as the total set of all actors involved in the execution of any process model. Based on modeling elements offered by CPM new *process models* can be defined (e.g., order procurement process in an enterprise or treatment processes in a hospital).

Definition 1 (Process Model) *A tuple PM = (A, D, W, AT, CtrlE, CT, DataE, WorkE) is called a process model with*

- *$A \subset \mathcal{A}$ is the set of activities, $D \subset \mathcal{D}$ is the set of data elements, and $W \subset \mathcal{W}$ is the set of actors involved in the execution of instances based on PM*
- *AT denotes the function which assigns to each activity from A a particular type, i.e., AT: $A \mapsto$ {`Activity, StructureNode, Start, End`}; thereby structure nodes are used for structuring purposes (e.g., as split or join nodes).*
- *CtrlE $\subset A \times A$ denotes the set of control edges in PM. A control edge $a \rightarrow b$ denotes that activity a must be completed before activity b can start.*
- *CT denotes the function which maps control edges from CtrlE onto their particular type, i.e., CT: CtrlE $\mapsto$ {`Control, Sync, Loop`}*
- *DataE $\subset (A \times D) \cup (D \times A)$ denotes the set of data edges in PM; thereby $D \times A$ ($A \times D$) describes a read (write) access*
- *WorkE $\subset W \times A$ denotes the set of actor edges in PM; a actor edge $w \rightarrow a$ denotes that activity a is worked on by actor w.*

An activity $a \in A$ denotes a particular work task within a process model PM, e.g., `Instruct Expertise` (cf. Fig. 4). The direct successor of this activity has activity type `StructureNode` (i.e., it is not associated with a specific work task). Since this node has several outgoing control edges $e_1, ..., e_n$ with CT(e_i) = `Control` (i = 1, ..., n) it acts as a split node of an alternate or parallel branching. Within an alternate branching one branch is selected for execution during runtime (e.g., based on process data) whereas for parallel branchings all branches are executed concurrently. Control edges describe the execution order between activities. They can be further distinguished into basic control edges, synchronization edges, and loop edges. Synchronization edges determine the order of activities within different branches of a parallel branching. Cyclic process structures can be described by using loop backward edges. Besides these control flow constructs a process model contains additional elements, e.g., data elements (e.g., `change request` in Fig. 4) and data edges. Read (write) data edges describe which data elements are read (written) by which activity. In Fig. 4, for example, data element `change request` is written by activity `change request` and read by activity `generate expertise (CAD)`. Finally, we describe which activity is worked on by which actors by using actor assignments `WorkE` (e.g., actor `car body engineer` works on activity `Instruct Expertise`).

A process model can be (graphically) represented as a process graph for which we want to find a user-friendly and process-adequate layout in the following. The idea is to start with layouting a certain projection of the process consisting of *core* nodes and edges. This layout is then stepwisely enhanced with the remaining *satellite* nodes and edges. In order to keep the layout configurable we allow the user to specify the sets of core and satellite objects.

Definition 2 (Core and Satellite Objects) *Let PM = (A, D, W, AT, CtrlE, CT, DataE, WorkE) be a process model. Let V := (A $\cup$ D $\cup$ W) and E := (CtrlE $\cup$ DataE $\cup$ WorkE). Then PG = (V, E) denotes the process graph. Based on PG the user can specify the set of core nodes CN by choosing one of the sets A, D, and W. Then the set of core edges CE and the set of satellite nodes (edges) SN (SE) can be derived accordingly (i.e., CN = A $\Longrightarrow$ CE = CtrlE, CN = D $\Longrightarrow$ CE = DataE, CN = W $\Longrightarrow$ CE = WorkE, SN:= V $\setminus$ CN, SE:= E $\setminus$ CE).*

To achieve *structurally correct* process models the associated process graph must obey certain correctness constraints. We define *projections* on process graphs which are used to introduce correctness constraints afterwards.

Definition 3 (Process Graph and Projections) *Let PM = (A, D, W, AT, CtrlE, CT, DataE, WorkE) be a process model and let PG = (V, E) be the associated process graph based on which we define different projections:*

- *PG_{CF} := (V_{CF}, E_{CF}) with V_{CF} = A and E_{CF} = {e ∈ CtrlE | CT(e) =* `Control`*} denotes the projection on activities and control edges of type* `Control`*.*
- *PG_{Sync} := (V_{CF}, E_{Sync}) with E_{Sync}:= E_{CF} ∪ {e ∈ CtrlE | CT(e) =* `Sync`*} denotes the projection on PG_{CF} plus sync edges.*
- *PG_{Loop} := (V_{CF}, E_{Loop}) with E_{Loop}:= E_{Sync} ∪ {e ∈ CtrlE | CT(e) =* `Loop`*} denotes the projection on PG_{Sync} plus loop edges. We denote PG_{Loop} as control flow skeleton.*

A process graph must adhere the following constraints in order to represent a structurally correct process model (e.g., avoiding deadlock causing cycles).

