Process mining using BPMN: relating event logs and process models

Kalenkova, A.A.; van der Aalst, W.M.P.; Lomazova, I.A.; Rubin, V.A.

Published in:
Software and Systems Modeling

DOI:
10.1007/s10270-015-0502-0

Published: 01/10/2017

Document Version
Accepted manuscript including changes made at the peer-review stage

Softw Syst Model
DOI 10.1007/s10270-015-0502-0

REGULAR PAPER

Process mining using BPMN: relating event logs and process models

Anna A. Kalenkova[1] · Wil M. P. van der Aalst[1,2] · Irina A. Lomazova[1] · Vladimir A. Rubin[1]

Received: 11 March 2015 / Revised: 17 July 2015 / Accepted: 22 September 2015

Abstract Process-aware information systems (PAIS) are systems relying on processes, which involve human and software resources to achieve concrete goals. There is a need to develop approaches for modeling, analysis, improvement and monitoring processes within PAIS. These approaches include process mining techniques used to discover process models from event logs, find log and model deviations, and analyze performance characteristics of processes. The representational bias (a way to model processes) plays an important role in process mining. The BPMN 2.0 (Business Process Model and Notation) standard is widely used and allows to build conventional and understandable process models. In addition to the flat control flow perspective, subprocesses, data flows, resources can be integrated within one BPMN diagram. This makes BPMN very attractive for both process miners and business users, since the control flow perspective can be integrated with data and resource perspectives discovered from event logs. In this paper, we describe and justify robust control flow conversion algorithms, which provide the basis for more advanced BPMN-based discovery and conformance checking algorithms. Thus, on the basis of these conversion algorithms low-level models (such as Petri nets, causal nets and process trees) discovered from event logs using existing approaches can be represented in terms of BPMN. Moreover, we establish behavioral relations between Petri nets and BPMN models and use them to adopt existing conformance checking and performance analysis techniques in order to visualize conformance and performance information within a BPMN diagram. We believe that the results presented in this paper can be used for a wide variety of BPMN mining and conformance checking algorithms. We also provide metrics for the processes discovered before and after the conversion to BPMN structures. Cases for which conversion algorithms produce more compact or more complicated BPMN models in comparison with the initial models are identified.

Keywords Process mining · Process discovery · Conformance checking · BPMN (Business Process Model and Notation) · Petri nets · Bisimulation

Communicated by Prof. Ulrich Frank.

This work is supported by the Scientific Fund of the National Research University Higher School of Economics and is supported by Russian Fund for Basic Research (Project 15-37-21103).

✉ Anna A. Kalenkova
akalenkova@hse.ru

Wil M. P. van der Aalst
w.m.p.v.d.aalst@tue.nl

Irina A. Lomazova
ilomazova@hse.ru

Vladimir A. Rubin
vrubin@hse.ru

1 National Research University Higher School of Economics, Moscow, Russia

2 Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

1 Introduction

Process-aware information systems (PAIS) are the systems designed to manage processes, operating in various domains of human activity. There is a natural requirement to monitor their work and analyze executed processes. In many cases, analysts are interested in a real system behavior, which may be hidden from domain experts and system engineers. This

Published online: 20 October 2015

Table 1 An event log of a booking process

Case ID	Event name	Timestamp	Price	Client IP
1	Book flight	2014-12-24 08:30:00:232	145	188.44.42.45
1	Get insurance	2014-12-24 08:31:05:171	23	188.44.42.45
2	Book flight	2014-12-24 08:31:08:543	94	93.180.0.62
1	Book hotel	2014-12-24 08:32:08:703	295	188.44.42.45
3	Book flight	2014-12-24 08:32:11:534	192	188.44.50.103
1	Pay	2014-12-24 08:34:17:456	463	188.44.42.45
1	Confirm	2014-12-24 08:35:17:537	463	188.44.42.45
...	...	...	...	...

real behavior can be reconstructed from the event logs. For that purpose, Data Science approaches can be applied. The interest in Data Science and Big Data signifies the growing importance of evidence-based approaches. Process mining techniques provide wide range of data-driven approaches that are process-centric at the same time. Other data-driven approaches are often no process-centric.

Process mining offers techniques for automatic discovery of *process models* from event logs, checking compatibility of process models and event logs (conformance checking) and enhancing discovered processes with additional data [1]. Process mining has been successfully applied in a variety of *application domains* such as healthcare [2], tourism [3] and education [4]. There is a IEEE process mining community, including more than 60 organizations [5]. Moreover, there is a wide range of *research and commercial tools* available in this area: ProM, Disco (Fluxicon), ARIS Process Performance Manager (Software AG), Perceptive Process Mining (Perceptive Software), ProcessAnalyzer (QPR) and Celonis.

Today, BPMN 2.0 (Business Process Model and Notation) [6] is the de facto *standard notation* for modeling business processes understandable by a wide audience of people. Business analysts and product managers, technical designers and developers, system and enterprise architects effectively use this notation in their everyday job almost everywhere where BPM is applied. An absolute majority of *freeware and commercial BPM tools* and Business Suites, like Oracle BPM Suite, IBM Business Process Manager, jBPM, Activiti, Appian BPM Suite, Bizagi BPM Suite, MagicDraw Enterprise Architect (Sparx), Mega Process (MEGA), Signavio Process Editor and others, either natively support BPMN or provide conversion in order to stay compatible and up to date. BPMN applicability and best BPMN modeling practices are presented in [7–9].

The *representational bias* used for process mining is not only relevant for the understandability of the results, it is also vital to guide process discovery by setting a suitable class of target models. Using the *BPMN notation as a representational bias within process mining* opens excellent perspectives for applicability of process mining: Discovered process models become available and *understandable* by the majority of users, the models can be *imported/exported* from/to any BPMN-aware modeling tool and executed, process mining techniques can be easily *integrated* to the existing suites (BPMN serves as an interface in this case). Moreover, BPMN models allow for the combination of *different perspectives* varying from control flow (including subprocesses) to the perspective of resources.

In this paper, we present methods for discovering the *control flow perspective of a process in terms of BPMN*. It should be noted that process mining offers *plenty of algorithms* for the control flow discovery and each of them has its own characteristics. *The goal is not to invent new algorithms, but to benefit from the existing ones and to make them BPMN-compatible*. Thus the discovery of the control flow relies on *conversion algorithms* and *existing process mining techniques*. To that end we firstly formalize the *semantics* for a subset of BPMN models and then present the conversion algorithms from well-known control flow modeling formalisms such as *Petri nets* (including non-free-choice Petri nets), *causal nets* [10] and *process trees* [11] to BPMN. The conversion algorithms presented in the paper are also given the *justifications* in order to show that behavioral properties of process models discovered from an event log are preserved. Moreover, we show relations between languages of Petri nets and corresponding BPMN models, tacking into account properties of the initial Petri nets.

As a short introductory example, let us consider an event log[1] reflecting a small portion of the history of a ticket booking process, which is presented in Table 1. In this process, people use a Web site to book a flight, a hotel, get insurance and pay for the ticket. Different people in different cases execute these activities in a different order. Beside case identifiers, event names and timestamps, an event log can contain

[1] In order to give an intuitive example, we consider a simple and expressive synthetic event log. The analysis of process models discovered from real-life event logs is demonstrated in Sect. 9.

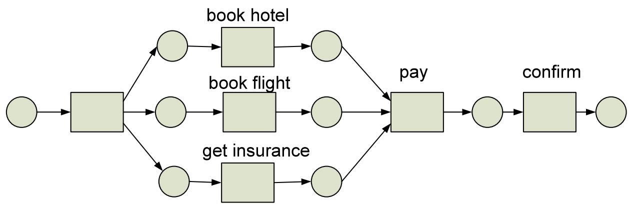


Fig. 1 A Petri net constructed from the log

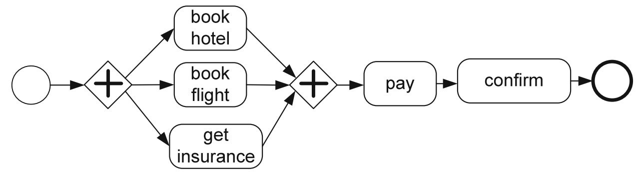


Fig. 2 A BPMN model obtained by a conversion from the Petri net

additional event properties, such as costs and resources (participants of the process); in this example, they are represented by prices and clients ip addresses.

To discover a control flow, an event log is represented as a multiset of traces, each of which is a sequence of events, corresponding to a concrete case identifier:

$$\begin{aligned} L = \Big[&\langle book\ flight, get\ insurance, book\ hotel, pay, confirm\rangle^5, \\ &\langle book\ flight, book\ hotel, get\ insurance, pay, confirm\rangle^4, \\ &\langle book\ hotel, book\ flight, get\ insurance, pay, confirm\rangle^4, \\ &\langle book\ hotel, get\ insurance, book\ flight, pay, confirm\rangle^3, \\ &\langle get\ insurance, book\ hotel, book\ flight, pay, confirm\rangle^1, \\ &\langle get\ insurance, book\ flight, book\ hotel, pay, confirm\rangle^1 \Big]. \end{aligned}$$

A *Petri net* discovered from L by the Alpha mining algorithm [12] is presented in Fig. 1.

With the help of a conversion algorithm, we construct a *BPMN model* from the Petri net, as shown in Fig. 2. This BPMN model is more compact than the initial Petri net. Thus, the result of process mining is available in a BPMN notation now; this BPMN model can be easily imported and executed by any of BPM tools mentioned above.

In order to estimate the advantages of using the BPMN notation for mining, we additionally *compare the complexity* of the models produced by the existing control flow discovery algorithms and the complexity of the corresponding BPMN models. We use the various metrics, such as the number of nodes, density and diameter for this evaluation. Correlations between the process metrics and the quality of process models are discussed in [13–16]. We present not only theoretical but also practical evaluations based on real-life event logs. Moreover, applied to these event logs, the metrics of the discovered BPMN models are compared to the metrics of the models designed manually with a BPMN-editor. This helps us to understand structural differences between models, which are created by process analysts and models discovered from event logs.

Since not only discovery but also *conformance checking* and *process enhancement* are essential steps in process mining, this paper also shows how to enable them for BPMN models. A BPMN model is converted to a Petri net, and then replay techniques are applied [17] to retrieve performance and conformance information. This information is used to enhance BPMN models. Theoretical observations presented in this paper help us to relate states of a BPMN model with the states of a corresponding Petri net. Thus, both conformance and performance information obtained for a Petri net can be visualized using the initial BPMN model.

A general scheme of using BPMN for process mining is presented in Fig. 3. The user discovers a BPMN model by applying discovery and BPMN conversions plugins. To show performance and conformance information and to annotate the BPMN diagram, the BPMN model is converted to a Petri net, such that replay techniques can be used.

The paper is organized as follows. Section 2 overviews related work. Section 3 introduces basic definitions and notions, including traces, Petri nets, system nets, (weak)

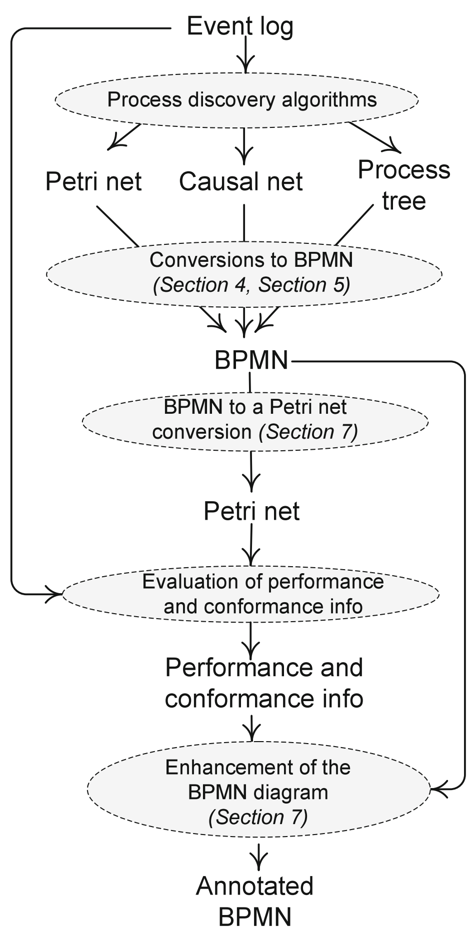


Fig. 3 A general scheme of using BPMN for process mining

simulation and (weak) bisimulation relations. In Sect. 4, we propose algorithms for conversion from well-known formalisms such as Petri nets to BPMN and prove correctness of these conversions. In Sect. 5, transformations of causal nets and process trees to BPMN are introduced. Section 6 contains a set of BPMN simplification rules. In Sect. 7, a technique for conformance checking and performance analysis on the basis of BPMN models is presented. A tool, which implements the proposed conversion and enhancement techniques, is presented in Sect. 8. Section 9 includes a *case study*, which shows the results of an application of the algorithms presented in the paper to real-life event logs. Also the structural business-processes metrics are calculated and presented in this section. Section 10 concludes the paper.

2 Related work

This section surveys previous work on Petri net and workflow graph conversions and includes an overview of existing discovery and conformance checking techniques dealing with "high-level" process models (e.g., BPMN models and models presented not only by the control flow, but also including data and resource perspectives as well).

Algorithms for conversion of free-choice workflow nets [18] (a special subset of Petri nets) to *workflow graphs* (generalization concept for process modeling notations such as BPMN, UML Activity [19], EPC [20,21], etc) were proposed in [22,23]. In our paper, we will deal with arbitrary Petri nets structures and arbitrary safe initial markings. Moreover, in contrast to the approaches proposed before [22,23], we prove that the target model will simulate (i.e., will be able to mimic) the behavior of the initial net and vice versa. These simulation relations give the possibility to prove important (in the context of process mining) propositions on the language relations. Another key result is that having simulation relations between initial and target models allows us to project various analysis data (such conformance and performance information) obtained for one model to another.

An approach for constructing BPMN models containing subprocesses was presented in [24]. In contrast to our paper, this technique is mainly focused on deriving subprocesses from event logs. We present basic robust conversion algorithms, which help to construct flat control flow skeletons of the target BPMN models from the discovered Petri nets and other low-level models. We hope that our approach can be further used as a basis for discovering more advanced constructs. As [24] our approach is generic and can work with multiple discovery algorithms (such as [12,25,26]) in the process mining context. The approach presented in [27] demonstrates the possibility of mining BPMN models covering both the control flow perspective and the resource perspective. Unfortunately, this approach is rather narrow in scope as it presents very concrete algorithms for mining BPMN control flow and resource elements. Algorithms for discovering data and resources from event logs were proposed in [28–30], respectively. To evaluate the quality of BPMN models and multiperspective process models obtained from the real event logs, an analysis of process quality metrics proposed in various studies, such as [13–16], can be applied. An approach for finding deviations between an event log and a multiperspective process model was presented in [31].

3 Preliminaries

In this section, we introduce basic notions, event logs, Petri nets, system nets and BPMN semantics.

Multisets are used to present states of Petri nets and BPMN models, and also they are used to define event logs, in which one trace can appear multiple times.

$\mathcal{B}(A)$ is the set of all multisets over some set A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. By $b = [{a_1}^2, {a_2}^3]$ we denote that elements $a_1, a_2 \in A$ appear in b two and three times, respectively.

The sum of two multisets b and b' over set A is defined as: $(b \uplus b')(a) = b(a) + b'(a)$ for all a from A. We say that $b \supseteq b'$ iff $\forall a \in A : b(a) \geq b'(a)$. For two multisets b and b' over set A, such that $b \supseteq b'$, the difference function is defined as: $(b \backslash b')(a) = b(a) - b'(a)$. The size of a multiset b over set A is denoted by $|b|$ and defined as $|b| = \sum_{a \in A} b(a)$. Sets will be considered as a special case of multisets, where each element can appear 0 or 1 times. Thus, operations applicable to multisets can be applied to sets.

Function $f : X \nrightarrow Y$ is a partial function with domain $dom(f) \subseteq X$ and range defined as $rng(f) = \{f(x) | x \in dom(f)\} \subseteq Y$. $f : X \rightarrow Y$ is a total function, i.e., $dom(f) = X$. Let $f : X \nrightarrow Y$ be a partial function, f can be applied to sequences of X using the recursive definition $f(\langle\rangle) = \langle\rangle$ and for some $\sigma \in X^*$ and $x \in X$ $f(\langle x \cdot \sigma\rangle) = \langle f(x)\rangle \cdot f(\sigma)$, if $x \in dom(f)$ and $f(\langle x \cdot \sigma\rangle) = f(\sigma)$ otherwise.

3.1 Event logs and Petri nets

Definition 1 (*Petri Net*) A *Petri net* is a tuple $PN = (P, T, F)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation.

Definition 2 (*Marking*) Let $PN = (P, T, F)$ be a Petri net. A *marking* M is a multiset of places, i.e., $M \in \mathcal{B}(P)$.

Definition 3 (*Safe Marking*) A marking M of a Petri net $PN = (P, T, F)$ is *safe* iff $\forall p \in P : M(p) \leq 1$, i.e., each place contains not more than 1 token.

Pictorially, places are represented by circles, transitions by boxes and the flow relation F by directed arcs. Places may carry tokens represented by filled circles. A current marking M is designated by putting $M(p)$ tokens into each place $p \in P$.

For a node $n \in P \cup T$, the set of *input nodes* and the set of *output nodes* are defined as $^\bullet n = \{x | (x, n) \in F\}$ and $n^\bullet = \{x | (n, x) \in F\}$, respectively.

A transition $t \in T$ is *enabled* in a marking M of net PN, denoted as $(PN, M)[t\rangle$, iff $\forall p \in {^\bullet t} : M(p) \geq 1$, i.e., each of its input places contains at least one token.

An enabled transition t may *fire*, i.e., one token is removed from each of the input places $^\bullet t$ and one token produced for each of the output places $t^\bullet$. Formally: $M' = (M \setminus {^\bullet t}) \uplus t^\bullet$ is the marking resulting from firing enabled transition t in marking M of Petri net PN. $(PN, M)[t\rangle(PN, M')$ denotes that t is enabled in M and firing results in marking M'.

