Making Process Model Versions Comparable by Quantifying Changes

Nico Herzberg, Mathias Weske

Making Process Model Versions Comparable by Quantifying Changes

Nico Herzberg, Mathias Weske

Hasso Plattner Institute at the University of Potsdam
{nico.herzberg, mathias.weske}@hpi.uni-potsdam.de

Abstract. A central task in the business process management life cycle is the evaluation of business processes to get a solid base for process improvements. Business processes are described as process models to explicitly state the operations needed to be carried out to reach the companies' business goals. These process models evolve over time because of changing market conditions, process improvements, or legal changes, etc. and result in multiple process versions. In such circumstances, process analysis requires the comparison of several process versions, the so-called multi-version evaluation. This paper tackles this challenge and introduces a technique to identify and quantify individual influences caused by differing process versions.

Key words: Business Process Management, Business Process Models, Process Evaluation, Process Analysis, Process Versions

1 Introduction

Nowadays, organizations face a competitive market environment and need to be as flexible and efficient as possible. Therefore, they are managed in a process-oriented fashion to be able to react quickly on market changes. Business process management (BPM) combines concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of these business processes in an iterative way [1]. One corner stone of the BPM life cycle is the evaluation of business process executions to identify the weak points and potential improvements. Central to BPM are process models as they explicitly describe the operations that need to be carried out to reach the companies' business goals and are used, among others, for enactment.

For instance, changing market conditions, elimination of identified weak points by implementing business process improvements, or legal changes result in business process model changes and therewith in a variety of business process models, so-called process versions. Therefore, reliable business process analysis requires all versions of a business process taken into account to get a complete picture of the business process executions so far. Most information systems that are used for business process execution collect data about the performed activities. This data can already be analyzed in the context of business processes and their corresponding process models by using methods and techniques from the field of business process intelligence (BPI) [2, 3, 4]. Still, these methods and concepts

are targeting on one specific process version only and are not able to handle the evolution of a process including several process versions.

In this work, we present an approach for identifying and quantifying variations of process metrics that are caused by the differences in the process versions to allow process analysis with several versions of a process. We introduce a technique to determine the actual effects of structural process model changes between process versions for a specific process metric. With the approach, we make several process versions comparable based on their common denominator. A multi-version evaluation computes a single metric, e.g., a point-to-point duration metric, using execution data from the set of all process versions or a subset of those. The studies in this work are performed using as example the point-to-point duration. This metric represents the time that elapsed between reaching a first point (A) and reaching a second point (B) during the execution of a process. In this work, we show the algorithm for determining which parts of a process model, i.e., process fragments, affect a specific point-to-point duration. These calculated process metrics can be used for process monitoring and analysis.

In the remainder, we introduce basic terms and concepts our work bases on (Section 2) followed by the description of a motivating example from health care in Section 3. In Section 4, we explain our approach in detail and present how we use newly introduced measured region models and process model structuring techniques to determine whether process model changes can be highlighted for multiple process versions analysis. In Section 5, we apply the presented approach to our motivating example and show concrete duration delta calculations. We put our work into the context concerning related work in Section 6, before we summarize and conclude the paper in Section 7.

2 Preliminaries

For this work, we rely on a simplified notion of process models defined as follows.

Definition 1 (Process Model). *A* process model *is a tuple* $P = (N, F, \kappa)$*, where* N *is a finite set of nodes* N *and* $F \subseteq N \times N$ *is the control flow relation.* $N = A \cup G \cup E$ *comprises the activity nodes* A*, the gateways* G*, and the start and end nodes* $E = \{s, e\}$*.* $\kappa \colon G \to \{and, xor\}$ *associates each gateway node with a type.* ◇

Activities have exactly one predecessor and one successor. Whereas gateways have either one predecessor and several successors (split) or several predecessors and one successor (join). We assume the process model to be structural sound, to ensure that the process model P has exactly one start node s and one end node e, and every node is part of a valid path from s to e. Refactoring techniques may be applied to normalize models that do not match these assumptions [5]. We further require process models to be block-structured and their petri-net representation to be sound [6], which ensures the absence of behavioral anomalies such as deadlocks.

The process model serves as a blueprint for the so-called process instances. A process instance is a concrete execution of the activities defined in a process model.

Process monitoring and analysis is targeting on the evaluation and comparison of process instances.

As we are interested in measuring the time it takes to execute a part of a process instance, we require measurement points that denote a particular node is reached. However, positioning of such a point on process model node level only is too coarse-grained, because it is often required to track the begin and the end of an activity for instance, e.g., to distinguish between waiting times and the time it actually requires to carry out a certain activity. Therefore, we assign a *node life cycle* to each node of the process model to allow a more fine-grained positioning of the measurement points based on the state transitions of those life cycles, cf. [7].