Definition 4 (Correctness of a Process Graph) *Let PG = (V, E) be a process graph and PG_{CF}:= (V_{CF}, E_{CF}), PG_{Sync} := (V_{CF}, E_{Sync}), PG_{Loop}:= (V_{CF}, E_{Loop}) be the projections as defined in Def. 3. Then PG is a correct process graph iff the following constraints hold:*

1. **Unique Start and End Node:**
   $\dot{\exists}$ *s* ∈ V_{CF}: /∃ *e = (v', s) with CT(e)* ∈ {`Control`, `Sync`} ∈ *E, v'* ∈ V_{CF} ∧
   $\dot{\exists}$ *e* ∈ V_{CF}: /∃ *e = (e, v")* ∈ *E with CT(e)* ∈ {`Control`, `Sync`}, *v"* ∈ V_{CF} ∧ *s* ≠ *e*
2. **Connectivity:**
   PG_{CF} is connected ∧
   ∀*s* ∈ $V \setminus V_{CF}$: (∃$e = (s, v) \in E \vee \exists e = (v, s) \in E$), $v \in V_{CF}$
3. **Synchronization:** *Control edges between activities from different parallel branches are only of type* `Sync`*, formally:*
   ∀$e = (a_1, a_2) \in E$ *with* $a_1, a_2 \in V_{CF} \wedge a_1 \notin$ *(succ*(PG,* a_2*)* ∪ *pred*(PG,* a_2*)): CT(e) =* `Sync`
   where
   • *succ*(PG, n):= succ(PG, n)* ∨ *succ(PG, succ*(PG, n)) with*
   succ(PG, n):={n' ∈ V_{CF} | ∃ e = (n, n') ∈ E with CT(e) ∈ {`Control, Sync`}}
   • *pred*(PG, n):= pred(PG, n)* ∨ *pred(PG, pred*(PG, n)) with*
   with pred(PG, n):={n' ∈ V_{CF} | ∃ e = (n', n) ∈ E with CT(e) ∈ {`Control, Sync`}}
4. **Deadlockfree:** *PG_{Sync} is an acyclic graph, i.e., the use of control and sync edges must not lead to deadlock-causing cycles.*

After introducing the necessary fundamentals we now describe our approach for layouting business process graphs. Generally, users can configure which nodes and edges are considered as core and which are considered as satellite objects. This influences the resulting process graph layout. As we know from our practical studies in most cases users prioritize an adequate layout of the control flow skeleton (i.e., the work tasks themselves and the order in which they are to be carried out). For this we select the activities as the set of core nodes (i.e., $CN = A$). Thus the set of core edges contains the control edges (i.e., $CE = CtrlE$). Accordingly, the set of satellite nodes comprises data and actor nodes (i.e., $SN = D \cup W$) and the set of satellite edges contains data and actor edges (i.e., $SE = DataE \cup WorkE$). Due to lack of space in this paper we present our approach for focusing on the control flow first and enhancing it with data and actor elements in the following. Nevertheless, this approach can be transferred to other methodologies (e.g., starting with the data flow graph) as well.

4.2 On Layouting Process Graphs

As discussed above often an appropriate layout of the control flow skeleton is fundamental for the process graph layout. Therefore we start with layouting the control flow skeleton followed by the placement of satellite objects. Let $PG = (V, E)$ be a process graph and CN, CE, SN, SE be the set of core and satellite nodes (edges) as specified by the user (in the following: $CN = A$, $CE = CtrlE$, $SN = D \cup A$, $SE = DataE \cup WorkE$). We start with finding an adequate layout of the control flow skeleton PG_{Loop}. Adequate means (at least) to focus on the control flow and to minimize edge crossings (Purchase, 2002). Besides these two most important aspects other esthetic criteria exist (e.g., mimizing the layout size). Due to lack of space we omit further details here.

First of all, comparable to heavy duty preprocessing approaches, we determine PG_{CF} (cf. Def. 3). We can show that PG_{CF} (together with correctness constraints 1 – 4) constitutes a series-parallel graph, i.e., it can be constructed by serial and parallel construction. This construction is reflected by the *structure tree* (Hong et al., 1998). For example, Fig. 5 shows the structure tree for an abstract process. Since PG_{CF} is series-parallel (and therefore planar) it can be drawn without any edge crossings (e.g., using the Sugiyama algorithm with Barycenter crossing reduction and coordinate assignment using (Sugiyama, 2002; Brandes and Köpf, 2002)).