Let $\sigma = \langle t_1, \ldots, t_n \rangle \in T^*$ be a sequence of transitions. $(PN, M)[\sigma\rangle(PN, M')$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$, such that $M_0 = M$, $M_n = M'$, and $(PN, M_i)\left[t_{i+1}\right\rangle(PN, M_{i+1})$ for $0 \leq i < n$. We say that M' is *reachable* from M if there exists σ, such that $(PN, M)[\sigma\rangle(PN, M')$.

$\mathcal{R}(PN, M)$ denotes the set of all markings reachable in PN from the marking M.

Definition 4 (*Labeled Petri Net*) A labeled Petri net $PN = (P, T, F, l)$ is a Petri net (P, T, F) with labeling function $l \in T \nrightarrow \mathcal{U}_A$ where $\mathcal{U}_A$ is some universe of activity labels. Let $\sigma_v = \langle a_1, \ldots, a_n \rangle \in \mathcal{U}_A{}^*$ be a sequence of activity labels. $(PN, M)[\sigma_v \triangleright (PN, M')$ iff there is a sequence $\sigma \in T^*$ such that $(PN, M)[\sigma\rangle(PN, M')$ and $l(\sigma) = \sigma_v$.

If $t \notin dom(l)$, transition t is called invisible. An occurrence of visible transition $t \in dom(l)$ corresponds to observable activity label l(t).

In the context of process mining, we normally consider so-called *complete* firing sequences, and thus we deal with processes, which have well-defined initial and end states. Therefore, let us give a notion of a system net.

Definition 5 (*System Net*) A system net is a triplet $SN = (PN, M_{init}, M_{final})$ where $PN = (P, T, F, l)$ is a labeled Petri net, $M_{init} \in \mathcal{B}(p)$ is the initial marking and $M_{final} \in \mathcal{B}(p)$ is the final marking.

Definition 6 (*Language of a System Net*) Suppose that $SN = (PN, M_{init}, M_{final})$ is a system net. Language L_{SN} of SN will be defined as a set of all visible execution sequences starting in M_{init} and ending in M_{final}, i.e., $L_{SN} = \left\{\sigma_v \mid (PN, M_{init})[\sigma_v \triangleright (PN, M_{final})\right\}$.

Event logs are considered as a starting point in the context of process mining, so let us give their formal definition.

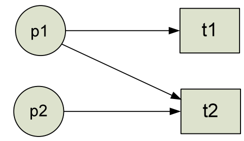


Fig. 4 A fragment of a non-free-choice Petri net

Definition 7 (*Trace, Event Log*) Let $A \subseteq \mathcal{U}_A$ be a set of activity labels. A trace $\sigma \in A^*$ is a sequence of activity labels. $L \in \mathcal{B}(A^*)$ is an event log, i.e., a multiset of traces.

Note that a trace can appear multiple times in an event log.

Some conversion techniques presented in this paper deal with free-choice nets. Let us define them.

Definition 8 (*Free-choice Nets*) A system net $SN = (PN, M_{init}, M_{final})$ and a corresponding labeled Petri net $PN = (P, T, F, l)$ are called *free-choice* iff for any two transitions $t_1, t_2 \in T$ with $^\bullet t_1 \cap {^\bullet t_2} \neq \emptyset$ holds $^\bullet t_1 = {^\bullet t_2}$.

To illustrate the concept of a free-choice Petri net, let us consider a fragment of a Petri net presented in Fig. 4. It is not a free-choice Petri net, and t_1 and t_2 share an input place, but do not have identical sets of input places. Indeed, if place p_1 contains tokens enabling t_1, while p_2 is empty, there is "no free choice" between t_1 and t_2, only t_1 is enabled and may fire. If an arc is added from p_2 to t_1, then the net is free-choice.

3.2 BPMN semantics

In this subsection, we will present an approach for the formalization of BPMN control flow semantics based on a concept of token. This formalization will give an ability to justify the conversion algorithms presented later in this paper. We restrict ourselves to the core set of BPMN elements, which includes activities, start and end events, exclusive and parallel gateways. We hope that these initial results will give a basis for formal description of more advanced BPMN modeling constructs.

Let us give a formal definition of a BPMN model.

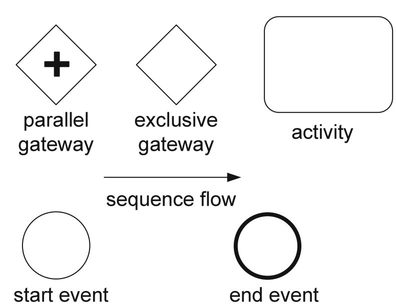


Fig. 5 Core BPMN modeling constructs

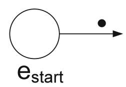


Fig. 6 Initial marking

Definition 9 (*BPMN Model*) A *BPMN model* is a tuple $BPMN_{model} = (N, A, G_{XOR}, G_{AND}, e_{start}, E_{end}, SF, \lambda)$, where

- N is a set of flow nodes,
- $A \subseteq N$ is a set of activities,
- $G_{XOR} \subseteq N$, $G_{AND} \subseteq N$ are sets of exclusive and parallel gateways,
- $e_{start} \in N$ is a start event,
- $E_{end} \subseteq N$ is a set of end events,
- sets $A, G_{XOR}, G_{AND}, \{e_{start}\}, E_{end}$ form a partition of N,
- $SF \subseteq N \times N$ is a set of sequence flows,
- $\lambda : N \nrightarrow \mathcal{U}_A$ is a labeling function, where $\mathcal{U}_A$ is some universe of activity labels,
- start event e_{start} does not have incoming sequence flows, and has not more than one outgoing sequence flow,
- end events from E_{end} do not have outgoing sequence flows.

Figure 5 shows the core BPMN constructs used to model processes.

Let $n \in N$ be an arbitrary BPMN model node, the *preset* $^{\bullet}n$ and the *postset* $n^{\bullet}$ are defined as sets of incoming and outgoing sequence flows for the node n, respectively.

To restrict the set of all possible BPMN models, we will consider and discover *well-formed* BPMN models, which are revealed as weakly connected graphs with a source and sink nodes.

Definition 10 (*Well-formed BPMN Model*) A BPMN model is called *well-formed* iff every node of this model is on a path from the start event to some end event.

Definition 11 (*BPMN Model Marking*) Let $BPMN_{model}$ be a BPMN model with a set of sequence flows SF. A *marking* M is a multiset over the set sequence flows, i.e., $M \in \mathcal{B}(SF)$. An *initial marking* M_{init} is a marking, such that for all sf from SF $M_{init}(sf) = 1$, if $sf \in e^{\bullet}_{start}$, and $M_{init}(sf) = 0$, otherwise.

An illustration for the initial marking is presented in Fig. 6.

Each node independently of its type may be *enabled*, and an enabled node may *fire*. Let us consider an arbitrary BPMN

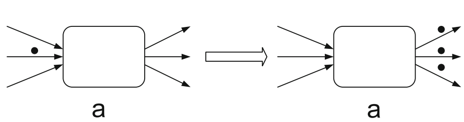


Fig. 7 Firing activity

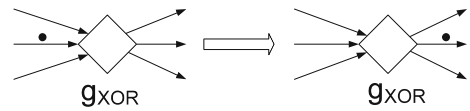


Fig. 8 Firing exclusive gateway

model $BPMN_{model} = (N, A, G_{XOR}, G_{AND}, e_{start}, E_{end}, SF, \lambda)$ and define its firing rules:

1. An activity $a \in A$ is *enabled* in a marking M iff $\exists sf \in SF : \left({}^{\bullet}a(sf) = 1\right) \wedge \left(M \supseteq \left[sf^1\right]\right)$. Suppose activity a is enabled and this activity may fire, producing a new marking M', such that $M' = M \setminus \left[sf^1\right] \uplus a^{\bullet}$. In other words, activity a is enabled in marking M iff it has an incoming sequence flow, which contains at least one token. When activity fires, it consumes one token from an incoming sequence flow and produces a token for each outgoing sequence flow (Fig. 7).
2. Exclusive gateways merge alternative paths: The incoming sequence flow token is routed to one of the outgoing sequence flows (Fig. 8).
   Similar to activities, exclusive gateway $g_{XOR} \in G_{XOR}$ is enabled in marking M iff there is an incoming sequence flow, which contains at least one token in marking M, i.e., $\exists sf \in SF : \left({}^{\bullet}g_{XOR}(sf) = 1\right) \wedge \left(M \supseteq \left[sf^1\right]\right)$. In contrast to activities, it produces a token to one of the outgoing sequence flows. Suppose an exclusive gateway g_{XOR} consumes a token from an incoming sequence flow sf and produces a token to an outgoing sequence flow sf', and then a new model marking M' will be defined as follows: $M' = M \setminus \left[sf^1\right] \uplus \left[sf'^1\right]$.
3. A parallel gateway $g_{AND} \in G_{AND}$ is enabled in marking M iff $\forall sf \in {}^{\bullet}g_{AND} : M(sf) \geq 1$, i.e., each incoming sequence flow contains at least one token. An enabled parallel gateway g_{AND} may fire and produce a new marking: M', such that $M' = M \setminus {}^{\bullet}g_{AND} \uplus g^{\bullet}_{AND}$, i.e., a parallel gateway consumes a token from each incoming sequence flow and produces a token to each outgoing sequence flow (Fig. 9).
4. The unique start event is never enabled, since it does not have any incoming sequence flow.
5. An end event $e_{end} \in E_{end}$ is enabled in marking M iff $\exists sf \in SF : \left(sf \in {}^{\bullet}e_{end}\right) \wedge \left(M(sf) \geq 1\right)$. When end event fires, it consumes a token from an incoming sequence flow sf, and yields in a new marking $M' = M \setminus \left[sf^1\right]$ (Fig. 10).

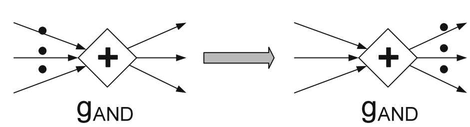


Fig. 9 Firing parallel gateway

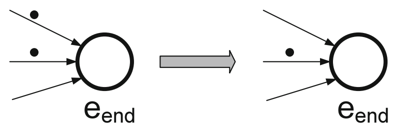


Fig. 10 Firing end event

When node $n \in N$ fires, we denote this firing as $(BPMN_{model}, M)\,[n\rangle\,(BPMN_{model}, M')$.

We write $(BPMN_{model}, M)\,[\sigma\rangle\,(BPMN_{model}, M')$ for some sequence of nodes $\sigma = \langle n_1, \ldots, n_k\rangle \in N^*$ iff there are markings $M_0, \ldots, M_k$, such that $M_0 = M$, $M_k = M'$, and for $0 \le i < k$ the following statement holds $(BPMN_{model}, M_i)\,\left[n_{i+1}\right\rangle\,(BPMN_{model}, M_{i+1})$.

Likewise in Petri nets marking M' is *reachable* from marking M iff there is a sequence $\sigma \in N^*$, such that $(BPMN_{model}, M)\,[\sigma\rangle\,(BPMN_{model}, M')$.

For some sequence of activity labels $\sigma_v \in \mathcal{U}_A^*$, we write $(BPMN_{model}, M)[\sigma_v \triangleright (BPMN_{model}, M')$, if there is σ, such that $(BPMN_{model}, M)\,[\sigma\rangle\,(BPMN_{model}, M')$ and $\lambda(\sigma) = \sigma_v$.

By $\mathcal{R}(BPMN_{model}, M)$, we will denote the set of all markings reachable in $BPMN_{model}$ from the marking M.

To define the notion of language generated by a BPMN model, let us first give a definition of a final marking.

Definition 12 (*Final BPMN Model Marking*) Let $BPMN_{model}$ be a BPMN model with an initial marking M_{init} and a set of nodes N. M_{final} is a *final marking* iff $M_{final} \in \mathcal{R}(BPMN_{model}, M_{init})$ and $\forall n \in N \nexists M' : (BPMN_{model}, M)\,[n\rangle\,(BPMN_{model}, M')$.

As it follows from this definition, the final marking of a BPMN model is the marking, in which no node can fire.

Definition 13 (*Language of a BPMN Model*) Let $BPMN_{model}$ be a BPMN model with an initial marking M_{init} and a set of final markings $\mathcal{M}_{final}$. The *language* of $BPMN_{model}$ is a set $L_{BPMN_{model}} = \{\sigma_v \mid (BPMN_{model}, M_{init})[\sigma_v \triangleright (BPMN_{model}, M) \wedge M \in \mathcal{M}_{final}\}$.

Thus, we define the language of a BPMN model as a set of all visible sequences starting in an initial marking and ending in some final marking.

According to the BPMN 2.0 specification [6], BPMN model gets the status *completed* iff there is no token remaining. Following the specification, a language of a BPMN model can be considered as a union of two disjoint sets: $L_{BPMN_{model}} = V_{BPMN_{model}} \cup I_{BPMN_{model}}$.

$V_{BPMN_{model}} = \{\sigma_v | ((BPMN_{model}, M_{init})[\sigma_v \triangleright BPMN_{model}, M)) \wedge (\forall sf \in SF : M(sf) = 0)\}$ is a set of *valid sequences*, corresponding to the model executions, which lead to markings with no tokens remaining. Note that according to BPMN semantics if no tokens remaining, no node is enabled.

$I_{BPMN_{model}} = L_{BPMN_{model}} \setminus V_{BPMN_{model}}$ stands for a set of *invalid sequences*, which are the traces of the BPMN model executions, stopped in markings with tokens on sequence flows. These sequences of activity labels correspond to the BPMN model deadlocks.

3.3 Transition systems, reachability graphs and simulation relations

In this subsection, some basic definitions, which are used for the justification of the conversion algorithms, will be given.

Definition 14 (*Transition system*) Let S and E be two disjoint non-empty sets of *states* and *events*, let $\tau \in E$ be a special *silent event*, and let $B \subseteq S \times E \times S$ be a *transition relation*. A *transition system* is a tuple $TS = (S, E, B, s_{in})$, where $s_{in} \in S$ is an initial state. Elements of B are called *transitions*.

We write $s \xrightarrow{e} s'$, when $(s, e, s') \in B$. Assume that $\forall s \in S : s \xrightarrow{\tau} s$, i.e., there is a transition from every state to itself, labeled by τ.

A state s is *reachable* from a state s' iff there is a (possibly empty) sequence of transitions leading from s to s' (denoted by $s \xrightarrow{*} s'$). The reflexive transitive closure of $\xrightarrow{\tau}$ will be denoted as $\Rightarrow$. By $s \overset{e}{\Rightarrow} s'$, we denote $s \Rightarrow s'' \xrightarrow{e} s''' \Rightarrow s'$, i.e., s' can be reached from s via e preceded and followed by zero or more τ transitions.

A transition system must satisfy the following basic axiom: Every state is reachable from the initial state: $\forall s \in S : s_{in} \xrightarrow{*} s$.

Definition 15 (*Simulation*) For transition systems: $TS = (S, E, B, s_{in})$ and $TS' = (S', E, B', s'_{in})$ relation $R \subseteq S \times S'$ is called a *simulation* iff:

- $(s_{in}, s'_{in}) \in R$,
- $\forall (u, v) \in R\ \forall e \in E$: if $\exists u' : u \xrightarrow{e} u'$ then $\exists v' : v \xrightarrow{e} v'$ and $(u', v') \in R$.

Definition 16 (*Weak simulation*) Let us consider two transition systems: $TS = (S, E, B, s_{in})$ and $TS' = (S', E, B', s'_{in})$. Relation $R \subseteq S \times S'$ is called a *weak simulation* iff:

- $(s_{in}, s'_{in}) \in R$,
- $\forall (u, v) \in R\ \forall e \in E$: if $\exists u' : u \xrightarrow{e} u'$ then $\exists v' : v \overset{e}{\Rightarrow} v'$ and $(u', v') \in R$.

Definition 17 (*Bisimulation*) If R is a *(weak) simulation* relation and R^{-1} is a *(weak) simulation* relation as well, then relation R is called a *(weak) bisimulation*.

Definition 18 (*Reachability Graph for a System Net*) A *reachability graph* for a system net $SN = (PN, M_{init}, M_{final})$, where $PN = (P, T, F, l)$, and $l \in T \nrightarrow \mathcal{U}_A$, is a transition system $TS = (S, E, B, s_{in})$, such that:

- $S = \mathcal{R}(SN, M_{init})$, i.e., the set of states is defined as a set of markings reachable from M_{init},
- $E = rng(l) \cup \{\tau\}$, i.e., the set of events is defined as a union of the range of l and a silent event τ,
- B contains a transition (M, e, M') iff at least one the following conditions holds:
  - $\exists t \in T : (SN, M)[t\rangle(SN, M')$, such that $l(t) = e$, if $t \in dom(l)$, or $e = \tau$, otherwise,
  - $M = M'$ and $e = \tau$, this holds for silent transitions from states to itself.
- $s_{in} = M_{init}$, i.e., the initial state in TS is the initial marking of SN.

Definition 19 (*Reachability graph for a BPMN model*) A *reachability graph* for a BPMN model $BPMN_{model} = (N, SF, A, G_{XOR}, G_{AND}, e_{start}, E_{end}, \lambda)$ with an initial marking M_{init} is defined as a transition system $TS = (S, E, B, s_{in})$, such that:

- $S = \mathcal{R}(BPMN_{model}, M_{init})$,
- $E = rng(\lambda) \cup \{\tau\}$, where τ is a special silent event,
- $(M, e, M') \in B$ iff at least one of the following conditions holds:
  - there exists $n \in N$, such that $(BPMN_{model}, M)[n\rangle(BPMN_{model}, M')$, where $\lambda(n) = e$, if $n \in dom(\lambda)$, or $e = \tau$, otherwise,
  - $M = M'$ and $e = \tau$.
- $s_{in} = M_{init}$.

4 Converting process models into BPMN

In this section, we will propose algorithms for the conversion from well-known formalisms such as Petri nets, causal nets and process trees to BPMN. These formalisms are widely used within process mining as results of application of process discovery algorithms [12,25,26,32–35]. Having conversion algorithms to BPMN format will give an opportunity to discover control flow models, which could be integrated with other process perspectives. The correctness of the proposed system nets conversion algorithm will be proven.