Definition 2 (Node Life Cycle). *A* node life cycle *is a tuple* $L = (S, T)$*, where* S *is a finite set of* node states *and* $T \subseteq S \times S$ *is a finite set of* node state transitions*. Let* $P = (N, F, \kappa)$ *be a process model. There exists a function* $\varphi : N \to L$ *that assigns a node life cycle to every node* $n \in N$ *of* P*.* ◇

State transitions are the elementary constructs that can be leveraged to position points in the process model where measurements could be taken. A point where a measurement could be taken on a state transition of a node is called *process event monitoring point* (PEMP), cf. [7]. The set of all node state transitions of a process model $P = (N, F, \kappa)$ is comprised by $\bigcup_{n \in N} \{(n, t) | t \in T_{\varphi(n)}\}$, each of which could be potentially linked to a PEMP. $T_{\varphi(n)} \subseteq T$ represents all node state transitions $t \in T$ returned by function $\varphi(n) = (S, T)$ for node $n \in N$.

Definition 3 (Process Event Monitoring Point). *A* process event monitoring point *is a tuple* $PEMP = (P, n, t)$*, where* P *is the process model it is contained in,* $n \in N$ *is the node it is created for, and* $t \in T$ *is the state transition within the node life cycle of* n *it is assigned to.* ◇

3 Motivating Example

In our motivating example, we will look on the diagnosis of a patient's disease in a hospital, see Fig. 1a. The first activity in the initial diagnosis process version is "Withdraw Blood Samples". Afterwards the blood samples are analyzed (activity "Analyze Blood Samples") in parallel to the examination of the patient (activity "Examine Patient"). Updating the patient's records finishes the process (activity "Update Patient's Records").

For quality assurance it is essential to measure the time consumed between the point in time when the patient is introduced to the doctor (A) and the patient's data are updated according to the examination results and the blood sample analysis (B). For measuring such a point-to-point duration, we utilize PEMPs. A PEMP, cf. Definition 3, is bound to a specific state transition of a node's life cycle, cf. Definition 2 to indicate that this state transition happened by occurrence of the corresponding data in the information system landscape. Referring to Fig. 1a and Fig. 1b, we assigned to each activity of the process model the same node life cycle consisting of the states enabled, running and terminated with the corresponding state transitions *(e)nable*, *(b)egin* and *(t)erminate*. However,

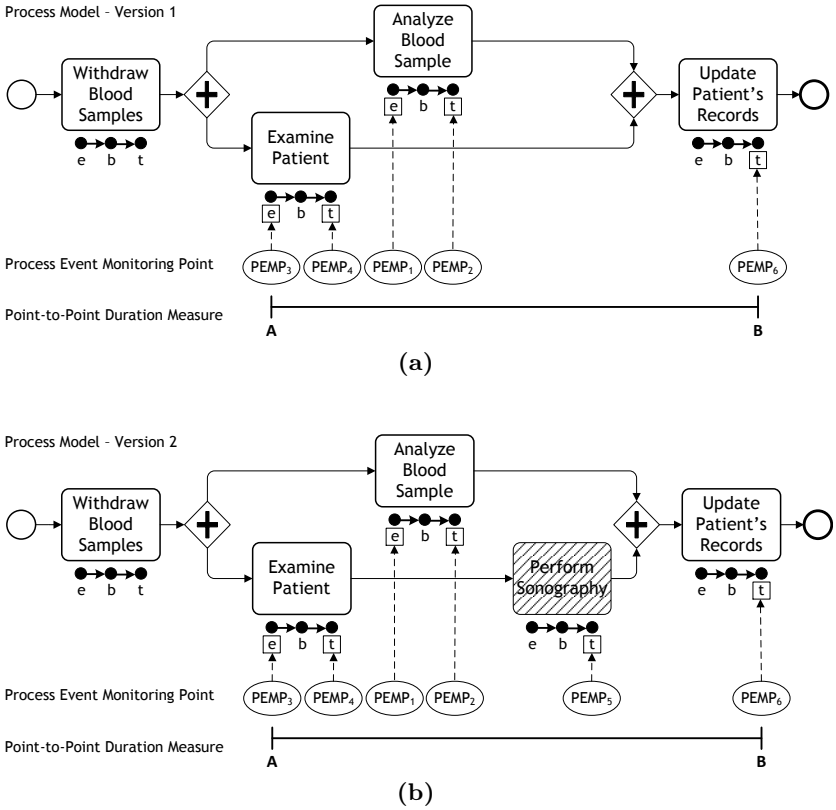


Fig. 1: Evolved process model of a diagnosis process for a patient. (a) shows process version 1 with the activities "Withdraw Blood Samples", "Analyze Blood Samples", "Examine Patient", and "Update Patient's Records". (b) shows process version 2 with the additionally inserted activity "Perform Sonography".

the solution allows applying individual life cycles to the nodes. In Fig. 1 the life cycles are represented by black dots and edges between them. A black dot represents a node state transition while an edge between two black dots represent the node state. In the scenario not every node state transition is observable by occurance of data from the information systems. Therefore, only PEMPs that can be observed are shown in Fig. 1. $PEMP_3$ in Fig. 1 indicates that activity "Examine Patient" is enabled for execution and $PEMP_6$ represents the end of the patient's data record update.