However, if we also consider synchronization edges (graph projection PG_{Sync}) edge crossings may occur (cf. Fig. 5), i.e., crossings between sync edges or crossings between sync and control edges of type `Control`. As it can be seen from Fig. 5 the number of (sync) edge crossings depends on the aligment of the associated parallel branches. Therefore our aim is to find an alignment of the parallel branches for which the number of (sync) edge crossings becomes

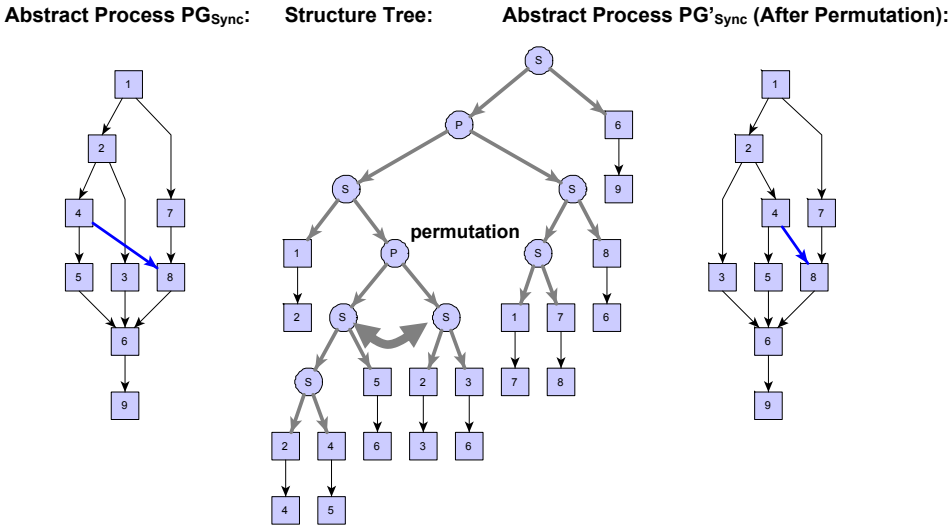

Figure 5: PG$_{CF}$ with Associated Structure Tree.

minimal. In the following we use the correlation between the order of branches in the structure tree and the alignment of the parallel branches in PG$_{CF}$. By permuting the order of the branches in the structure tree we obtain the different possible alignments of the parallel branches in PG$_{CF}$. Since we must not change the original order of working tasks we only allow to permute the order ot the siblings of parallel composition nodes (cf. Fig. 5).

Among all permutations the layout with minimal number of edge crossings can be found. In general, routing of loop edges can be handled analogously. However, the evaluation of the resulting layout with respect to the number of edge crossings becomes more complex since we are confronted with different types of edge crossings. Therefore we need a more sophisticated evaluation metrics minimizing the number cross$_{sync}$ of sync edge crossings plus the number cross$_{loop}$ of loop edge crossings where users can weight the numbers with priorization factors d_s for sync edges and d_l for loop edges (i.e., min($d_s * cross_{sync}$ + $d_l * cross_{loop}$)).

4.3 Satellite Objects

After determining the layout for PG$_{Loop}$ the control flow skeleton is enhanced with information about data flow and actor assignments. This remaining information is captured by the sets of satellite nodes and edges (i.e., $SN = D \cup W$ and SE = DataE $\cup$ WorkE). There are different possibilities for integrating the satellite objects into the existing control flow skeleton layout. We sketch them and describe which factors may influence the decision for one of these possibilities as well as their advantages and drawbacks. Basically, we distinguish between a *local alignment* and a *global alignment* of satellite objects. Local alignment means that the satellite objects belonging to a work task are aligned in the "surrounding" of the work task what may lead to duplication of satellite objects. Choosing global alignment each object is unique und connected to one ore more activities by the respective edges.

Local Alignment: If we choose local alignment the satellite objects associated with a certain activity are aligned "around" this activity. Then the activity together with its satellite objects can be seen as one (complex) activity. Inserting this complex activity into the control flow skeleton can be achieved by shifting the other activities in order to obtain the necessary space. This can be done, e.g., with the Force-Scan algorithm for maintaining the mental map after changes (Yang et al., 2004). The advantage of local alignment is that the number of edge crossings is not increased by the alignment of the satellite objects. A possible drawback is that users may loose the process overview or the correlation between the different duplicates of satellite objects is not visible.

Global Alignment: The first possibility is that users *manually* align satellite objects, i.e., they take the control flow skeleton layout and place the satellite objects together with the respective edges manually around the skeleton, Then satellite objects, e.g., data elements, are placed "around" the skeleton. Reasons for this approach may be that there are only few satellite objects or the user prefers a special alignment. Another approach is to treat the alignment of satellite objects as dynamic changes and to apply one of the algorithms proposed in the literature, e.g., the Force-

Scan algorithm (Yang et al., 2004).