First, let us show that every system net with a safe initial marking can be transformed to an equivalent system net, which contains a unique source place.

4.1 Adding a source place to an arbitrary system net with a safe initial marking

In most cases, models discovered from event logs are arbitrary system nets with safe initial markings. We start with transforming of a system net with a safe initial marking into a system net, which contains a unique source place and does not contain hanging places (places without outgoing arcs). In the next subsections, we will show algorithms for conversion of such nets to BPMN.

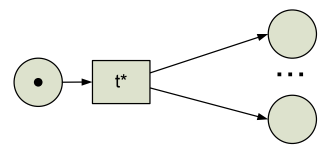


Fig. 11 Creating a source place

Algorithm 1 [Adding a source place to a system net].
Input: A system net $SN = (PN, M_{init}, M_{final})$, where $PN = (P, T, F, l)$, such that $\forall p \in P : M_{init}(p) \leq 1$.

Step 0: Adding a source place. Add a novel place $i \in P$, a novel initial transition t^* (note that t^* does not have a label, since $t^* \notin dom(l)$) and connect them with an arc (i, t^*). For each place $p \in P$, such that $M_{init}(p) = 1$, add an arc (t^*, p). This step is presented in Fig. 11.

Step 1: Handling unconnected transitions. For each transition $t \in T$, such that $^\bullet t = \emptyset$, add a place p, connected with t by an incoming and outgoing arc. Add an arc from the initial transition t^* to the place p (Fig. 12).

Step 2: Removing dead places. Remove each place $p \in P$ and transitions from $p^\bullet$ along with incident arcs, if there is no path from i to p. Repeat Step 2 until there are no more dead places.

Step 3: Removing hanging places. Remove all places $p \in P$, such that $|p^\bullet| = 0$, along with incident arcs.

Step 4: Constructing novel markings. Suppose that P' is the resulting set of places, and $P^* \subseteq P'$ is the set of places added at Step 1. Then the initial and final markings M'_{init} and M'_{final} are defined as follows: For all $p \in P'$, such that $p \neq i$, $M'_{init}(p) = 0$, $M'_{init}(i) = 1$, for all $p \in P^*$ holds that $M'_{final}(p) = 1$, and for all $p \in (P \cap P')$ the number of tokens is preserved, i.e., $M'_{final}(p) = M_{final}(p)$. The source place

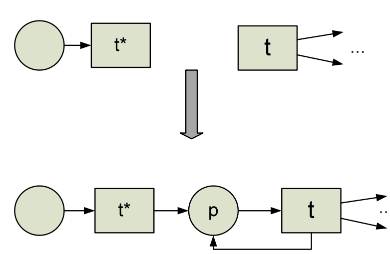


Fig. 12 Handling unconnected transitions

does not contain any tokens in the final marking, i.e., $M'_{final}(i) = 0$.

Output: A system net $SN' = (PN', M'_{init}, M'_{final})$, where $PN' = (P', T', F', l)$ is defined on the basis of $PN = (P, T, F, l)$ at Steps 0–3. Markings M'_{init} and M'_{final} are defined at Step 4.

Proposition 1 *Let* $SN = (PN, M_{init}, M_{final})$ *be a system net and* $SN' = (PN', M'_{init}, M_{final})$*, where* $PN' = (P', T', F', l)$*, be a result of applying Algorithm 1 to SN. Let* $i \in P'$ *be a source place constructed by Algorithm 1. Then for each node* $n \in (P' \cup T')$ *exists a path from* i *to* n.

Proof Suppose that $n \in P'$. Since all the places, to which there were no paths from i, were deleted at the Step 2, there exists a path from i to n. If $n \in T'$, then either n did not have incoming arcs and was connected with i at the Step 1, or either it is connected by an incoming arc with a place, and for this place there is a path from i, and hence there is a path from i to n. □

Note that places, which were added at Step 1, contain tokens in any reachable marking.

Algorithm 1 transforms a system net with a safe initial marking to an *equivalent* system net with a source place and no hanging places. More formally, there is a *weak bisimulation* relation between reachability graphs of the initial and the target system nets. The proof is straightforward according to Definition 17. Further we will consider only system nets with unique source places and without hanging places and call them just system nets.

4.2 Free-choice system nets to BPMN conversion

In this subsection, an algorithm for conversion from a free-choice system net to a BPMN model will be presented.

The conversion algorithm will be illustrated by a running example: a system net, which defines a booking process

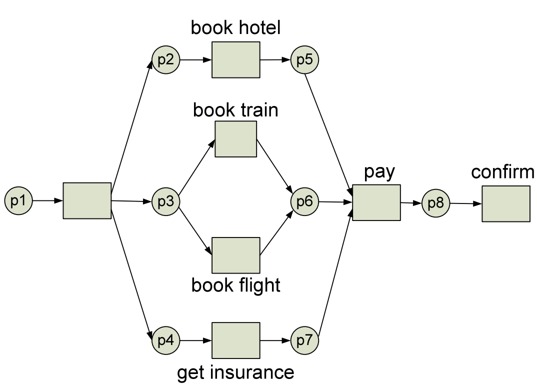


Fig. 13 A system net of a booking process

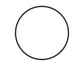


Fig. 14 An initial BPMN model

(Fig. 13), will be converted to an equivalent BPMN model. The source place is $p1$, the final marking M_{final} is the marking, such that $M_{final}(p) = 0$ for all p.

Note that in contrast to the booking model presented earlier (Fig. 1), this model contains a choice construction (the user books a flight or a train ticket), also note that there is a transition used as a splitting node, and this transition does not have a label.

Algorithm 2 [Constructing a BPMN model for a system net]. Input: A free-choice system net SN, where $SN = (PN, M_{init}, M_{final})$, $PN = (P, T, F, l)$, and i is a source place.

Step 0: Initializing BPMN model.
Determine a BPMN model $BPMN_{model} = (N, A, G_{XOR}, G_{AND}, e_{start}, E_{end}, SF, \lambda)$, which contains a start event only, i.e., $N = \{e_{start}\}$, $SF = \emptyset$, $A = \emptyset$, $G_{XOR} = \emptyset$, $G_{AND} = \emptyset$, and $E_{end} = \emptyset$ (Fig. 14).

Step 1: Converting transitions.
Create a BPMN model activity $a \in A$ for each transition $t \in T$ and determine the corresponding bijective *mapping* function $M : T \to A$. The labeling function λ is defined as follows $\lambda(M(t)) = l(t)$, for all t from $dom(l)$. If there exists a transition $t \in T$, such that $|t^{\bullet}| > 1$, i.e., t has more than one outgoing arc, add a parallel gateway g_{AND} and a sequence flow $(M(t), g_{AND})$. $BPMN_{model}$ with activities and a parallel gateway added is shown in Fig. 15.

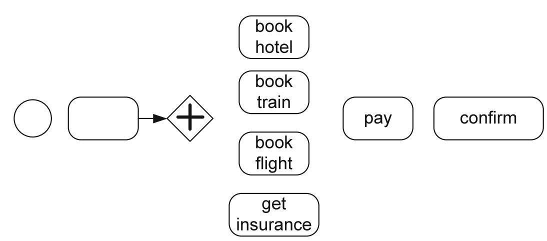


Fig. 15 Adding activities and parallel gateways to the BPMN model of a booking process

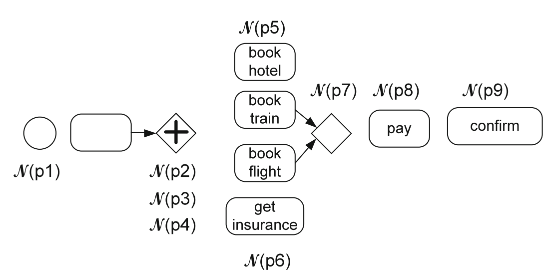


Fig. 16 Identifying place nodes

Step 2: Converting places.

In this step, each place $p \in P$ is converted to BPMN routing constructs, identifying a corresponding *place node* and a corresponding *place flow* within $BPMN_{model}$. During the BPMN model construction, we will define functions, which map places from P to corresponding *place nodes* and *place flows* for $BPMN_{model}$, and denote them as $\mathcal{N} : P \to N$ and $\mathcal{F} : P \to SF$, where N and SF are the sets of $BPMN_{model}$ nodes and sequence flows, respectively. The function $\mathcal{N}$ will be used to define nodes, which correspond to places, and used for establishing connections within the target model. The function $\mathcal{F}$ will be used to show the relations between places and sequence flows, and will help to relate a system net and a BPMN model markings.

Step 2.1: Connecting to inputs. Let us transform places and identify *place nodes*, taking into account presets:

- If $|{}^\bullet p| = 0$ (p does not have incoming arcs), then place p is a source place of SN, and the *place node* will be defined as e_{start}, i.e, $\mathcal{N}(p) = e_{start}$.
- If $|{}^\bullet p| = 1$, i.e., there exists one and only one transition $t \in T$ connected with p by an outgoing arc. If there exists $g_{AND} \in G_{AND}$, such that $(M(t), g_{AND}) \in SF$, then the *place node* is set to g_{AND}: $\mathcal{N}(p) = g_{AND}$, otherwise $\mathcal{N}(p) = M(t)$.
- If $|{}^\bullet p| > 1$ (there is more than one transition connected with p by outgoing arc), then an exclusive gateway g_{XOR} is added to G_{XOR} and for each transition t from ${}^\bullet p$ a sequence flow is added to SF. If there exists $g_{AND} \in G_{AND}$, such that $(M(t), g_{AND}) \in SF$, this sequence flow is defined as (g_{AND}, g_{XOR}), otherwise the sequence flow is $(M(t), g_{XOR})$. The exclusive gateway g_{XOR} is set as the *place node* for p, i.e., $\mathcal{N}(p) = g_{XOR}$.

The result of applying Step 2.1 to the booking process is shown in Fig. 16. For each place of the initial system net, a corresponding *place node* is specified.

Step 2.2: Merging places with coinciding postsets: For all maximum sets of places $\{p_1, \ldots, p_n\} \subseteq P$ with coinciding non-empty postsets ($p_1^\bullet = \ldots = p_n^\bullet$)[2], such that $n \geq 2$, an additional parallel gateway g_{AND} is added to G_{AND}. This gateway is connected by incoming sequence flows with all the corresponding *place nodes*, i.e., sequence flows $(\mathcal{N}(p_1), g_{AND})$, ..., $(\mathcal{N}(p_n), g_{AND})$ are added to SF and are defined as place flows: for all s_i from $\{s_1, \ldots, s_n\}$, $\mathcal{F}(s_i) = (\mathcal{N}(p_i), g_{AND})$. After that the parallel gateway g_{AND} is considered to be a novel *place node* for places $p_1,\ldots,p_n$, i.e., $\mathcal{N}(p_1) = g_{AND}$, ..., $\mathcal{N}(p_n) = g_{AND}$. Fig. 17 shows the result of applying the places merge procedure to the booking process presented in Fig. 16.

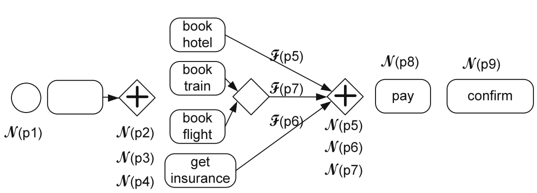


Fig. 17 Merging places with coinciding postsets

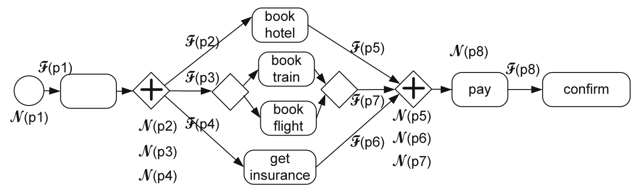


Fig. 18 The resulting BPMN model

Step 2.3: Connecting to outputs. In this step for each group of places $p_1,\ldots, p_n$ with coinciding postsets: $post = p_1^\bullet = \ldots = p_n^\bullet$[3] corresponding *place nodes*: $\mathcal{N}(p_1), \ldots, \mathcal{N}(p_n)$ are connected by outgoing sequence flows with other BPMN model elements.

- If $|post| = 1$, i.e., there is only one transition $t \in T$ connected with $p_1,\ldots,p_n$ by incoming arcs, then sequence flow $(\mathcal{N}, M(t))$, where $\mathcal{N} = \mathcal{N}(p_1) = \ldots = \mathcal{N}(p_n)$, is added to SF. If the group of places with coinciding postsets contains only one node (let this node be p_1), then $\mathcal{F}(p_1) = (\mathcal{N}, M(t))$.
- If $|post| > 1$, an exclusive gateway g_{XOR} and a sequence flow $(\mathcal{N}, g_{XOR})$ are added to G_{XOR} and SF, respectively.[4] Then for each t from $post$ a sequence flow $(g_{XOR}, M(t))$ is added to SF. If $n = 1$, $\mathcal{F}(p_1) = (\mathcal{N}, g_{XOR})$.

The resulting BPMN model is shown in Fig. 18.

Output: $BPMN_{model}$ and mappings: $M, \mathcal{N}, \mathcal{F}$.

[2] Note that due to the free-choice structure of PN, postsets either coincide or do not intersect.

[3] Note that we consider system nets without hanging places.

[4] All the places have the same place node $\mathcal{N}$, obtained on the previous step of the algorithm.

4.3 Non-free-choice system nets to BPMN

Often non-free-choice Petri nets, which allow more behavior than free-choice Petri nets, are obtained as a result of applying process discovery algorithms. In this subsection, we will introduce an algorithm for constructing free-choice Petri nets from Petri nets with non-free-choice constructions. This algorithm works as follows: for each arc, which produces a non-free-choice construction, do the transformation presented in Fig. 19. A more formal description of the algorithm is presented below.

Algorithm 3 [Constructing a free-choice Petri net from an arbitrary Petri net]. Input: A labeled Petri net $PN = (P, T, F, l)$.

For each arc (p^*, t), $p^* \in P$, $t \in T$, such that $\exists t' \in T : p^* \in (^\bullet t \cap {}^\bullet t')$ and $\exists p' \in P : p' \in {}^\bullet t, p' \notin {}^\bullet t'$ do the following transformation: remove flow (p^*, t), add transition t'', place p'', and connecting sequence flows: $(p^*, t''), (t'', p''), (p'', t)$ (see Fig. 19). The labeling function l is not defined for t'', i.e., $t'' \notin dom(l)$.
Output: Labeled Petri net $PN' = (P \cup \{p''\}, T \cup \{t''\}, F \cup \{(p^*, t''), (t'', p''), (p'', t)\}, l)$.

The algorithm can be applied iteratively and arcs can be considered in any order, since each transformation does not change the set of arcs, which have to be replaced.

4.4 System nets conversions justification

This subsection presents justifications of the system nets conversion algorithms.

Let us prove that Algorithm 2 preserves structural and some behavioral properties of a process model.

Lemma 1 *Let SN, where $SN = (PN, M_{init}, M_{final})$, and $PN = (P, T, F, l)$, be a free-choice system net with a source place i. Let $BPMN_{model}$ be a result of applying Algorithm 2 to SN. Suppose that $M : T \to A$ is a mapping function obtained during an execution of Algorithm 2. Suppose also that $\mathcal{N} : P \to N$ is a function, which defines* place nodes *in $BPMN_{model}$. Then for any two places $p_1, p_2 \in P$, such that $\exists t \in T : (t \in p_1^\bullet) \wedge (t \in {}^\bullet p_2)$ (Fig. 20), there are paths from $\mathcal{N}(p_1)$ to $M(t)$ and from $M(t)$ to $\mathcal{N}(p_2)$ within $BPMN_{model}$.*

Proof According to the Algorithm 2, node $\mathcal{N}(p_1)$ is either directly connected with $M(t)$ or directly connected with its immediate predecessor—an exclusive gateway (see Step 2.3). Hence there is a path from $\mathcal{N}(p_1)$ to $M(t)$. Now let us consider $\mathcal{N}(p_2)$. This node is either $M(t)$ activity or a gateway, such that there is a path from $M(t)$ to this gateway (see Steps 2.1, 2.2). This implies that there is a path from $M(t)$ to $\mathcal{N}(p_2)$ within $BPMN_{model}$. □

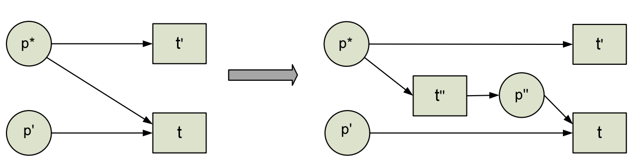


Fig. 19 Converting non-free-choice Petri nets into free-choice Petri nets

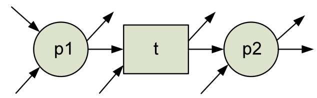


Fig. 20 Connected places

Lemma 2 *Suppose that SN is a system net with a source place i. Then the result of applying Algorithm 2 is $BPMN_{model} = (N, SF, A, G_{XOR}, G_{AND}, e_{start}, E_{end}, \lambda)$, such that for each node there is a path from e_{start} to this node.*

Proof Source place i is converted to the start event e_{start}. It inductively follows from Lemma 1 that all the *place nodes* and all the activities A are located on paths from e_{start}. All other BPMN model nodes are gateways G_{XOR}, G_{AND} and end events E_{end}, which lie on paths from *place nodes* to activities or from activities to the *place nodes* by the construction; consequently, they are also located on paths from e_{start}. □

Theorem 1 (Well-formedness) *Every system net with a safe initial marking can be converted to a* well-formed *BPMN model.*

Proof Algorithm 3 allows to construct free-choice system nets from non-free-choice system nets, preserving nodes connectivity. Proposition 1 shows that an arbitrary system net with a safe initial marking can be converted to an equivalent system net, which has a unique source place, such that for every node of this net there is a path from the source place to this node. Lemma 2 allows us to construct a BPMN model, where for each node there is a path from the start event to this node. According to Algorithm 2, the only possible hanging nodes in the target BPMN model are activities. Thus, additional end events can be added to the BPMN model and connected by incoming sequence flows with activities, making all the nodes be on paths from a start event to end events. □

Note that end events consume tokens from incoming sequence flows, and thus the global execution order of BPMN model will not be changed. Since end events do not change the global execution order, further we will prove some propositions for model conversions, which do not involve addition of end events.