Due to legal changes, the hospital adjusted the diagnosis process for the patients by adding a process fragment, i.e., the activity "Perform Sonography", see Fig. 1b - shaded activity. In this second process version the termination of the sonography could be captured by $PEMP_5$. Introducing a process fragment, e.g., an activity, might affect the overall run time of the diagnosis process instances and therewith the point-to-point duration between the introduction of the patient to the doctor and the completion of patient's data update. An analysis summarizing

process instances of both versions, the first version without the sonography, and the second version including the sonography, is not possible, because the results would be mixed up and falsified accordingly.

With the comparability of process instances of several different process versions several questions about the quantification of effects between the single process versions arise. For instance in our case it would be interesting to answer the following: (i) What are the diagnosis durations after adding sonography activity? (ii) Are the durations for the unchanged part of the model different before and after adding sonography?

4 Solution

Our approach targets on the determinability of effects caused by process changes. We will show the concept of our approach using the example of the point-to-point duration process metric. The point-to-point duration is a measure for the elapsed time it takes to traverse the sub-graph of a process model that is spanned by an activity indicating the start (A) and an activity indicating the end (B), see Fig. 1a and Fig. 1b. We assume that both activities can be measured by a PEMP. We use the existing process model, the defined point-to-point duration metric, the change log to the process model that captures all the changes applied to the model, and the defined PEMPs to apply our four-step approach.

First of all, a *measured region model* is derived from the source process model (version 1) based on the point-to-point duration metric. A node of the process model is part of the measured region model if its duration influences the point-to-point duration. The resulting connected sub-graph contains all such influencing nodes and builds the basis for our approach (see Section 4.1). A measured region model is a process model according to Definition 1. Secondly, the measured region model is enriched by the changes applied to the source process model contained in the change log (see Section 4.2). Afterwards the enriched measured region model is transformed to a *component tree* by utilizing the technique of the refined process structure tree (RPST) [8]. The resulting component tree is enriched with the information of defined PEMPs and an *event monitoring model* is built in a third step (see Section 4.3). In the fourth step, based on this event monitoring model it is determined whether the effects of the process model changes can be identified and quantified for more accurate process analysis (see Section 4.4).

4.1 Transformation to a Measured Region Model

The *measured region model* is the basis for determining, which structural changes might affect the point-to-point duration metric, as it contains the relevant process nodes only; see the processes shown in Fig. 2 for an example. The left side shows the original process model and the right shows the measured region model for the given point-to-point duration metric. Per definition, all elements not contained in the measured region model can be removed from the process model without affecting the given point-to-point duration metric.

Fig. 2a shows that every node on a path from node A to node B (including both nodes) is contained in the measured region model. However, this is not the

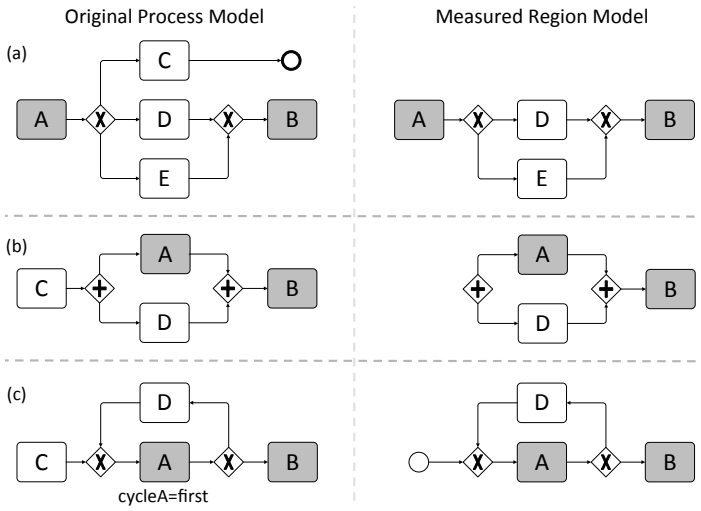


Fig. 2: A selection of measured region models for a point-to-point duration metric influenced by (a) an exclusive process fragment, (b) a parallel process fragment, and (c) a loop process fragment. In (c) a start event is added to assures that there is always a single entry and single exit point in the measured region model.

case with parallel and inclusive join gateways as they represent *synchronization points* in a process, which wait for the completion of all incoming branches. If node A is contained in such a branch, then the other branches are not on a path from node A to node B, but they might still influence the point-to-point duration because of the synchronizing behavior of the joining gateway, see Fig. 2b. This does not apply to exclusive conditional join gateways, as they trigger immediately when a single branch is completed.