One disadvantage of all global approaches is that the number of edge crossings is potentially increased. To overcome this limitation the insertion of the satellite edges (i.e., data flow or work assignment edges) could be already integrated in the permutation step introduced in Section 4.2.

5 SUMMARY AND OUTLOOK

We have discussed several use cases related to the visualization and layout of business process graphs which have been identified within the **Proviado** project (**pro**cess **vi**sualisation in the **a**utomotive **do**main) in cooperation with DaimlerChrysler Research Ulm. Furthermore, a layout approach which exploits the different semantics of the nodes and edges of a process graph has been introduced.

This approach can be improved by using already existing information (e.g., knowledge about process meta model properties or existing layout information) within the algorithm. In our approach the following meta model properties could be useful for a respective improvement: We start with layouting the series-parallel control flow skeleton of a business process. For certain process meta models like BPEL4WS or WSM Nets (Rinderle et al., 2004) it can be shown that they are *block-structured*, i.e., they are not only series-parallel but possess a nested structure (i.e., for each split node a unique join node can be found and vice versa). If we know that the business process was modeled in a block-structured way we can use this information in constructing the series-parallel (or block-structured) control flow skeleton. If we know that the process was modeled according to an acyclic process meta model, e.g., Activity Nets as used in IBM Websphere products, we can use this information to abstain from the last step of inserting the loop edges into the directed ayclic control flow skeleton.

The current implementation of our approach comprises a visualization component for process graphs based on the scalable vector graphic (svg) format. Furthermore we plan to integrate this component within our adaptive process management system ADEPT2. Based on this we can, for example, evaluate approaches for maintaining the mental map after process changes (Rinderle et al., 2004).

REFERENCES

Bobrik, R., Reichert, M., and Bauer, T. (2005). Requirements for the visualization of system-spanning business processes. In *DEXA'05*, pages 948–954.

Brandes, U. and Köpf, B. (2002). Fast and simple horizontal coordinate assignment. In *GD01*.

de Fraysseix, H., Pach, J., and Pollack, R. (1988). Small sets supporting fary embeddings of planar graphs. In *STOC'88*, pages 426–433.

Diguglielmo, G., Durocher, E., Kaplan, P., Sander, G., and Vasiliu, A. (2002). Graph layout for workflow applications with ILOG jViews. In *Proc. GD02*.

Eades, P., Lin, T., and Lin, X. (1993). Two tree drawing conventions. *Computional Geometry & Applications*, 3(2):133–153.

Fleischer, R. and Hirsch, C. (2001). Graph drawing and its applications. In *Grawing Graphs: Methods and Models*, pages 1–21.

Frick, A., Ludwig, A., and Mehldau, H. (1994). A fast adaptive layout algorithm for undirected graphs. In *GD94*.

Fruchterman, T. and Reingold, E. (1991). Graph drawing by force-directed placement. *Software Practice and Experience*, 21(11):1129–1164.

Hong, S., Eades, P., Quigley, A., and Lee, S. (1998). Drawing algorithms for series-parallel digraphs in two and three dimensions. In *Proc. GD98*, pages 198–209.

Kikusts, P. and Rucevskis, P. (1995). Layout algorithms of graph-like diagrams for GRADE windows graphic editors. In *Proc. GD95*, pages 361–364.

Liu, D. and Shen, M. (2003). Workflow modeling for virtual processes: An order–preserving process–view approach. *Information Systems*, 28(6):505–532.

Purchase, H. (2002). Metrics for graph drawing aesthetics. *Visual Languages and Computing*, 13:501–516.

Reichert, M. and Dadam, P. (1998). ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. *JIIS*, 10(2):93–129.

Rinderle, S., Reichert, M., and Dadam, P. (2004). Flexible support of team processes by adaptive workflow systems. *DPD*, 16(1):91–116.

Sadiq, W. and Orlowska, M. (2000). Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134.

Schönhage, B., van Ballegooij, A., and Elliens, A. (2000). 3D gadgets for business process visualization - a case study. In *Web3D – VRML 2000*, pages 131–138.

Six, J. and Tollis, I. (2002). Automated visualization of process diagrams. In *Proc. GD01*, pages 45–59.

Sugiyama, K. (2002). *Graph Drawing and Applications for Software and Knowledge Engineering. Series on Software Eng. & Knowledge Eng.* World Scientific.

Wittenburg, K. and Weitzman, L. (1996a). Process visualization in ShowBiz. In *Proc. GD96*.

Wittenburg, K. and Weitzman, L. (1996b). Relational grammars: Theory and practice in a visual language interface for process modeling. In *In Proc. Workshop on Theory of Visual Languages*, Gubbio, Italy.

Yang, Y., Lai, W., Shen, J., Huang, X., J.Yan, and Setiawa, L. (2004). Effective visualisation of workflow enactment. In *APWeb'04*, pages 794–803.