Now let us discuss the behavioral relation between initial system net and the BPMN model generated by Algorithm 2. We will show that each firing of a Petri net corresponds to a sequence of the BPMN model firings.

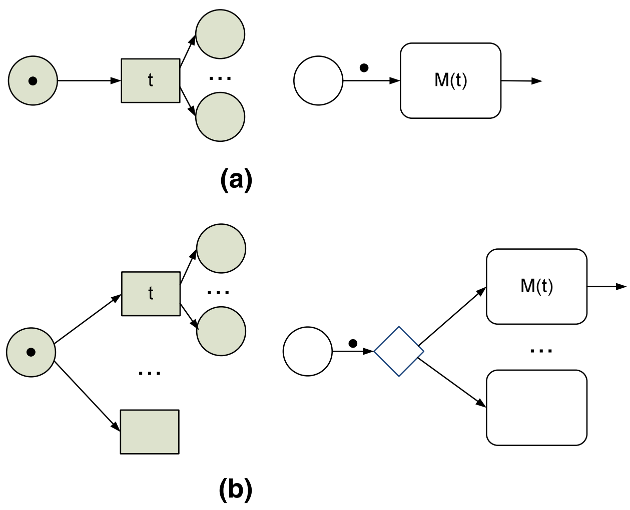


Fig. 21 Initial markings

Theorem 2 (Weak similarity) *Let SN be a free-choice system net with a source place i, where* $SN = (PN, M_{init}, M_{final})$, $PN = (P, T, F, l)$. *Let* $BPMN_{model}$ *be a result of applying Algorithm 2 to SN,* $M : T \rightarrow A$ *is the mapping function. Let* $TS = (S, E, B, s_{in})$, $TS' = (S', E, B', s'_{in})$ *be reachability graphs of SN and* $BPMN_{model}$*, respectively. There exist* weak simulation *relations* R *and* R' *from* TS *to* TS' *and from* TS' *to* TS*, respectively, such that:*

1. $(u, v) \in R$ *iff* $\forall p \in P : u(p) = v(\mathcal{F}(p))$,
2. *if* $(u, v) \in R$ *then* $(v, u) \in R'$,
3. $\forall v \in S' \exists v' \in S' : (v \xrightarrow{*} v') \wedge (\exists u' \in S : (u', v') \in R)$. *In other words, from each state* $v \in S'$ *it is always possible to reach some state* $v' \in S'$*, which is in the relation* R *with some state* $u' \in S$.

But it is not guaranteed that a weak bisimulation relation exists.

Proof Let us prove the existence of *weak simulation* relations R and R' between TS and TS' inductively on pairs of states $u \in S$, $v \in S'$ such that $\forall p \in P : u(p) = v(\mathcal{F}(p))$.

Induction basis.

1. Pairs (s_{in}, s'_{in}) and (s'_{in}, s_{in}) belong to R and R', respectively, by the definition of a *weak simulation* relation. Both variants for initial markings of *SN* and $BPMN_{model}$ are presented in Fig. 21. Tokens in Fig. 21 are represented by black dots. As can be seen $\forall p \in P : s_{in}(p) = s'_{in}(\mathcal{F}(p))$. For the proof of condition 3. see the *Induction step*.
2. Let us prove that there is no weak bisimulation relation between TS and TS'. Suppose there is such a relation R'' (Fig. 22), then by the definition $(s'_{in}, s_{in}) \in R''$.

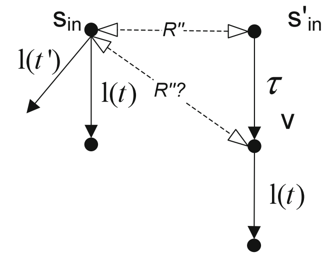


Fig. 22 Construction of a *weak bisimulation* relation

For variant **b.** it holds that exists $v \in S'$, such that $s'_{in} \xrightarrow{\tau} v$, and $M(t)$ is enabled in v. The only state in *TS*, to which there is a transition from s_{in} labeled by τ, is s_{in} itself, thus $(s_{in}, v) \in R''$. State s_{in} has at least one outgoing transition labeled with $l(t')$, such that $t' \neq t$. Suppose that $l(t) \neq l(t')$, then we get a contradiction, since v does not have an outgoing transition labeled by $l(t')$.

Induction step.

1. Now let us consider state $u \in S$ (Fig. 23)
   By the induction hypothesis, there exists a state v in TS' (a marking of $BPMN_{model}$), such that $(u, v) \in R$ (Fig. 24). Furthermore, by the induction hypothesis the following condition holds: $\forall p \in P : u(p) = v(\mathcal{F}(p))$, i.e., each place and its corresponding *place flow* contain the same number of tokens. Note that more than one token can appear in a place.
   Now let us show that if *TS* has a transition from state u, TS' has a corresponding sequence of transitions from state v, i.e., $\exists v''' : v \overset{l(t)}{\Longrightarrow} v'''$ and $\forall p \in P : u'(p) = v'''(\mathcal{F}(p))$. Thus, (u', v''') will belong to R. State v of $BPMN_{model}$ is presented in Fig. 25.
   Note that we consider the most general case, which represents all the routing elements. The remaining cases can be treated similarly. The gateway g_1 is enabled in marking v (by the construction, since t is enabled in u) and can fire, producing a novel marking, in which firing g_2 yields $M(t)$ being enabled. Let us call the marking, in which

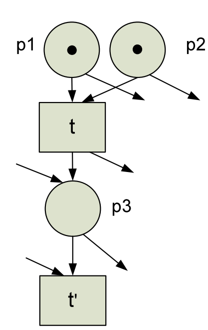


Fig. 23 Marking u of a system net

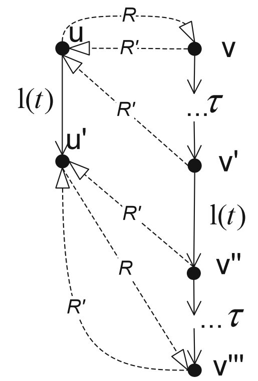


Fig. 24 Current states in transition systems

$M(t)$ is enabled, u', then $u \Rightarrow u'$. After $M(t)$ fires, some marking v'' is reached: $v' \overset{l(t)}{\rightarrow} v''$. Starting from marking v'' firings of gateways lead to adding a novel token to each *place flow*, which corresponds to some place from $t^{\bullet}$, and producing marking v''': $v'' \Rightarrow v'''$, i.e., $v \overset{l(t)}{\Longrightarrow} v'''$. Note that $\forall p \in P$, holds that:

$$v'''(\mathcal{F}(p)) = \begin{cases} v(\mathcal{F}(p)) - 1, & \text{if } p \in {}^{\bullet}t,\ p \notin t^{\bullet}, \\ v(\mathcal{F}(p)) + 1, & \text{if } p \in t^{\bullet},\ p \notin {}^{\bullet}t, \\ v(\mathcal{F}(p)), & \text{otherwise.} \end{cases}$$

Initial conditions $u' = u - {}^{\bullet}t + t^{\bullet}$ and $\forall p \in P : v(\mathcal{F}(p)) = u(p)$ allow to conclude that $\forall p \in P : u'(p) = v'''(\mathcal{F}(p))$.

2. Let us consider state $v \in S'$. By the induction hypothesis $\exists u \in S' : ((v, u) \in R') \wedge (\forall p \in P : v(\mathcal{F}(p)) = u(p))$. Two variants are possible either v does not have outgoing transitions and then $v \Rightarrow v$ and all the theorem conditions hold. Or there is a set of states V', such that $\forall v' \in V' : v \Rightarrow v'$, i.e., all $v' \in V'$ are reachable from v by τ transitions. In this case, there is a state v_1', such that $v \Rightarrow v_1'$ and $M(t)$ is enabled in v_1'. Transition t is enabled in u by the induction hypothesis and by the construction. Pair (v'', u') belongs to R', where $v_1' \overset{l(t)}{\rightarrow} v''$ and $u \overset{l(t)}{\rightarrow} u'$. Let us denote the set of states v^*, such that $v'' \Rightarrow v^*$, as V^*, and (v^*, u') belongs to R' for $\forall v^* \in V^*$. The state $v''' \in TS'$, such that $\forall p \in P : v'''(\mathcal{F}(p)) = u'(p)$, is reachable from any state in V^* by the construction. Thus, the induction hypothesis and the condition 3 are proven.

Thus, it was shown that there are *weak simulation* relations between TS and TS', and conditions 1–3. are hold. In the *induction basis*, it was shown that there is no *weak bisimulation* relation between TS and TS'. □

This theorem has an important corollary within the process mining context: The conversion algorithm allows to preserve the language of a free-choice system net under some assumption.

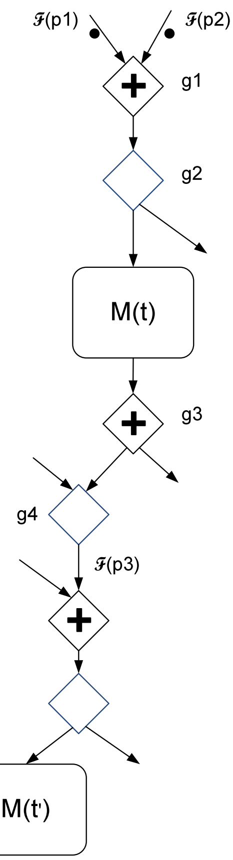


Fig. 25 Marking v of a BPMN model

Corollary 1 (Language equivalence for free-choice system net) *Suppose there is a free-choice system net SN, where* $SN = (PN, M_{init}, M_{final})$*, and i is a source place. Suppose also that* M_{final} *is the only reachable marking, in which no transition enabled, i.e., if* $M \in \mathcal{R}(PN, M_{init})$ *then* $M \neq M_{final}$ *iff* $\exists t \exists M' : (PN, M)\,[t\rangle\,(PN, M')$*. Let* $BPMN_{model}$ *be a result of applying Algorithm 2 to SN, then* $L_{SN} = L_{BPMN_{model}}$.

Proof 1. Let is consider a trace σ_v, such that $(PN, M_{init})\,[\sigma_v \rhd (PN, M_{final})$, i.e., $\sigma_v \in L_{SN}$. There is a weak simulation relation $R \subseteq (S \times S)$ from TS to TS', where TS and TS' are reachability graphs for SN and $BPMN_{model}$, respectively. Thus, σ_v can be replayed in $BPMN_{model}$, and after the replay $BPMN_{model}$ will be in a marking M, such that $(M_{final}, M) \in R$. If $\exists n \exists M' (BPMN_{model}, M)\,[n\rangle\,(BPMN_{model}, M')$, then since $\forall p \in P : M_{final}(p) = M(\mathcal{F}(p))$, $\exists M'' (PN, M_{final})\,[t\rangle\,(PN, M'')$, we get a contradiction. Thus, $L_{SN} \subseteq L_{BPMN_{model}}$.

2. Now let us prove that $L_{BPMN_{model}}$ do not contain traces, which do not belong to L_{SN}. Suppose there is a trace $\sigma_v \in L_{BPMN_{model}}$, such that $(BPMN_{model}, M'_{init})[\sigma_v$

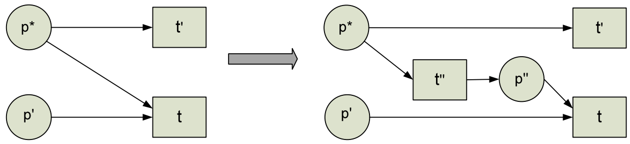


Fig. 26 Splitting non-free-choice construction

$\triangleright (BPMN_{model}, M)$, where M'_{init} is an initial marking of $BPMN_{model}$. Theorem 2 states that there exists $BPMN_{model}$ marking M', such that $(M \Rightarrow M') \wedge (\exists M'' : (M'', M') \in R)$, where M'' is a marking of *SN*. By the definition of BPMN model language, no node can fire at the marking M, thus $M = M'$, and $(M'', M) \in R$. We get that M'' is also a state (*SN* marking) without outgoing transitions, otherwise M is not a final marking of a $BPMN_{model}$, since $(M'', M) \in R$. Thus, σ_v : $(SN, M_{init})[\sigma_v \triangleright (SN, M'')$, and $\sigma_v \in L_{SN}$. □

Now let us compare behavioral properties of non-free-choice Petri nets and corresponding free-choice Petri nets constructed by Algorithm 3.

Theorem 3 (Non-free-choice Petri nets conversion) *Let $PN = (P, T, F, l)$ be an arbitrary labeled Petri net, and $PN' = (P', T', F', l)$ be a result of applying Algorithm 3 to PN. Let $TS = (S, E, B, s_{in})$ and $TS' = (S', E, B', s'_{in})$ be reachability graphs of PN and PN', respectively. Then there are weak simulation relations from TS to TS', and from TS' to TS. But it is not guaranteed that a weak bisimulation relation exists.*

Proof Let us define *weak simulation* relations R and R' between *TS* and *TS'* in such a way that for every two states $s \in S$ and $s' \in S'$ if $\forall p \in P : s(p) = s'(p)$, then (s, s') belongs to R and (s', s) belongs to R'. Let us consider a place $p^* \in P$ (Fig. 26), such that $\exists t, t' \in T : p^* \in (^\bullet t \cap {}^\bullet t')$ and $\exists p' \in P : p' \in {}^\bullet t', p' \notin {}^\bullet t$.

For this place, the output flow will be modified according to Algorithm 3. Let us consider u - a marking of *PN*: $u(p^*) \geq 1$ and construct fragments of reachability graphs for *PN* and *PN'*, containing the marking u and a corresponding marking of *PN'* - $v : \forall p \in P : v(p) = u(p)$ (Fig. 27).

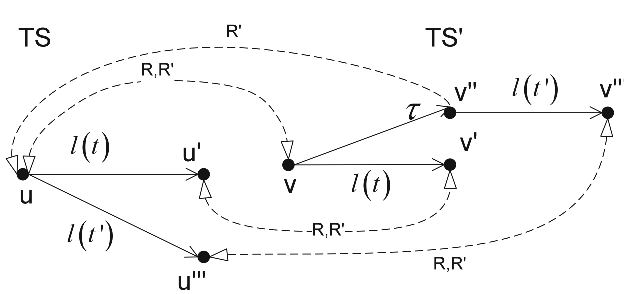


Fig. 27 Simulation of non-free-choice net by the corresponding free-choice net

Suppose that t is enabled in u, and u' is a state (*PN* marking) obtained after firing of transition t: $u \xrightarrow{l(t)} u'$, then t is also enabled in v, and v' is a marking of *PN'*, such that $v \xrightarrow{l(t)} v'$, then $\forall p \in P : v'(p) = u'(p)$, and $(u', v') \in R$, $(v', u') \in R$. Now suppose that t' is enabled in u and can fire producing a novel marking u'''. For *TS'*, there is a corresponding state v''', such that $v \xrightarrow{\tau} v'' \xrightarrow{l(t')} v'''$, and $\forall p \in P : v'''(p) = u'''(p)$. Pair (u''', v''') will belong to R, pairs (v''', u'''), (v'', u) will belong to R'. Note that the state u can simulate v'', since $\forall p \in P, p \neq p^* : u(p) = v''(p)$, and thus, it includes behavior allowed in v''. The procedure of defining R and R' can be considered for each transformation of *PN*, and thus *weak simulation* relations between *TS* and *TS'* can be derived.

There is no bisimulation relation, since there is no state, which bisimulates v''. □

Corollary 2 (Language of non-free-choice system net) *Let $SN = (PN, M_{init}, M_{final})$ be an arbitrary system net, where $PN = (P, T, F, l)$. Let us apply Algorithm 3 to PN and obtain a free-choice Petri net $PN' = (P', T', F', l)$. Let us consider a system net $SN' = (PN', M_{init}, M_{final})$ with the same initial and final markings (we can construct such a system net since $P \subseteq P'$). Then $L_{SN} = L_{SN'}$.*

Proof Let *TS* and *TS'* be reachability graphs of *SN* and *SN'*, respectively. As it follows from Theorem 3 *TS'* simulates *TS*, and vice versa, also they have the same final marking M_{final}, thus $L_{SN} = L_{SN'}$. □

Corollary 3 (Language inclusion) *Let us consider a system net $SN = (PN, M_{init}, M_{final})$, such that $PN = (P, T, F, l)$, and $\forall t \nexists M : (PN, M_{final})[t\rangle (PN, M)$, i.e, there is no transition enabled in M_{final}. Then let us apply Algorithm 3 and to obtain a free-choice system net $SN' = (PN', M_{init}, M_{final})$ with the same initial and final markings. Suppose that $BPMN_{model}$ is a result of applying Algorithm 2 to SN'. Then $L_{SN} \subseteq L_{BPMN_{model}}$.*

Proof Let *TS*, *TS'*, TS_{BPMN} be reachability graphs of *SN*, *SN'*, and $BPMN_{model}$, respectively. $L_{SN} = L_{SN'}$ by Corollary 2. According to Corollary 1 if *SN'* does not contain any state, in which no transition can be enabled, except M_{final}, then $L_{SN'} = L_{BPMN_{model}}$. But under hypothesis of this corollary *TS* (and consequently *TS'*) may contain states, in which no transition can be enabled, and which are not final. Also note that *SN'* may contain additional states (see the proof of Theorem 3), which represent the reduced behavior of *SN*; among them there may be states without outgoing transitions. Hence, $L_{BPMN_{model}}$ may contain additional traces. □

Corollary 4 (Language equivalence for empty final marking) *Suppose a BPMN model $BPMN_{model}$ was constructed from a system net $SN = (PN, M_{init}, M_{final})$, where $PN =$*

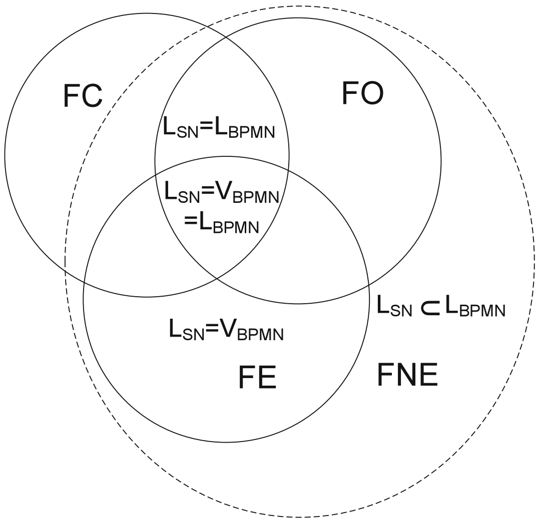


Fig. 28 Relations between languages for system nets and corresponding BPMN models. *FC* is a class of free-choice system nets. *FNE*—a class of system nets, for which no transitions are enabled in final markings. A system net belongs to the class *FE*, iff the final marking is the empty marking. *FO* is a class of system nets, for which their final markings are the only markings with no transitions enabled

(P, T, F, l), using Algorithm 2. If $\forall p \in P : M_{final}(p) = 0$, then $V_{BPMN_{model}} = L_{SN}$. In other words, the set of valid sequences coincides with the language of the system net.