When considering cycles, the start point, end point, or both of the duration metric might be part of the cycle. Thus, the relating activities can be executed several times and therewith, multiple activity instances exist. Therefore, we need to consider user input to know whether the first or last activity instance is relevant for the point-to-point duration. We describe these inputs in the metric parameters $cycleA, cycleB = \{first, last\}$. It depends largely on the parameters of the metric, whether the corresponding cycle is part of the region or not, see Fig. 2c. If only node A (and not node B) is part of a cycle and $cycleA = first$, then that cycle needs to be part of the region. If $cycleA = last$, then the cycle is not part of the region, as it was not executed again after the last triggering of A. The parameter $cycleB$ has a corresponding effect on the inclusion of a cycle containing node B.

The algorithm to create a measured region model starts with the search of all relevant paths from A to B by using a traversal of the succeeding nodes starting with node A. If node B is encountered, then the current path is saved as a relevant path and that path is not traversed further. Traversal can also be stopped, if a node has no unvisited successors or node A is encountered. In the latter case, the path is also marked as relevant, if A is in a cycle and $cycleA = first$. If B is in a

cycle and $cycleB = last$ we also need to find all paths from node B to itself. This case can be handled by starting a second traversal of succeeding nodes in the process model, this time starting from node B. As in the first traversal, paths to node A or node B are added depending on the mentioned parameters.

Once the set of relevant paths has been determined, each path is traversed in order and a new node is added to the measured region model for each node in the path, if the node does not exists in the model already. Further, we have to consider the case of encountering synchronizing gateways, whose start gateway is outside of all relevant paths. While traversing each path, we keep two counters, one for splitting and another for joining parallel and conditional gateways. If both counters are equal and such a joining gateway is encountered, then we know that this gateway does not have a splitting counterpart in the measured region model yet. Thus, the next step is to add all its incoming branches to the measured region model together with the splitting gateway to ensure structural soundness. The original process model is traversed backwards using the predecessors of nodes to find the first splitting gateway that is common to all branches that is then used as split in the measured region model. This backward tracing starts from the joining gateway.

To connect all the nodes, a directed edge is added from each node to it's successor in the path. Eventually, an explicit start node s or end node e is added when the measured region model includes cycles. This assures that there is always a single entry and single exit (SESE) point in the measured region model. As the final step, gateways with a single predecessor and a single successor are removed from the measured region model as they do not affect the duration and are not allowed in a process model per definition.

With this, the creation of the measured region model is complete. The measured region model contains the elements of a process model that affect a specific point-to-point duration from activity (A) to another activity (B). In the following, we will enrich this measured region model with information from the process model change log.

4.2 Enriching Measured Region Models with Change Information

In a second step, the changes to the source process model as described in the change log are applied to the measured region model created in the previous step. In this work, we assume that the change log contains only change patterns as described by Weber et. al [9]. The change patterns describe the process model change relating a particular part in the process model, the so-called process fragment. These change patterns do not contain atomic or complex activities only, but also sub-graphs that need to have a SESE. The change operations are rather "simulated" in the measured region model, as the changes are marked in the model only, e.g., a removal operation applied to a measured region model's process fragment is marking that corresponding fragment only, e.g., the removal of an activity. Same holds for adding process fragments. This is necessary to keep the previous positions of the nodes for further evaluation.

For instance, the change log for our source process model shown in Fig. 2a contains an entry for the insertion of a process fragment, i.e., an additional

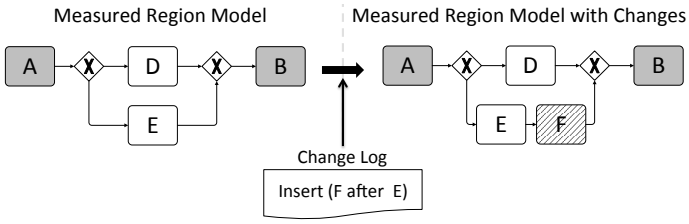


Fig. 3: Applying change operations contained in the process model change log to the measured region model.

activity F after activity E in the lower branch of the process model, e.g., stated by the change operation *Insert(F after E)*. This operation is applied to the measured region model and F is marked as inserted, see Fig. 3 - shaded activity.

A specialty in this processing step is the application of change patterns to the start and end point of the point-to-point duration metrics, i.e., to node A and B of the shown example. Changes involving node A or node B need to be handled separately, as they change the basis on which the measured region model is defined. Once the measured region model is enriched by the change information, the structure of the resulting model needs to be parsed to attach the defined PEMPs and to determine the effects of the change in the next steps.

4.3 Build the Event Monitoring Model

For parsing the structure of the measured region model, we apply the approach of the RPST introduced by Vanhatalo et. al [8]. The RPST represents the process model structured as a tree with the root symbolizing the whole graph, the nodes representing canonical elements of the parent element, and the leaves denoting the edges of the graph.

As we are interested in the nodes of the process graph, the RPST has to be transformed into a *component tree*. The component tree consists of the canonical fragments of the RPST as process components, i.e., the component tree nodes, and the measured region model nodes itself as the component tree leaves. During the transformation, canonical fragments of the RPST are transformed into process components that subsume the process elements that are contained in the corresponding canonical fragment of the RPST. From the children of the canonical fragments of the RPST, the original measured region model nodes got extracted and attached to the corresponding process component as a child. For instance, the measured region model with the applied changes from the previous step, see Fig. 3, is structured in a component tree as shown in Fig. 4a. Note that during the transformation of the RPST into a component tree, measured region model nodes may get duplicated in different process components of the component tree. However, this will not be an issue for our approach. The marking of the applied changes are still contained in the component tree.

To the component tree leaves, the corresponding life cycles are attached by $\varphi : N \rightarrow L$ to every node $n \in N$ of the measured region model P. With that step, every component tree leaf gets new children indicating the state transitions of the corresponding life cycle as shown in Fig. 4b. For simplicity reasons, every node gets the state transitions (e)nable, (b)egin and (t)erminate assigned.

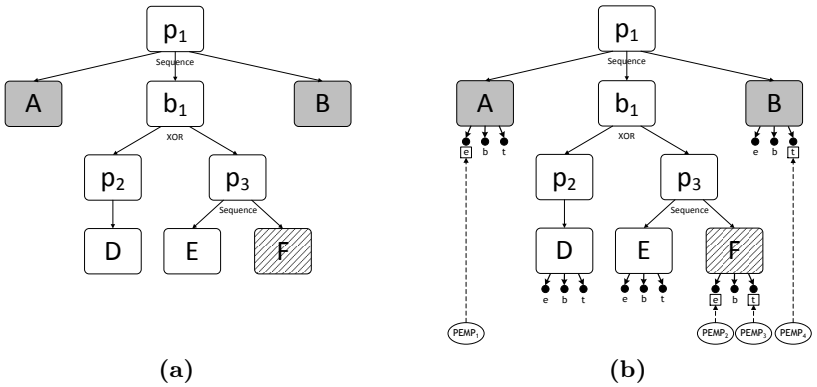


Fig. 4: (a) shows the measured region model enriched by changes structured in a component tree. To this component tree the corresponding life cycles of the process nodes are attached by $\varphi(n)$ and the defined PEMP are assigned (b).

Afterwards, the defined PEMPs are connected according their definition $PEMP = (P, n, t)$ to the component tree containing the measured region model nodes and the assigned state transitions. P is the indicator for the initial process model and the resulting measured region model, n the measured region model node the PEMP belongs to, and t the exact state transition that could be observed by the PEMP. In Fig. 4b, the enablement of node A, the termination of node B as well as the enablement and termination of the newly introduced node F can be observed by a PEMP.

With the consolidation of measured region model information and PEMP definitions the *event monitoring model* is created. Based on that, several algorithms can be run to analyze which information can be derived out of the spread PEMPs. For instance, it is possible to indicate whether the effects of a process model change can be identified and quantified to enable process analysis on multi-version processes.

4.4 Determining the Evaluation of Process Model Change Effects

Based on the event monitoring model, it is possible to determine if and how process changes influence a particular point-to-point duration metric. We categorize the determinability of effects caused by a process change to a process metric into the following four groups:

- Unchanged—the process model changes do not effect on the process metric.
- Indeterminable—the effects of process model changes cannot be identified nor quantified for any process instance.
- Determinable—the effects of process model changes can be identified and quantified for all process instances.
- Partly Determinable—the effects of changes can be identified and quantified only for some process instances.

We consider not only the process versions but the single process instances for categorization as we would like to provide a detailed indication on the determinability of the quantification of process changes. If there are changes whose effects can not be determined exactly, the algorithm determines whether this indeterminability holds for all process instances. To give an example, an indeterminable change in a sequence of process fragments will cause the effect of the whole sequence to be indeterminable. In the contrary, an indeterminable change in a branch of an XOR causes only those process instances that executed that particular branch to be indeterminable, thus, the effects are partly determinable.

Changes can be quantified, if their effects can be measured by PEMPs. This is the case when the beginning and the end of the added, removed or changed process fragment, e.g., the insertion of an activity, can be measured by a PEMP. As shown in Fig. 4b, for the added activity F the enablement and termination can be measured. Thus, additional time spend during process execution that may result from the activity F can be identified and quantified to allow a comparison between the initial process version and the changed one containing activity F.