Proof As it follows from Theorem 2 for every marking M of a system net *SN*, there is a marking M' in $BPMN_{model}$, such that for every position p, holds that $M(p) = M'(\mathcal{F}(p))$ and vice versa. In a system net, such that for every node of this net there is a path from the source place to this node, and in a BPMN model no node can fire in an empty marking, and hence, the theorem is proven. □

Fig. 28 summarizes theoretical results presented above: It shows relations between languages of system nets and corresponding BPMN models depending on the type of system nets.

Note that if the initial system net contains transitions with no incoming arcs (unconnected transitions), these transitions will be enabled in any reachable marking of this net. Such nets do not meet the sufficient condition for the language inclusion, i.e., some transitions are always enabled in the final marking.

5 From other process notations to BPMN models

In this section, process modeling formalisms other than Petri nets but relevant for process mining are mapped onto BPMN. Basic ideas of conversion algorithms will be given. First we will present additional BPMN routing constructs, which are naturally used in these conversions.

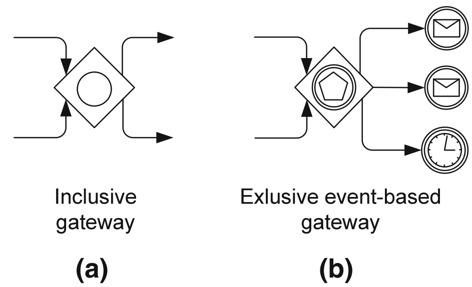


Fig. 29 Additional gateway types **a** Inclusive gateway, **b** Exclusive event-based gateway

5.1 Additional BPMN routing constructs

Let us consider two additional BPMN gateway types specified in [6]. Inclusive gateways (Fig. 29a) implement *multichoice* and *synchronized merge* control flow patterns [36]. An inclusive gateway is enabled in a current marking if some of the incoming sequence flows contain tokens, and it is not possible to reach a marking from the current marking, in which another (currently empty) incoming sequence flow will contain tokens, without firing this gateway. Note that the semantics of inclusive gateways is *non-local*. An enabled inclusive gateway may fire, adding one token to one or more outgoing sequence flows. The gateway presented in Fig. 29a can produce a token for one of the outgoing sequence flows or for both of them.

Next, we consider event-based exclusive gateways, which implement the *deferred choice* control flow pattern [36]. Like an ordinary exclusive gateway, an event-based exclusive gateway (Fig. 29b) is enabled if at least one of the incoming sequence flows contains at least one token. An exclusive event-based gateway produces a token to one of the outgoing sequence flows depending on the event, which first occurred. Thus, a gateway presented in Fig. 29b produces a token to a path marked with a first happened event (a receipt of a concrete message or a timer event).

5.2 Transforming causal nets to BPMN models

Causal nets are known to be suitable for the representation of discovered process models. They are often used for

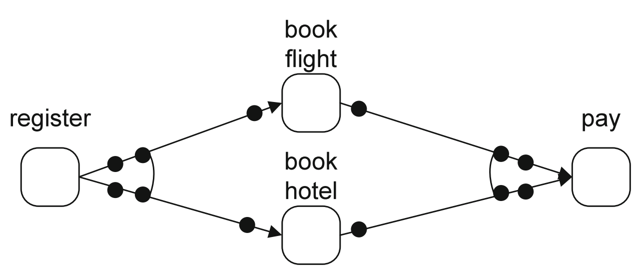


Fig. 30 Causal net of a booking process

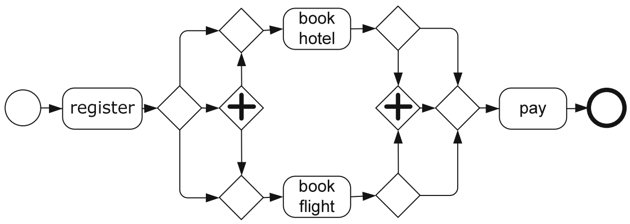


Fig. 31 BPMN model constructed from the causal net example (Fig. 30)

process mining (see e.g., the Heuristic miner [26]), but tend to be unclear for the majority of process analysts. Although an algorithm for the conversion of causal nets to Petri nets was already presented in [10], conversions from causal nets to BPMN models should take into account the free-choice nature of BPMN models.

Causal nets are represented by activities and their bindings: Each activity has a set of input and a set of output bindings (pre- and post-conditions). Let us consider a causal net of a booking process shown in Fig. 30.

The start activity *register* has only empty input binding. There are three possible output bindings for this activity: {*book flight*}, {*bookhotel*} and {*book flight*, *book hotel*}. These bindings imply that activity *register* is followed by activity *book flight*, or activity *book hotel*, or activities *book flight* and *book hotel*. The end activity *pay* has an empty output binding and three possible input bindings, i.e., activity *pay* is preceded by *book flight* and *book hotel* activities, or *book flight* activity, or *book hotel* activity.

While each activity of a causal net is converted to a BPMN model activity, bindings are represented in terms of gateways. If some activity has multiple input (output) bindings, a special exclusive gateway is created, if some binding contains more than one element, a parallel gateway should be added. In case causal net has many start or end nodes, unique start/end nodes are added to simplify the conversion to a BPMN model.

The result of the causal net (Fig. 30) conversion is presented in Fig. 31.

It is important to mention that causal nets provide *declarative* description of the process behavior, while BPMN presented in Fig. 31 has a *local firing rules semantics*. Thus, potentially "unsound" BPMN models may be obtained as a

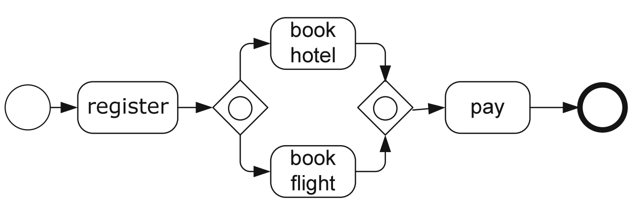


Fig. 32 BPMN model with inclusive gateways

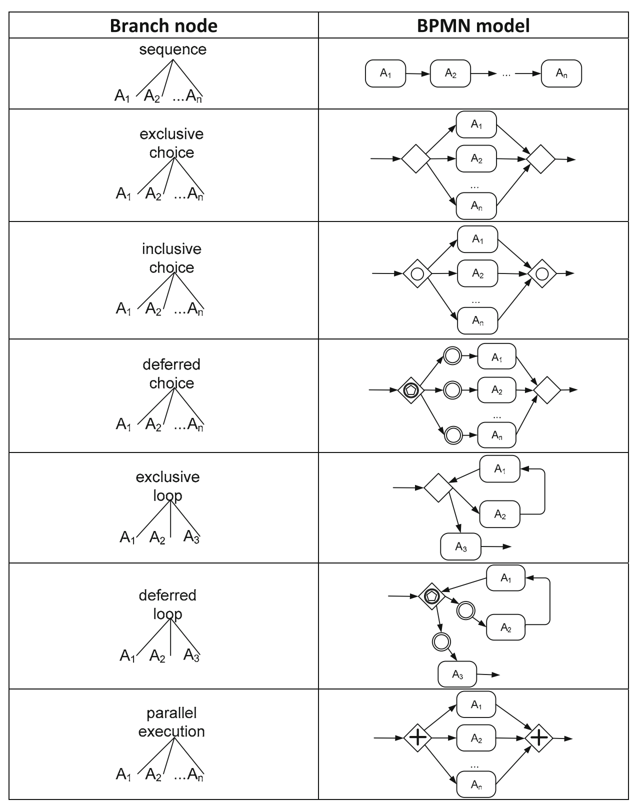
result of conversion. Inclusive gateways can be exploited to obtain simpler models. Fig. 32 shows how the initial causal net can be represented as a BPMN diagram using inclusive gateways.

5.3 Converting process trees to BPMN models

Process trees are often obtained as a result of applying process discovery algorithms (e.g., Inductive miner [25] or Genetic miner [37]). In this subsection, basic transformations for constructing a BPMN model from a given process tree will be proposed. Although process trees can be represented as system nets, and system nets can be converted to BPMN models using algorithms introduced above, a direct conversion algorithm gives an ability to consider additional perspectives during the conversion. Moreover, the BPMN standard natively supports such modeling elements as inclusive gateways and exclusive event-based gateways. Hence, there is no need to convert these constructions to problematic, unreadable process models.

Process trees were proposed in [25] and defined as direct acyclic graphs with branch and leaf nodes. Each branch node has outgoing edges and is considered to be an operator node,

Table 2 Process tree to BPMN conversion



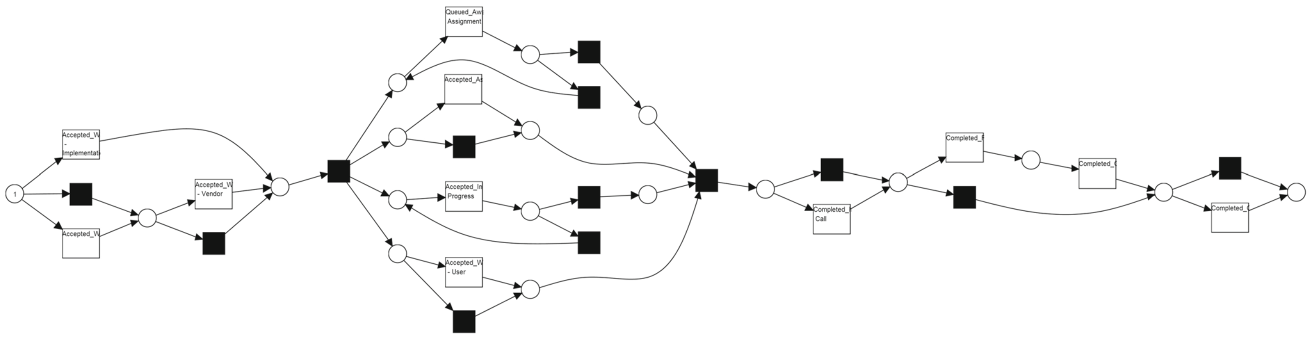


Fig. 33 A Petri net discovered by the Inductive miner from the event log

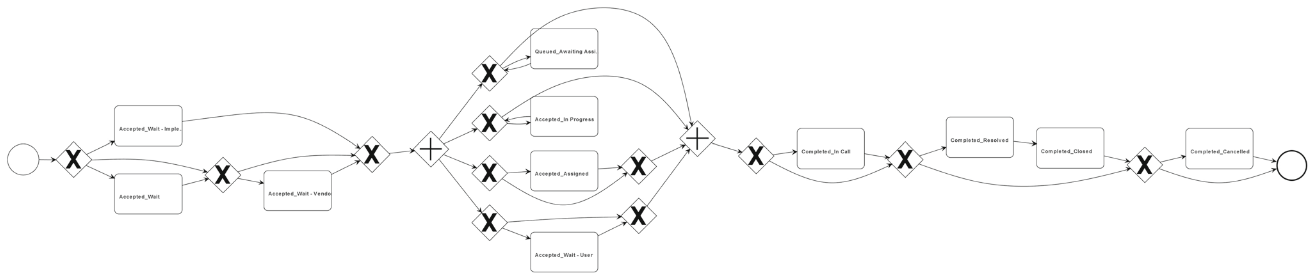


Fig. 34 A BPMN model obtained from the Petri net

and leaf nodes do not have outgoing edges and stand for atomic activities.

Transformation rules are applied inductively, starting from the root node. For each branch node, the transformation depends on a node type and each leaf node is transformed to an activity or BPMN model event. We consider the following basic operators: sequence, exclusive/inclusive/deferred (event-based) choice, exclusive/deferred (event-based) loop and parallel execution. Transformation rules for each type of a branch node are presented in Table 2. Note that for each exclusive event-based gateway, the types of the following events are specified by an additional information attached to the corresponding branch node of the tree.

5.4 Example

In this subsection, we will show how the presented conversion techniques can be applied to discover BPMN models from real-life event logs. We took an event log generated by an *Incident and Problem Management System* from Volvo IT Belgium as an example. For this log, we have applied the

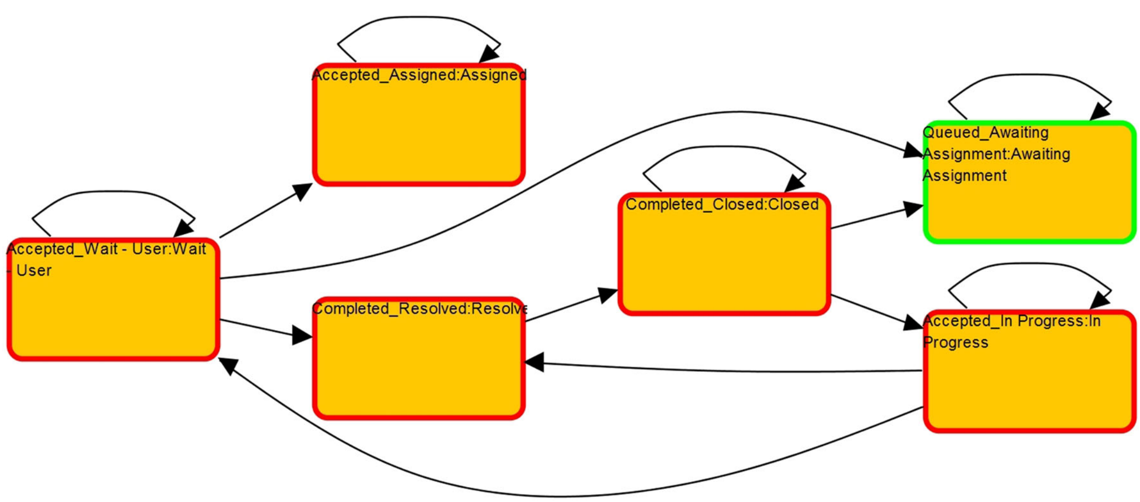


Fig. 35 A causal net discovered from the event log

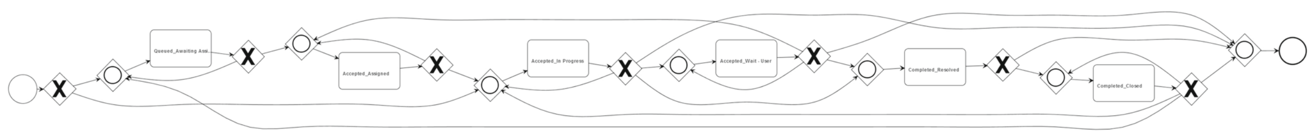


Fig. 36 A BPMN model obtained from the causal net

Inductive miner [25] and have obtained a Petri net presented in Fig. 33.

This Petri net was converted to a BPMN model presented in Fig. 34.

This BPMN model looks more compact, since empty transitions can be omitted, and elements, such as activities, can be connected with each other directly without using intermediate elements. The analysis of cases when the conversions produce simpler models is presented in Sect. 9. A causal net constructed from this log by the Heuristic miner [26] is presented in Fig. 35.

The causal net can be converted to a BPMN model as well (Fig. 36). Note that this BPMN model explicitly represents all routing constructions by means of inclusive and exclusive gateways.

In Sects. 4 and 5, we have shown multiple ways to construct BPMN models from event logs using conversions (refer to Fig. 3 for the overall picture).

In the remainder, we only consider process models without additional routing constructs (inclusive gateways and event-based exclusive gateways). The reason is that we need to convert BPMN models to classical Petri nets for replay.

6 BPMN model simplification

In this subsection, BPMN model transformations are presented. These transformations allow us to reduce the size of target BPMN models. Similar Petri nets and YAWL reduction rules have already been presented in [38–40] and can be applied to BPMN constructions as well.

1. *Removing silent activities.* In contrast to Petri nets, BPMN models allow connecting arbitrary activities and

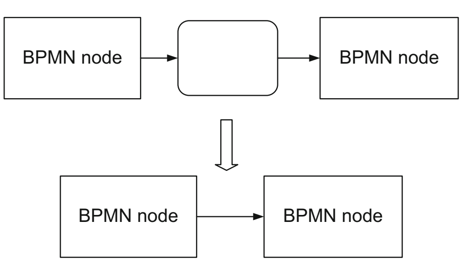


Fig. 37 Reducing gateways of the same type

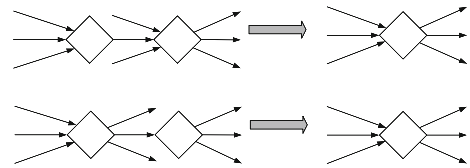


Fig. 38 Removing silent activities

gateways, and thus activities, which are not labeled, may be removed (Fig. 38).

Note that all the activities constructed during an execution of Algorithm 2 have exactly one incoming and exactly one outgoing sequence flow.
2. *Reducing gateways.* Sequential gateways of the same type, serving as join and split routers, can be reduced (Fig. 37).
3. *Merging activities and gateways.* According to the semantics of BPMN, activities can be merged with preceding and following gateways of the right type (Fig. 39).

7 Mapping conformance and performance info onto BPMN models

Process mining is not limited to discovery, but also offers conformance checking and enhancement techniques. To apply existing replay methods, which will allow us to obtain detailed conformance and performance information, the initial BPMN model should first be converted to a Petri net, and after that this Petri net can be analyzed using existing techniques for Petri nets. Note that BPMN models presented in this section are based on the core subset of BPMN modeling elements.

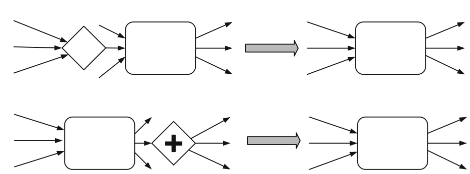


Fig. 39 Merging activities with preceding exclusive join and following parallel split gateways

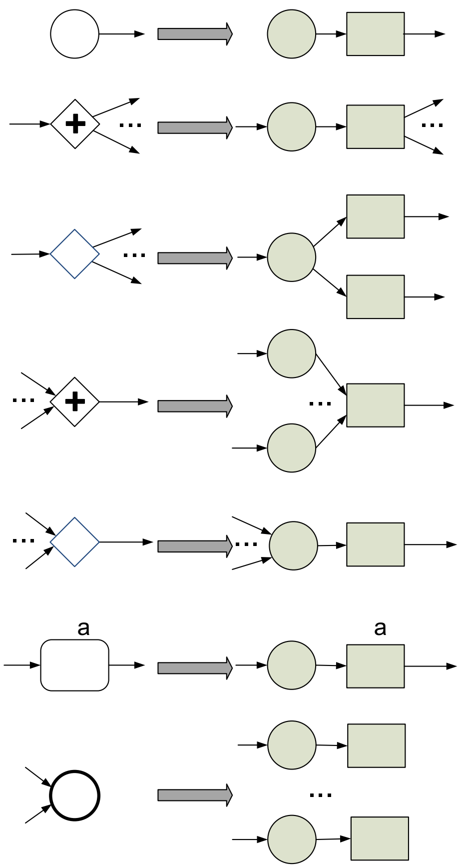


Fig. 40 BPMN to Petri net conversion patterns

7.1 Converting BPMN models to Petri nets

BPMN models based on the core subset of BPMN modeling elements and can be considered as workflow graphs. Every node of a workflow graph can be simply converted to a corresponding Petri net pattern (Fig. 40) by the algorithms, presented in [22,23].

Note that according to [23] some preliminary transformations should be applied to a BPMN model: Each gateway and activity, containing multiple inputs and outputs, should be splitted. Also note that this basic conversion preserves semantics and guarantees *bisimilarity* between a BPMN model and a target Petri net due to the correspondence between BPMN sequence flows and workflow net places: $Map : SF \rightarrow P$, i.e., for each sequence flow of the BPMN model, there is a corresponding place in the target Petri net. The proof of *bisimulation* is straightforward.

7.2 Mapping conformance and performance info

The *bisimulation* relation defines a mapping between states of an initial BPMN model and a target Petri net, and gives us an ability to visualize performance and conformance information, which is attached to some states of the Petri net, within the initial BPMN model.

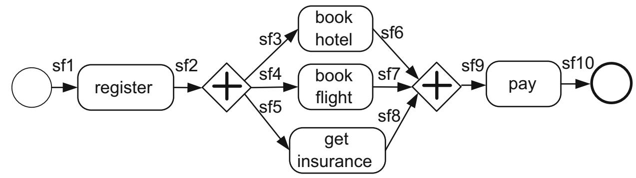


Fig. 41 A BPMN of a booking procedure

To give an example, which shows how a conformance information can be visualized within a BPMN diagram, we have to introduce the notion of alignment. Alignment establishes log and model similarities and discrepancies by defining correspondence between moves on log and moves (firings) on model.

Let A_L be a set of log events. Let also $PN = (P, T, F, l)$ be a labeled Petri net, where $l : T \nrightarrow A_M$, and A_M is a set of model events. Let $\gg$ be a special event, such that $\gg \notin (A_L \cup A_M)$.

Alignment step is a pair (x, y):

- (x, y) is a *move on log* if $x \in A_L$, $y = \gg$,
- (x, y) is a *move on model* if $x = \gg$, $y \in A_M$,
- (x, y) is a *move in both* if $x \in A_L$, $y \in A_M$,
- (x, y) is an *illegal move* if $x = \gg$, $y = \gg$.

An *alignment* is a sequence of *alignment steps* that are not *illegal moves*.

Now let us consider a BPMN model of a booking process (Fig. 41).

Let us apply the conversion algorithm to obtain a Petri net with places, corresponding to BPMN sequence flows. The result is shown in Fig. 42.

To illustrate enhancement of a process model with conformance and performance information, an event log consisting

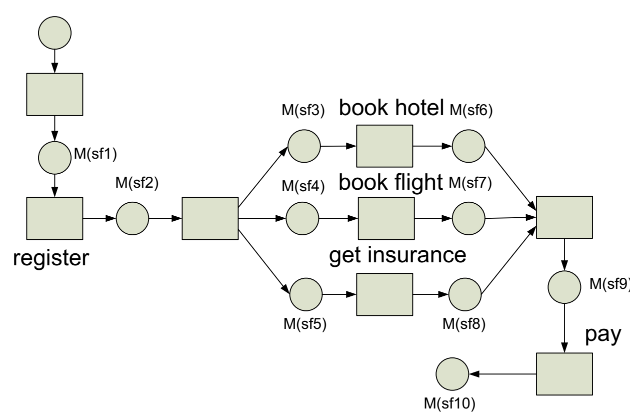


Fig. 42 A Petri net constructed from the BPMN model presented in Fig. 41

Table 3 An event log of a booking process with cancelation

Case ID	Event name	Timestamp
1	Register (r)	2014-12-24 09:30:01:727
1	Book flight (bf)	2014-12-24 09:43:23:353
1	Book hotel (bh)	2014-12-24 09:52:14:252
1	Cancel insurance (ci)	2014-12-24 09:52:20:732
1	Pay (p)	2014-12-24 10:04:24:754

of only one trace, containing insurance cancelation event, is considered (Table 3).

To construct an alignment, this log should be represented as a multiset of traces:

$$L = \Big[\langle register, book\ flight, book\ hotel, cancel\ insurance, pay \rangle^1 \Big].$$

The result of application of the algorithm [41], which finds an optimal (with the minimal number of log only and model only moves) alignment, is presented below (the names of events are represented by their first letters, firing of silent transitions are denoted as τ). The first row represents log moves, while the second row stands for model firings:

$$\gamma = \begin{array}{|c|c|c|c|c|c|c|c|c|} \gg & r & \gg & bf & \gg & bh & ci & \gg & p \\ \hline \tau & r & \tau & bf & gi & bh & \gg & \tau & p \end{array}$$

Such alignments can be used to enhance existing BPMN models with conformance information (Fig. 43).

Note that the relation between states of the models allows us to attach log move information to sequence flows, which correspond to a concrete state of the BPMN model—the state, in which log move is performed.

This BPMN model can be enriched with a performance information (such as activity execution times) obtained as a result of alignment-based replay (Fig. 44).

Note that different types of performance information, such as average, minimal, maximal, relative execution times, can be added to a diagram.

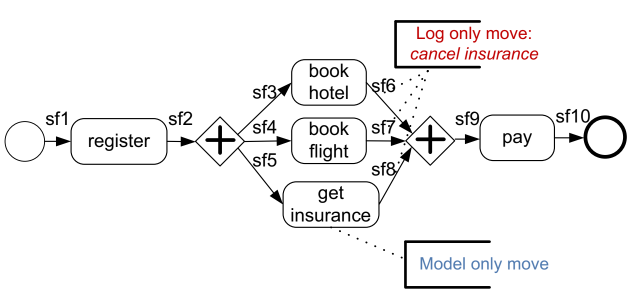


Fig. 43 A BPMN model annotated with conformance information

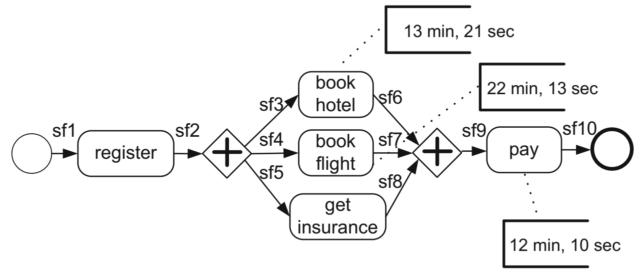


Fig. 44 A BPMN model annotated with performance information

8 Tool support

The techniques presented in this paper have all been implemented in ProM, an open source framework for process mining. Let us consider BPMN packages architecture and their functionality in ProM (Fig. 45).

The core BPMN package provides basic interfaces for working with BPMN models, including import and export of BPMN diagrams in BPMN XML [6] and XPDL 2.2 [42] formats. The BPMN conversions package depends on the core BPMN package and provides the ability to convert Petri nets, causal nets [10] and process trees [11] to BPMN, thus supporting the discovery of BPMN models. Besides that the resource and the data flow perspectives can be discovered as well: Data Petri nets obtained using the data-aware process mining algorithm [28] can be used to create BPMN models with data perspective, and process trees with resources can be

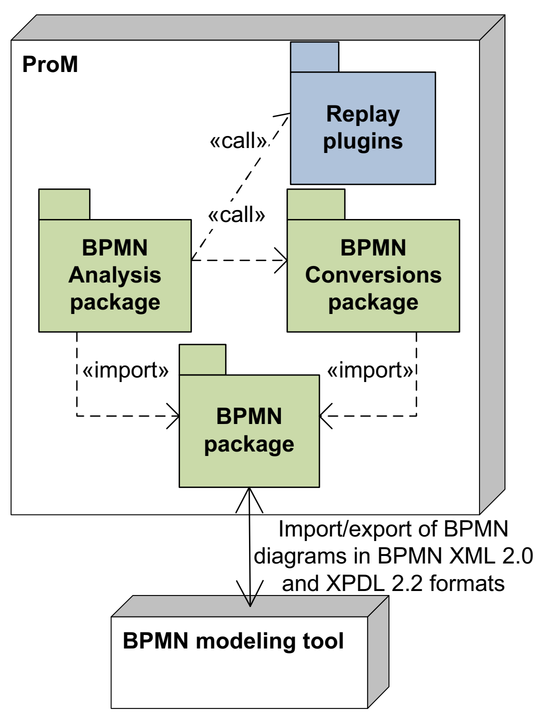


Fig. 45 BPMN packages architecture

converted to BPMN model with lanes. The BPMN Analysis package is constructed on top of the core BPMN, and the BPMN conversions packages and its plugins can be used to enhance BPMN models with additional conformance and performance information.

To illustrate this, we consider two main use cases for working with BPMN in ProM.

Use case 1 (Discovering BPMN processes): The user discovers a BPMN model by applying discovery and BPMN conversions plugins, and then this model can be exported to an external BPMN modeling or execution tool (Fig. 46).

Use case 2 (Analyzing BPMN processes): The user loads a BPMN model from an external BPMN modeling tool, and then, by applying the BPMN Analysis package, it converts this model into a Petri net, replays a log to obtain conformance and performance information, and enhances the BPMN diagram with this information (Fig. 47).

More details about the functionality of the BPMN packages in ProM can be found at [43].

9 Case studies

In this section, we present case studies based on the event logs produced by information systems from different domains.

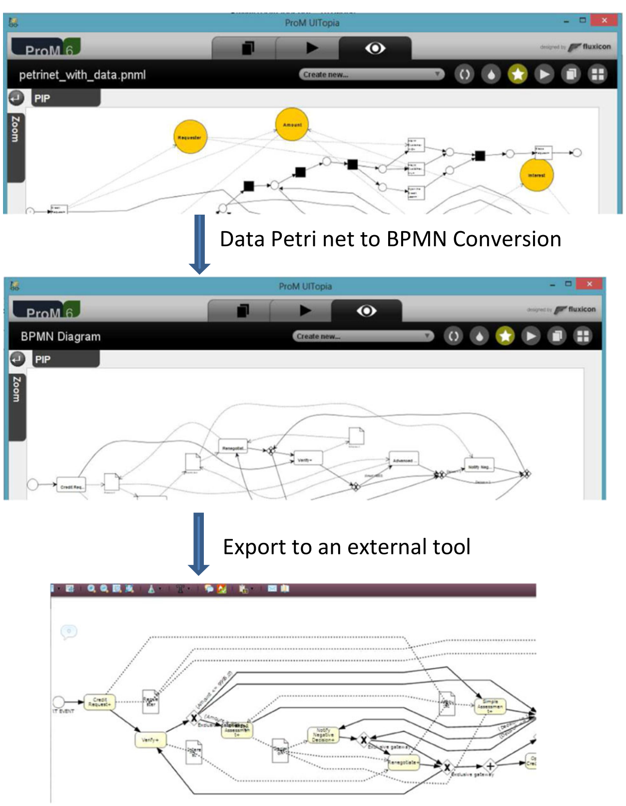


Fig. 46 Discovering a BPMN model with data

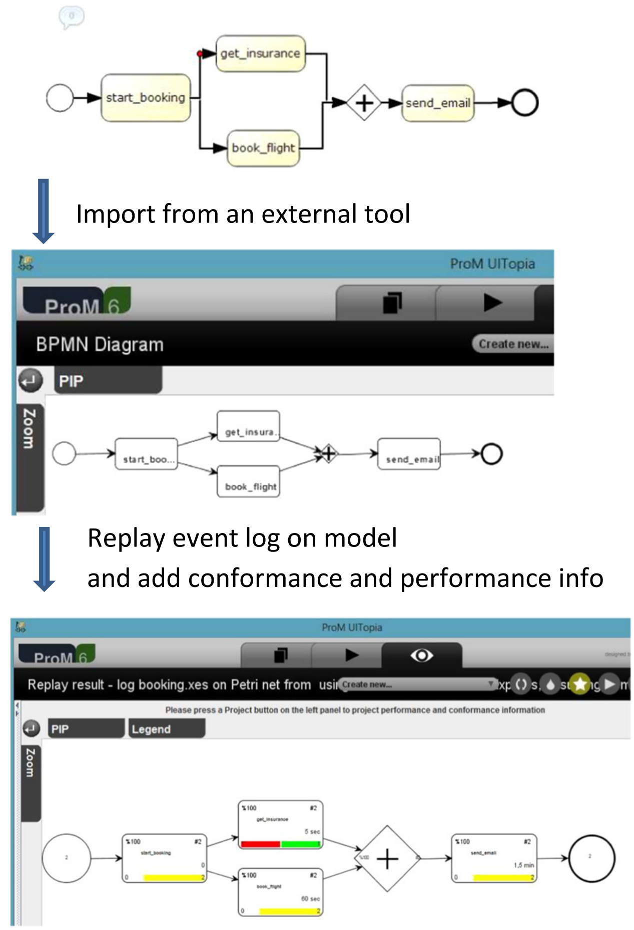


Fig. 47 Analysis of a BPMN model

First we consider a complex ticket reservation system from the traveling and transportation domain. This system belongs to a class of *Computer Reservation Systems* (CRS) and is used for booking all types of touristic products such as hotels, flights, cars, excursions, activities, trains, etc. It integrates multiple Global Distribution Systems (GDS) for pre-sales (search, reservation) and sales (book, pay) processes. The system can be accessed by normal customers through a Web interface or by expert users, e.g., travel agency or system support, through a special rich client interface. Independently of the client interface, calls to the backend of the system are logged in order to track the system state and performance, to gather statistics and to analyze the user behavior and make estimations and predictions.[5] These logs contain the timestamps, identifiers of the business cases (unique identifier of the reservation) and also different business attributes describing user operations (e.g., search, quote, book, pay), travel directions (e.g., origin, destination, booking code), traveling time (e.g., start and end date). For our experiments,

[5] These logs exclude user and commercial sensible information like names, credit cards and invoices.

Table 4 Event log from a CRS

Reservation ID	Event name	Timestamp	Booking code	Notification
390234516	T1-HF-H:TES	2013-12-18 08:36:00:570	BER	
390234516	M55Type010Rsp-034	2013-12-18 08:36:04:717		998
390234516	T3-HF-H:HH004	2013-12-18 08:36:09:337	BER	
390234516	M52Rsp	2013-12-18 08:36:09:337		998
390235717	T1-BA-H:TES	2013-12-18 08:36:12:155	BER45010 DH	
390235717	M52Rsp	2013-12-18 08:36:18:397		712

we took a log of the ticket reservation system, which contains 94 cases, 50 event names, and describes the system behavior during 4 h of its work. This event data are perfectly suitable for applying process mining, since we identify the case by reservation id, event names by a subset of business attributes and order of events by a timestamp.

The other system being analyzed is a *Tracker System* (TS) used for management of kanban and scrum software development processes. The history of the system usage contains short traces (the average number of events per trace is 3.5). Each trace can be identified by a unique number, it corresponds to a concrete task for developers and represents a sequence of task states, such as “Open,” “In Progress,” “Resolved,” “Reopened” and “Closed.” These task states are considered as event names, and timestamps are used to order them.

Event logs produced by six different anonymous municipalities [44,45] within the *CoSeLoG* project in the Netherlands were analyzed as well. These event logs contain information about executions of a permit application process. The original event logs were reduced in order to speed up the discovery techniques applied: The number of traces for these six logs varies from 17 to 324, and the number of different event names is 22 for the log, which contains receiving phase only, and between 69 and 176 for the other event logs. For the log corresponding to the receiving phase, the mean case duration time is 3.1 days, and for the other logs the mean case duration time lies between 21.4 days and 49.2 weeks.