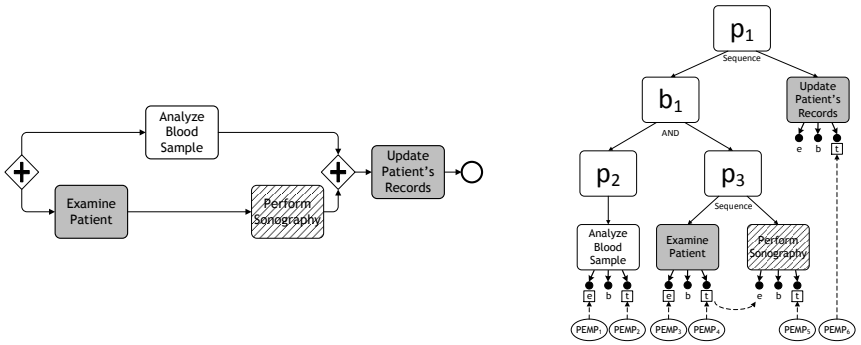
The process model changes may not be measurable directly. One can think of only measuring the enablement or the termination of F or nothing. In this case, the predecessor resp. successor activities may provide some more insights with respect to the timestamps of the newly introduced process fragment. If we assume that two activities are in sequence and between the termination of the first activity and the enablement of the second activity no time is consumed, then we can use the rule, that the measured termination of the first activity is also the time of the unmeasured enablement of the second activity and vice versa.

With the categorization of the determinability of effects caused by a process change to a process metric, it is stated whether algorithms that quantify the changes, e.g., by calculating the duration influence, could be applied or not. Such calculations are not in scope of this paper, however, we will give an example in our case study where we validate the approach based on the motivating example.

The presented approach was illustrated along the point-to-point duration metric. However, the approach can be adapted to other metrics as well, e.g., point-to-point cost metric, the probability to reach a certain point in the process, or the number of iterations in a process. As long as the metric has a defined start and end point the approach can be applied as described with (i) transforming the process model into a measured region model based on the start and end points, (ii) enriching the measured region model with change information, (iii) build the event monitoring model, and (iv) evaluate the effects to the metric cased by process model changes if possible.

5 Case Study

In the following, we will apply the presented approach to our motivating example shown in Section 3. As shown in Fig. 1, we have a source process model (process version 1), on which the insertion of a process fragment, i.e., the insertion of the activity "Perform Sonography" is performed. This change is logged in the corresponding process model change log and described as *Insert(Perform Sonography after Examine Patient)*. This indicates that the additional activity needs



(a) **(b)**

Fig. 5: (a) shows the measured region model enriched by changes from the process change log that is then transformed in an event monitoring model (b).

to be inserted after activity "Examine Patient". As already stated in Section 3, the required point-to-point duration metrics reaches from the enablement of the activity "Examine Patient" until the completion of patient's data updates. Furthermore, in the two process versions six different PEMPs are defined. $PEMP_1$ and $PEMP_2$ representing the start and the end of the activity "Analyze Blood Samples". $PEMP_3$ and $PEMP_4$ providing time measures for the enablement resp. termination of activity "Examine Patient", $PEMP_5$ indicating the termination of the sonography, and $PEMP_6$ providing information about the termination of the activity "Update Patient's Records". Based on this information, we will evaluate if the effect of adding the sonography to the diagnosis process can be identified and quantified.

Therefore, the point-to-point duration metrics and the process model, see Fig. 1a is used to create a measured region model, as it can be seen in see Fig. 5a. The specialty with the process model in our example is that we need to consider the upper branch of the AND split as well, although the beginning of the point-to-point duration metrics is specified just in lower branch. However, the upper branch will influence the duration in cases when "Analyzing blood samples" will take more time than the patient examination (process version 1) resp. patient examination and sonography (process version 2).

Afterwards, the changes contained in the change log are applied to the measured region model. In the change log only the insertion of activity "Perform Sonography" is recorded. Marking the corresponding process fragment in the measured region model simulates this change, see Fig. 5a - shaded activity.

Using the measured region model as a basis, the event monitoring model is build by constructing a component tree first, see Fig. 5b. The component tree is then enhanced by the life cycle information for each node. In the example, the same life cycle consisting of state transactions enable, begin and terminate and the corresonding states enabled, running and terminated is used for all activities. To these state transitions the corresponding PEMPs are assigned and therewith the event monitoring model is complete.