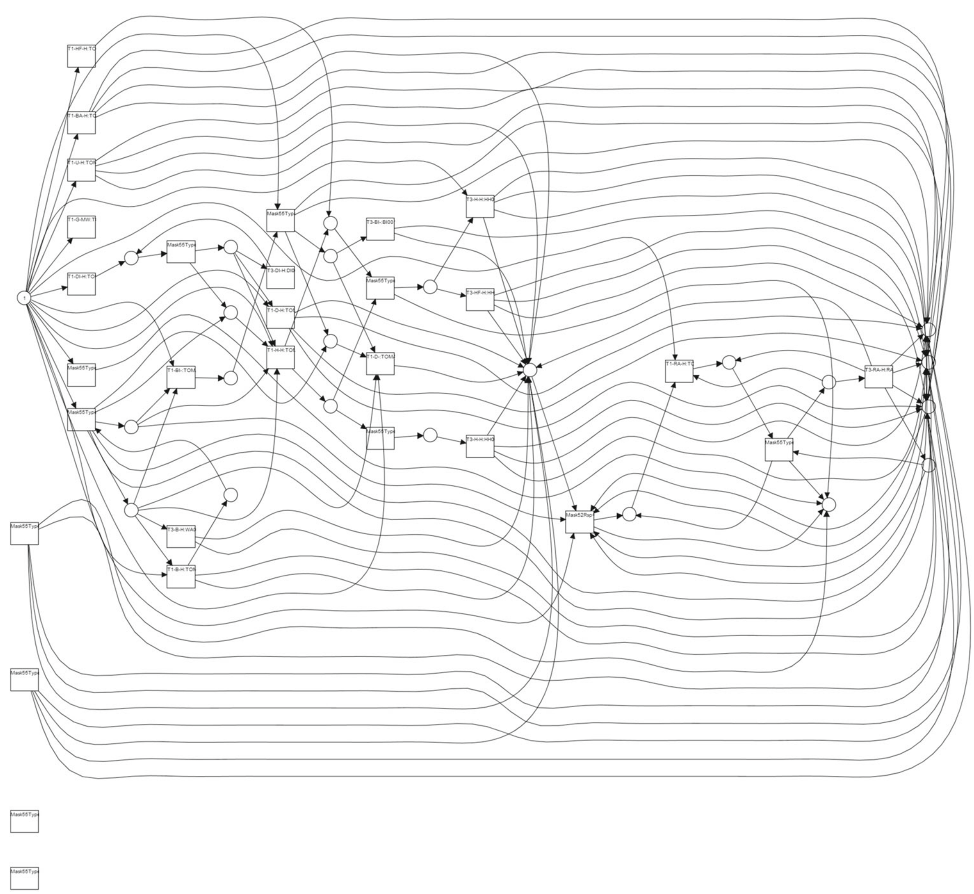


Fig. 48 A Petri net discovered by the Alpha miner from the event log

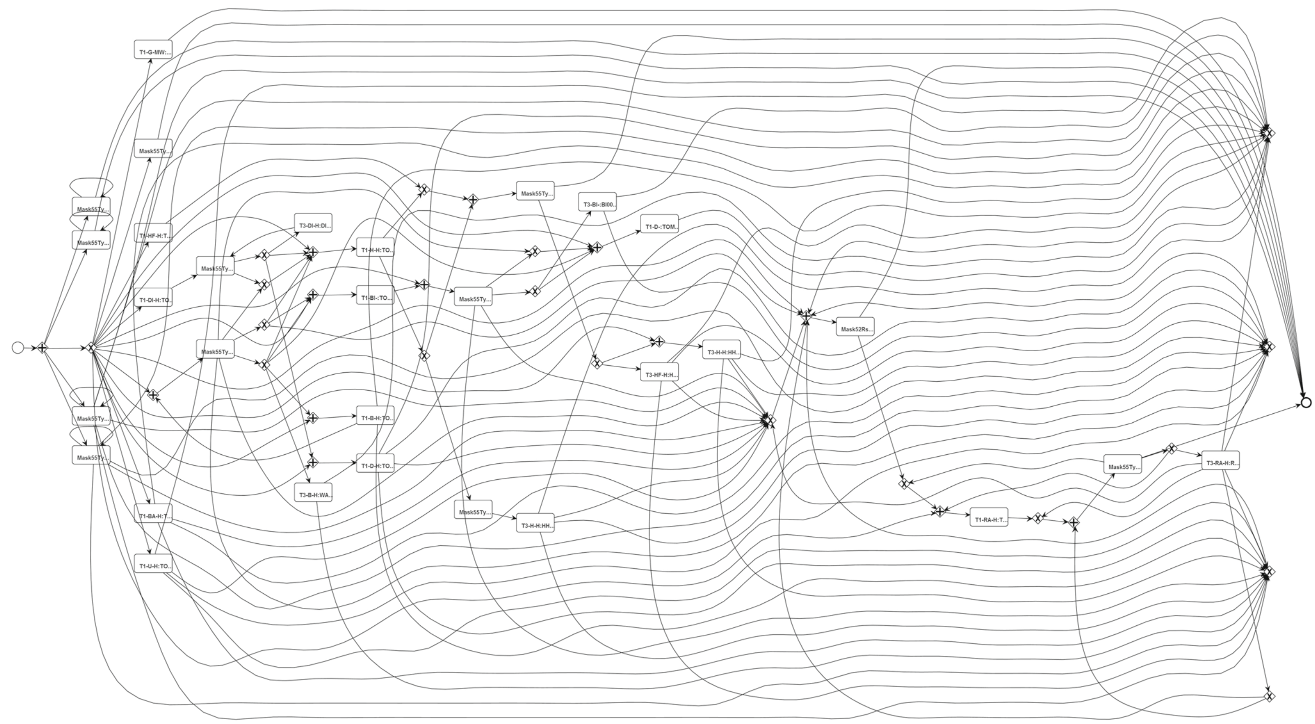


Fig. 49 A BPMN model constructed from the Petri net presented in Fig. 48

9.1 Discovering BPMN models

In this subsection, we show fragments of the CRS log and describe models discovered from this log.

A short typical fragment of an event log is shown in Table 4. Every log entry contains fields such as reservation id, an event name, a timestamp, a booking code and a notification. An event name is a shortcut built from a user operation and of a product type, e.g., “T1-HF-H:TES” means that user does operation “search” for a product type “Hotel.” The booking code identifies the location, e.g., “BER” means Berlin, Germany. Thus, in the example in Table 4 we show two reservations: “390234516” is a search for available hotels in Berlin and “390235717” is a reservation for a double room in a concrete hotel in Berlin.

Further in this subsection, we use different discovery (Heuristic [26], Alpha [12] and Inductive [25] miners) to construct process models from the event log. As a result of applying discovering methods, we obtain Petri nets, causal nets and process trees. All these models are converted to BPMN using algorithms presented before. The log of the booking process was filtered: The parts, which contain only positive or only negative cases (cases with booking error events), were identified. Moreover, since the booking flight procedure differs from other bookings, the log was also filtered by containing booking flight event.

Let us consider models constructed from a filtered log containing only positive traces without flight booking procedure[6]; Fig. 48 illustrates a Petri net discovered with Alpha miner[12] for this log.

The BPMN model constructed for the given Petri net (Fig. 48) by the conversion algorithm is shown in Fig. 49.

Note that in this case the amount of routing elements (gateways) in the BPMN model is comparable with the number places of the initial Petri net, and the number of activities coincides with the number of transitions. Also note that thanks to Algorithm 1 all the activities of the target BPMN model are on paths from the start event to end events.

Now let us consider a causal net (Fig. 50) discovered using the Heuristic miner [26]. Note that the start nodes and the end nodes are highlighted in green and red, respectively.

This causal net can be converted to a BPMN model (Fig. 51) using the conversion algorithm presented before.

The BPMN model reveals the operational semantics of the process. Fig. 50 is not explicitly showing the routing constructions, but the BPMN model does. Now we will consider a process tree discovered by the Inductive miner [25] (Fig. 52).

The corresponding Petri net is presented in Fig. 53.

[6] All the models presented in this section are discovered using ProM framework

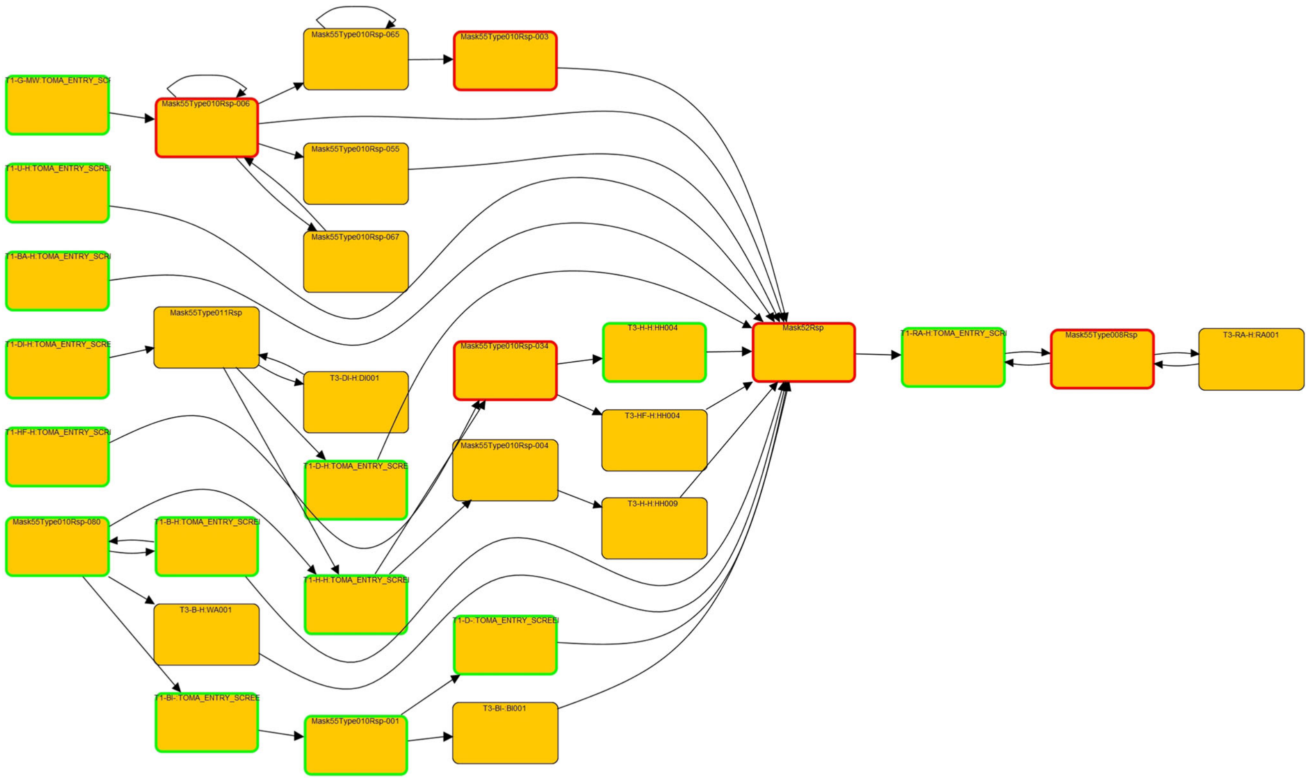


Fig. 50 A causal net discovered by Heuristic miner from the event log

The BPMN model constructed from the process tree by the conversion algorithm does not contain any silent activities (Fig. 54). Moreover, the number of gateways is significantly less than the amount of places in the corresponding Petri net (Fig. 53). Note that the process, discovered by the Inductive miner, is compactly presented in terms of BPMN.

9.2 Comparative analysis of the models discovered

In this subsection, we make a comparative analysis of processes models discovered from the event logs and corresponding BPMN models obtained as a result of conversions using various metrics [16]. We will consider the following

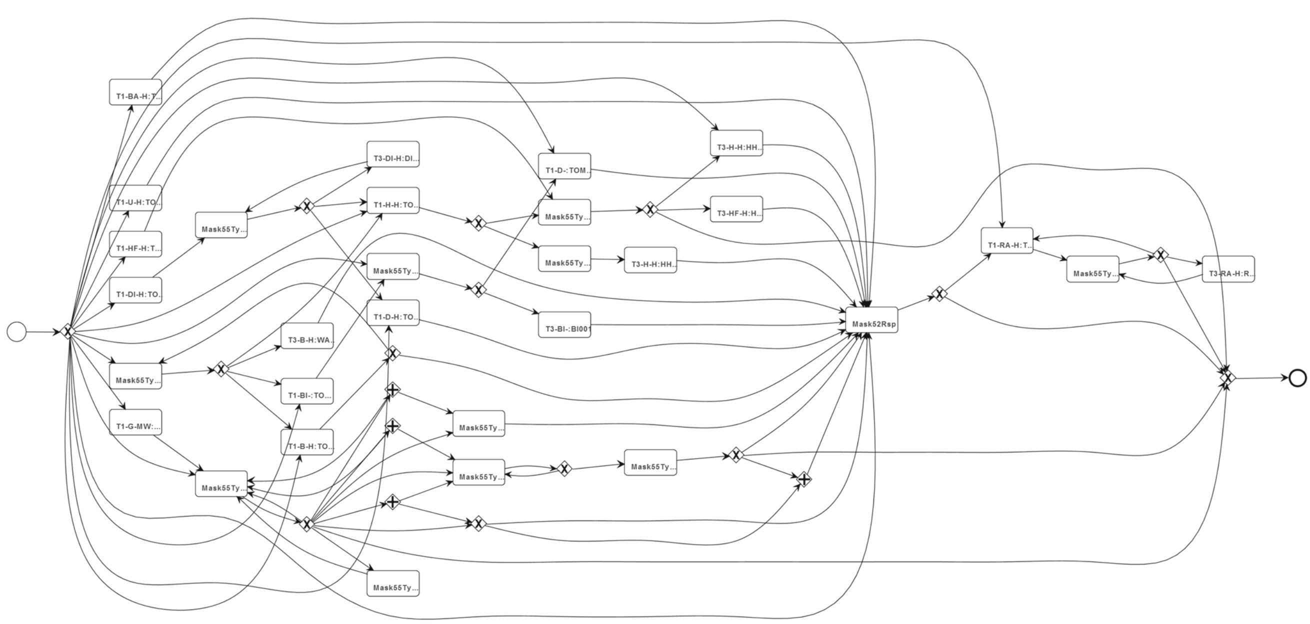


Fig. 51 A BPMN model constructed from the causal net presented in Fig. 50

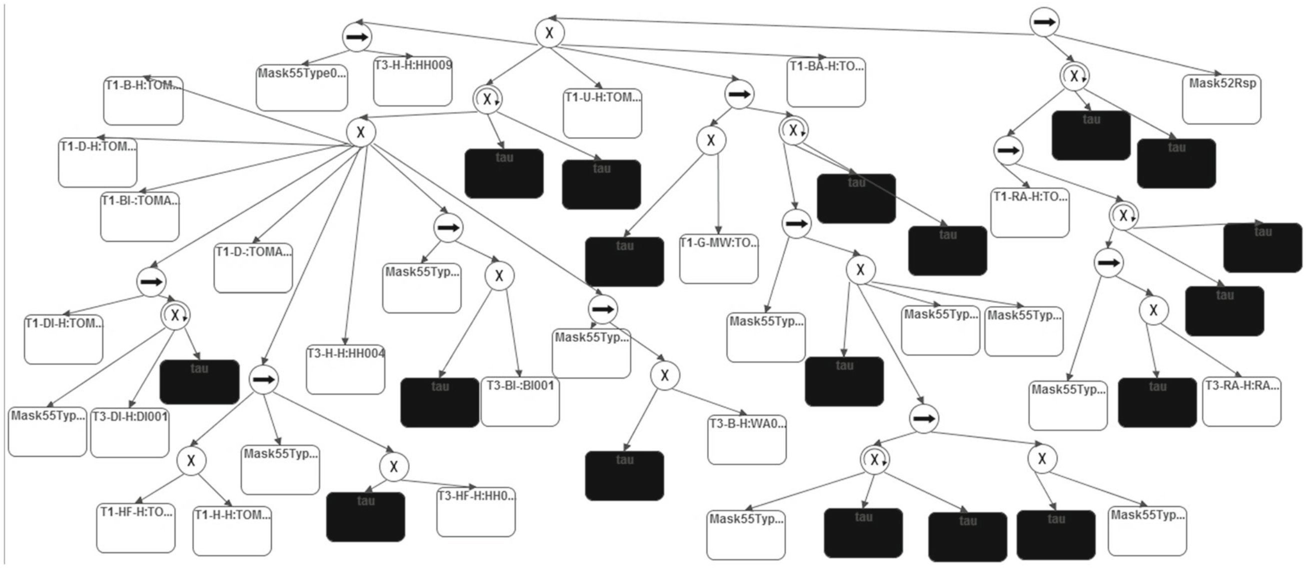


Fig. 52 A process tree discovered by the Inductive miner

metrics: the number of nodes, the diameter (the maximal length of a shortest path from a start node to a node of the graph) and density (ratio of the total number of arcs to the maximum possible number of arcs of the graph). As stated in [16], all these metrics have a negative correlation with the process model understandability.

Different process mining algorithms give process models with different characteristics. Of course, the set of considered process mining algorithms is far from complete. But our aim is to analyze the conversion techniques, thuswise we have selected algorithms, representing the entire classes of discovery approaches, which produce process models with certain properties. The Alpha mining algorithm is not tailored toward handling noisy real-life logs and discovers unstructured process models; meanwhile, the Inductive miner deals with noise and produces structured models. The Heuristic miner was chosen as a technique, which allows us to construct causal nets from event logs.