Table 1: Measures during patient diagnosis process correlated to PEMPs. The ID of the patient identifies process instances.

patientID	$PEMP_1$	$PEMP_2$	$PEMP_3$	$PEMP_4$	$PEMP_5$	$PEMP_6$
1	08:32	08:58	08:32	09:35		09:54
2	09:05	10:40	09:55	10:10		11:03
3	09:43	10:46	09:43	10:55		11:08
4	10:02	10:10	10:02	10:55	11:58	12:14
5	13:30	14:52	13:30	14:05	14:49	15:13
6	13:42	13:56	13:42	15:00	15:36	15:44

Having the event monitoring model, we can determine, that we can quantify the effects caused by the insertion of activity "Perform Sonography". The evaluation could happen due to the fact that we can measure the completion of the sonography, i.e., by $PEMP_5$. Further, we can determine the enablement of the sonography by utilizing $PEMP_4$ that is assigned to the termination of activity "Examine Patient", the predescessor of the inserted activity. Since we assume that during control flow no time is consumed this assumption holds.

In the following, we execute the approach to the motivating example for calculating the overall effect of all changes related to a particular point-to-point duration. We call these effects *duration delta*. Each structural change can be broken down to the addition or removal of process fragments. The local duration delta of such a change is thus equal to the duration of the added or removed process fragments. Once all local duration deltas are determined, we need to compute their effects on the overall duration. This can be done by applying techniques that are very similar to the aggregation of quality of service properties for process models, as described in [10] or [11]. These techniques determine the value of properties like execution time or cost for a whole process model, given the corresponding property values for each atomic process element. We can aggregate all local duration deltas to compute the duration delta for the whole process instance. We use a recursive calculation of duration deltas, which starts with the root fragment and terminates when a node is reached. When the recursion reaches a node, we determine the local duration delta for all its instances (remember that loops may be involved). The local duration deltas are then aggregated by the parent fragments according to their types during the roll-up of the recursion until a single value is determined by the root fragment.

For several instances of the diagnosis process, measures are captured by the PEMPs, see Table 1. Therewith, the requested point-to-point duration as well as the effect to the duration caused by the insertion of the sonography can be quantified. Patient 1 needs for the diagnosis after his blood was taken 82 minutes, patient 2 118 minutes, and patient 3 85 minutes. From this data, we can say that a patient needs in average 95 minutes (process version 1). The following patients where treated including the sonography. Patient 4 needs 132 minutes, patient 5 103 minutes, and patient 6 122, in average 119 minutes (process version 2). These results are not easy to compare adequately. For example, if we calculate the average of all patient diagnoses, we will get 107 minutes. That will lower the good results of the processing time of the first versions and increase the result of

the second process version. More comparable results would be achieved by hiding the results of the sonography for the multi-version process analysis. In that case, the durations of the first three patients will remain, but the results for patient 4 will change to 69 minutes, and for patient 6 to 86 minutes. For patient 5 103 minutes will remain as well, be cause in her case the blood analysis was on the critical path. With these results we will get an average of 90,5 minutes, which is more meaningful instead of comparing process versions without considering the effects of process changes.

With these numbers, the questions raised in Section 3 can be answered. (i) What are the diagnosis durations after adding sonography activity? This can be answered by comparing the average durations of process version 1 (95 minutes) and process version 2 (119 minutes). The sonography extends the diagnosis duration by 24 minutes in average. (ii) Are the durations for the unchanged part of the model different before and after adding sonography? The answer is yes, as we can see if we hide the sonography durations, the overall run time got slightly better. Process version 1 took 95 minutes in average; if we compare all process instances (process version 1 and 2) then the average is 4,5 minutes faster.

Multi-version process monitoring and analysis is enabled by these measures and the ability to quantify process model changes. It brings huge benefits to the evaluation phase of the BPM life cycle and therewith allows better improvements on business processes.

6 Related Work

The presented work builds on process versions and their process model changes. However, we abstract from actual details of version management methods and assume that the implemented version management method provide two essential functionalities: (1) retrieving a specific process version and (2) retrieving the set of change operations that transform one process version into another. The version management methods [12] and [13] meet these assumptions. [12] proposes a method for version control of process models based on change operations. The initial version of a process is stored as a whole, while all succeeding versions are only stored as the set of basic change operations. [13] decomposes process models into process fragments using SESE components, which then are used as the primary entities of a versioning system for process models.

One of the inputs for our approach is the process model change log. These changes can be described by change patterns that are investigated in multiple works. In [14] change patterns on arbitrary process fragments are used to model, store and retrieve process variants resp. versions by utilizing the concept of adjustment points. [15] defines a set of change patterns on canonical SESE components in a RPST. Based on these patterns, the authors are able to determine all changes needed to transform one process model into another. This technique could be used to determine the set of change operations between two process versions, if the version management method cannot provide them. Weber et. al [9] define an extensive and detailed classification of changes to process models into so-called *adoption patterns*. We used these adaption patterns in our approach,

because they provide the right level of abstraction by being detailed enough to unambiguously define their effects on a process model, while still carrying enough semantic information to determine the intention behind the changes.

The approach presented in this paper aims to enable business process intelligence (BPI) in multi-version process execution environments. The capability to monitor, visualize, and evaluate business process execution is one core topic of BPI [2], which addresses "managing process execution quality by providing several features, such as analysis, prediction, monitoring, control, and optimization" [3]. Bringing awareness about process changes that influence process metrics can be seen as one of that features of BPI. In [2], the authors argue that process monitoring and analysis are vital to BPI and propose, based on the specific requirements of BPI, a reference architecture, composed of an integration layer, a functional layer, and a visualization layer. The approach presented in this paper targets at the functional layer, i.e., enabling the comparability of process execution measurements from several process versions.

Del-Río-Ortega et al. [16] introduced the concept of process performance indicators (PPI), the process related form of key performance indicators, to enable process evaluation. PPIs target on process measurements, e.g., time, costs, and occurrences. Our presented approach could support the measurements from several process versions and can ensure the comparability of the PPIs.

7 Conclusion

In this paper, we presented an approach to determine whether the effects of process model changes influences a particular process metric, i.e., a point-to-point duration metric. We further investigate if the influences can be quantified so that a multi-version process analysis is applicable. Therefore, we reduce the original process model to a measured region model based on the given point-to-point duration metric to identify the metric influencing process model elements. Based on the measured region model, we apply the corresponding process model changes recorded in the change log and structure the resulting model via a component tree. The component tree is enriched by the life cycle information for every node and the corresponding process event monitoring points are attached to it. This enriched component tree form the event monitoring model that could be used for determining whether the process model changes could be identified and quantified to ensure a reliable multi-version process analysis. Motivated by an example from a German hospital, we show the whole technique for a diagnosis process.

In this paper, we used the point-to-point duration process metric as example. The effects on other specific process metrics can be determined with the event monitoring model as well, which we will prove in future work. Our approach does not consider indirect dependencies between process elements nor non-structural properties of process elements. We disregard sub-process relations as such process models could be flattened. The approach does not support process models with nested loops, multi-instance nodes and loop nodes.

With the approach multi-version process analysis is possible. Effects of process model changes may can be identified and quantified and hide for several process analysis tasks. The approach applies also to process monitoring and process

simulation. It is possible to simulate process model changes and evaluate the behavior of the surrounding nodes with respect to the given process metric.

Acknowledgment. Many thanks to the former student Philipp Maschke for the related initial work done during his master thesis, and the many fruitful discussions about the topic, its challenges, and the possible solutions.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Second Edition. Springer (2012)
2. Mutschler, B., Reichert, M.: Business Process Intelligence. EMISA Forum **26**(1) (2006) 17–31
3. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.C.: Business Process Intelligence. Comput. Ind. **53**(3) (April 2004) 321–343
4. Azvine, B., Cui, Z., Nauck, D., Majeed, B.: Real Time Business Intelligence for the Adaptive Enterprise, IEEE (2006) 29
5. Vanhatalo, J., Völzer, H., Leymann, F., Moser, S.: Automatic Workflow Graph Refactoring and Completion. In: ICSOC, Springer (2008) 100–115
6. van der Aalst, W.M.P.: Verification of Workflow Nets. In: ICATPN '97, Springer (1997) 407–426
7. Herzberg, N., Meyer, A., Weske, M.: An Event Processing Platform for Business Process Management. In: EDOC, Vancouver (2013) (to appear)
8. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data & Knowledge Engineering **68**(9) (September 2009) 793–818
9. Weber, B., Rinderle, S., Reichert, M.: Change Patterns and Change Support Features in Process-Aware Information Systems. In: CAiSE, Springer (2007) 574–588
10. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS Aggregation for Web Service Composition using Workflow Patterns. In: EDOC, IEEE (2004) 149–159
11. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of Service for Workflows and Web Service Processes. Web Semantics: Science, Services and Agents on the World Wide Web **1**(3) (2004) 281–308
12. Bae, H., Cho, E., Bae, J.: A Version Management of Business Process Models in BPMS. In: APWeb/WAIM, Springer (2007) 534–539
13. Ekanayake, C.C., La Rosa, M., Ter Hofstede, A.H., Fauvet, M.C.: Fragment-based Version Management for Repositories of Business Process Models. In: OTM, Springer (2011) 20–37
14. Hallerbach, A., Bauer, T., Reichert, M.: Configuration and Management of Process Variants. Handbook on Business Process Management 1 (2010) 237–255
15. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Detecting and Resolving Process Model Differences in the Absence of a Change Log. In: BPM, Springer (2008) 244–260
16. Del-Río-Ortega, A., Resinas, M., Ruiz-Cortés, A.: Defining Process Performance Indicators: An Ontological Approach. In: OTM, Springer (2010) 555–572