Let us consider a free-choice system net $SN = (PN, M_{init}, M_{final})$, where $PN = (P, T, F, l)$ is a labeled Petri net,

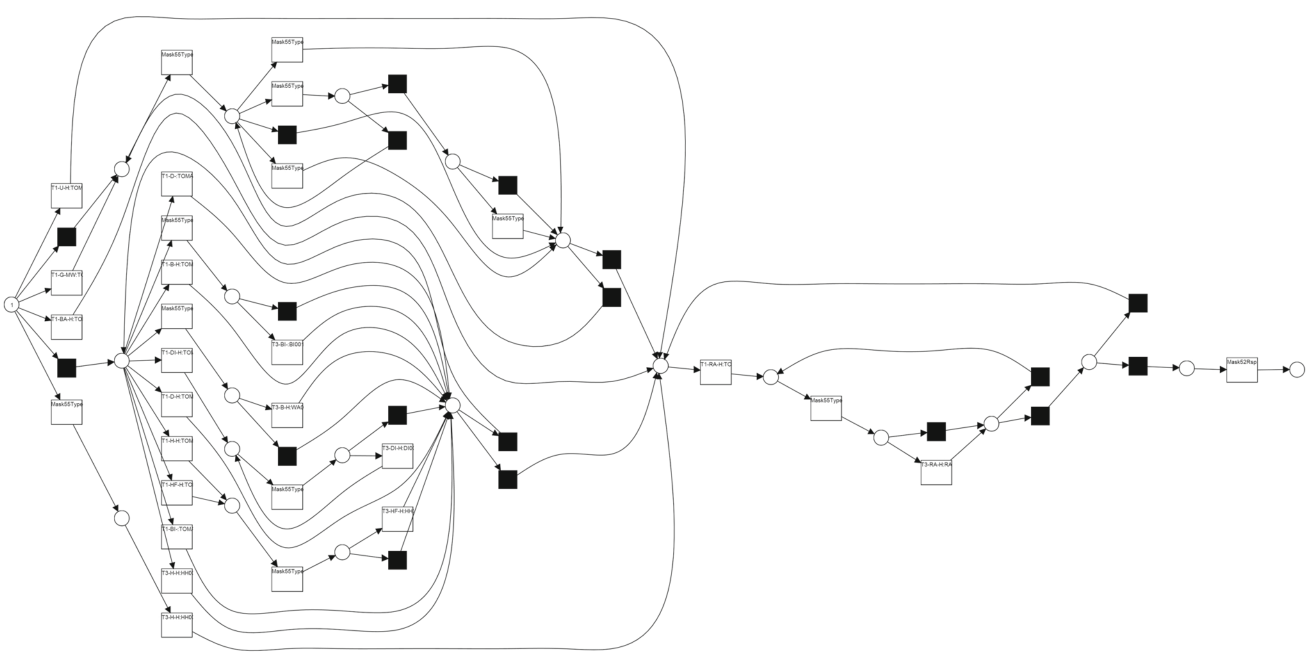


Fig. 53 A Petri net constructed from the process tree shown in Fig. 52

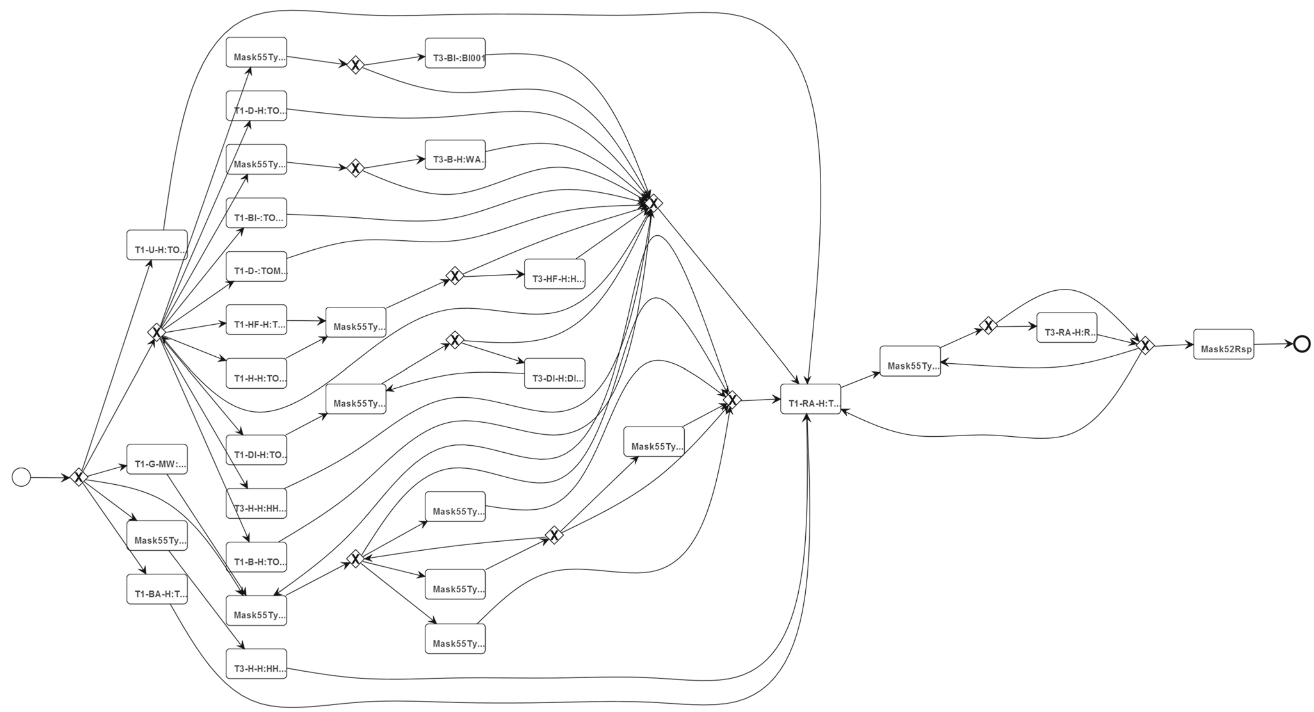


Fig. 54 A BPMN model, which corresponds to the process tree (Fig. 52)

and a BPMN model $BPMN_{model} = (N, A, G_{XOR}, G_{AND}, e_{start}, E_{end}, SF, \lambda)$, obtained by the conversion algorithm from this net. The number of activities equals to the number of non-silent transitions: $|A| = |\{t \in T : t \;\; \lambda(t) \neq \tau\}|$. In the worst case, the number of gateways $|G_{XOR} \cup G_{AND}|$ is comparable to the number of places plus the number of transitions : $|P|+|T|$, since every place can produce an XOR-join, and in the worst case, the number of AND-join gateways and the number of XOR-split gateways are both comparable to $\lfloor |T|/2 \rfloor$. Note that AND-split gateways will be deleted during the simplification of the BPMN model. In the best case (for a sequential control flow), a BPMN model will not contain any gateways. Also note that all the constructions produced during the transformation of arbitrary Petri nets to free-choice nets will contain only silent transitions and related places.

For a BPMN model constructed from a causal net, the number of activities equals the number of causal net activities. The number of gateways is determined by input and output bindings of the activities.

A BPMN model constructed as a result of a process tree conversion contains activities corresponding to non-silent process tree leafs, and every branch node of the process tree will be converted to a routing construction of the BPMN model, containing zero (in the case of a sequential control flow), one or more routing nodes. Note that some routing nodes might be merged during the simplification procedure.

To estimate the number of nodes of process models discovered from the real-life event logs, let us consider Table 5. The rows in this and other tables are ordered by the number of process models nodes.

The first column refers to the event logs used: six event logs, originating from municipal processes, of the *CoSeLoG* project (denoted as CSLG1–CSLG6), the logs of *CSR* (Computer Reservation System) with various types of filtering applied[7] and the logs of *TS* (Tracking system). Each of the other columns mentions the number of nodes of the initial process model constructed by a discovery algorithm and the number of nodes of the BPMN model obtained as a result conversion (for Petri nets and BPMN models the number of transitions, places and the number of activities, XOR gateways, AND gateways are specified, the values are separated by a comma).

This table shows that the number of BPMN model nodes depends on the properties of the initial Petri net: BPMN models constructed for structured Petri nets are more compact (see the Inductive miner column). This holds due to the fact that BPMN language allows simplifications, such as silent nodes deletion (structured models usually contain silent nodes) and gateways reduction, which is also applicable to structured nets, if some blocks can be merged. For non-structured Petri nets (see the Alpha miner column), the

[7] By CSR1, CSR2, CSR3 and CSR4 we denote event logs of the Computer Reservation System, containing only positive cases and/or booking flight event.

Table 5 Number of nodes of process models discovered from the event logs

Log traces	Heuristic miner (C-net/ BPMN[a])	Alpha miner (Petri net[b]/ BPMN[a])	Inductive miner (Proc. tree/ Petri net[b]/BPMN[a])
CSLG1	176/176,143,128	176,372/176,340,120	261/235,49/176,25,0
CSLG2	134/134,58,56	134,238/134,159,54	273/223,89 /134,42,0
CSLG3	93/93,58,70	93,198/93,153,42	219/173,78/93,35,0
CSLG4	75/75,49,52	75,143/75,101,33	151/120,50/75,21,0
CSLG5	68/68,58,58	68,181/69,159,55	84/77,9/68,3,0
CSR1	30/30,14,4	30,23/30,18,13	130/100,53/ 30,25,0
CSR2	30/30,14,5	30,13/30,5,7	102/81,43/ 30,20,0
CSR3	27/27,10,4	27,28/27,24,13	107/82,52/ 27,24,2
CSLG6	22/22,8,6	22,24/22,14,12	54/41,19/ 22,7,0
CSR4	21/21,6,4	21,23/21,18,11	88/ 67,41/ 21,18,0
TS	5/5,7,0	5,7/5,5,4	15 / 9,5 / 5,3,0

[a] For BPMN models, the number of activities, XOR gateways and AND gateways are specified, and the values are separated by a comma
[b] For Petri nets the number of transitions and places are specified and separated by a comma

number of BPMN model nodes is comparable or even greater than the number of nodes of the initial Petri net. Also according to the theoretical observations, the number of nodes of a BPMN model is not always lower than the number of nodes of the initial causal net, since BPMN models may have routing nodes.

Similarly let us estimate the density of the models discovered (Table 6). Graph density is defined as $D = |E|/(|V| * (|V| - 1))$, where E is a set of edges and V is a set of nodes. Density shows the relation between the real number of edges and the maximum possible number of edges in the graph.

The density of the BPMN models constructed from unstructured Petri nets is comparable with the density of these Petri nets (see the Alpha miner column). The density of structured Petri nets is larger than the density of corresponding BPMN models (see the Inductive miner column) due to reductions applied in the case of structured processes. The density of causal nets is certainly greater than the density of corresponding BPMN models, since novel gateways, which connect process activities, are added.

Table 6 Density of the models discovered

Log traces	Heuristic miner (C-net/ BPMN)	Alpha miner (Petri net/ BPMN)	Inductive miner (Petri net/ BPMN)
CSLG1	0.01/0.004	0.005/0.004	0.005/0.01
CSLG2	0.01/0.005	0.005/0.005	0.005/0.01
CSLG3	0.02/0.01	0.01/0.01	0.005/0.02
CSLG4	0.02/0.01	0.01/0.01	0.01/0.02
CSLG5	0.02/0.005	0.005/0.005	0.01/ 0.02
CSR1	0.07/0.04	0.05/0.04	0.01/0.04
CSR2	0.07/0.04	0.03/0.05	0.01/0.04
CSR3	0.06/0.04	0.04/0.03	0.01/0.03
CSLG6	0.07/0.05	0.04/0.04	0.03/0.06
CSR4	0.08/0.05	0.05/0.04	0.01/0.04
TS	0.5/0.2	0.19/0.12	0.1 / 0.16

Table 7 Diameter of the process models discovered from the event logs

Log traces	Heuristic miner (C-net/ BPMN)	Alpha miner (Petri net/ BPMN)	Inductive miner (Petri net/ BPMN)
CSLG1	12/24	17/21	37/14
CSLG2	39/53	49/42	76/23
CSLG3	23/38	37/26	71/23
CSLG4	25/32	39/34	56/16
CSLG5	18/35	13/17	16/5
CSR1	3/7	6/10	20/6
CSR2	3/7	5/7	8/9
CSR3	6/12	11/13	39/14
CSLG6	6/11	13/13	25/7
CSR4	5/8	10/11	41/12
TS	2/5	3/6	5/4

Now let us consider the diameter—the maximal length of a shortest path from a start node to a node of the graph. The results presented in Table 7 show that the statements valid for the number of nodes parameter are also true for the diameter: BPMN models corresponding to structured Petri nets are more compact than the initial models.

In this subsection, we have evaluated the discovered process models using metrics, such as the number of nodes, the density and the diameter. The results show that the compactness of the result BPMN models depends considerably on characteristics of the initial models.

Table 8 Characteristics of process models from the Signavio model collection

	Number of nodes	Density	Diameter
Maximal	58.00	0.87	25
Mean	20.76	0.10	8
Minimal	6.00	0.00	1

Another important issue in the context of our practical case studies presented in this subsection was the understandability of the process mining output format. The software architects and designers of the touristic system were especially interested in getting the results in the BPMN format. They were familiar with BPMN, and BPMN was also used in the specification and design phases of the software product for documenting the typical touristic business processes. Moreover BPMN exchange formats, such as BPMN XML [6] and XPDL 2.2 [42], give us an ability to integrate with a variety of BPMN supporting tools, and thus discovered processes can be analyzed, improved or even automated using external environments. In addition BPMN offers great opportunities to add other perspectives, such as data and resource information, results of conformance and performance analysis. This way the analyst can obtain a holistic view on the processes discovered.

9.3 Comparing discovered and manually created BPMN models

To compare the models discovered from the event log with manually created BPMN models, we analyzed the Signavio model collection, which contains a variety of BPMN models from different domains. We took only flat models represented by start and end events, tasks, gateways and sequence flows. Currently the Signavio model collection contains 4900 of such models. For these models, we calculated the structural characteristics: number of nodes, density and diameter (Table 8).

Comparing these results with the measurements presented in the Tables 5 and 7, one may conclude that models drawn manually (excluding models discovered from the small *TS* log) are usually more compact than those, which were automatically discovered using the well-known discovery and conversion algorithms presented in this paper.

Also these observations show that BPMN models created manually have higher density than automatically discovered BPMN models. The results obtained for the Signavio model collection are a consequence of the fact that business process analysts and engineers are used to work with more structured models, so an algorithm for subprocesses discovery is needed. An algorithm for the construction of BPMN subprocesses based on a log clustering and filtering was proposed in [24]. However, more research is needed to compare handmade and discovered models.

10 Conclusions and future work

This paper provides a solid basis for using BPMN in process mining. The results presented in the paper concentrate on the control flow perspective, as it is usually considered to be the main perspective. It is the starting point for extending with additional perspectives during an enhancement of the process model discovered from an event log.

In this paper, we used various control flow discovery algorithms. These produce Petri nets, causal nets, process trees, etc. Few algorithms produce directly a BPMN model. Hence, we developed various conversion algorithms to mine BPMN. Petri nets, process trees and causal nets discovered from a real-life event log were compared with the corresponding BPMN models on the basis of three process metrics. Moreover, these metrics were applied to measure the difference between BPMN models, which were created by analysts, and BPMN models retrieved as a result of process discovery algorithms. An approach for enhancing a BPMN model with additional conformance and performance information was proposed as well.

The results presented in the paper can be used to retrieve BPMN models out of event logs and verify them against event logs. This work can be considered as a first step toward the development of more advanced process discovery methods, including novel perspectives.

As was shown in Sect. 9.3, more structured process models are needed, and thus methods for subprocesses discovery should be introduced. In comparison with the approach presented in [24], we plan to build a method on top of the decomposition techniques [46–48] to obtain structural models, preserving behavior recorded in an event log.

References

1. van der Aalst, W.M.P.: Process Mining—Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
2. Mans, R.S., Schonenberg, M.H., Song, M., van der Aalst, W.M.P., Bakker, P.J.M.: Application of process mining in healthcare: a case study in a Dutch hospital. In: Biomedical engineering systems and technologies. communications in computer and information science, vol. 25, pp. 425–438. Springer, Berlin (2009)
3. Mitsyuk, A.A., Kalenkova, A.A., Shershakov, S.A., van der Aalst, W.M.P.: Using process mining for the analysis of an e-trade system: a case study. Bus. Informatics **3**, 15–23 (2014)
4. Trcka, N., Pechenizkiy, M.: From local patterns to global models: towards domain driven educational process mining. In: 9th International conference on intelligent systems design and applications (ISDA), pp. 1114–1119. IEEE Computer Society (2009)
5. IEEE task force on process mining: process mining manifesto. In: Business Process Management Workshops, Lecture Notes in Business Information Processing, vol. 99, pp. 169–194. Springer (2012)

6. OMG: Business Process Model and Notation (BPMN), Version 2.0. http://www.omg.org/spec/BPMN/2.0 (2011)
7. Muehlen, M.Z., Ho, D.T.: Service process innovation: a case study of bpmn in practice. In: Proceedings of the proceedings of the 41st annual Hawaii international conference on system sciences, HICSS '08, p. 372. IEEE Computer Society, Washington (2008)
8. Muehlen, M.Z., Recker, J.: How much language is enough? Theoretical and practical use of the business process modeling notation. In: Proceedings of the 20th international conference on advanced information systems engineering. CAiSE '08, pp. 465–479. Springer, Berlin, Heidelberg (2008)
9. Recker, J.: Opportunities and constraints: the current struggle with BPMN. Business Process Manag. J. **16**(1), 181–201 (2010)
10. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Causal nets: a modeling language tailored towards process discovery. In: CONCUR, Lecture Notes in Computer Science, pp. 28–42. Springer (2011)
11. van der Aalst, W.M.P., Buijs, J.C.A.M., van Dongen, B.F.: Towards improving the representational bias of process mining. In: IFIP international symposium on data-driven process discovery and analysis (SIMPDA 2011). Lecture Notes in Business Information Processing, vol. 116, pp. 39–54. Springer, Berlin (2012)
12. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
13. Cardoso, J., Mendling, J., Neumann, G., Reijers, H.A.: A discourse on complexity of process models. In: Proceedings of the 2006 international conference on business process management workshops. BPM'06, pp. 117–128. Springer, Berlin (2006)
14. Latva-Koivisto, A.: Finding a complexity measure for business process models Tech. rep, Helsinki University of Technology, Espoo, Helsinki (2001)
15. Laue, R., Gruhn, V.: Complexity metrics for business process models. In: Business Information Systems, 9th international conference on business information systems, BIS 2006, pp. 1–12 (2006)
16. Sánchez-González, L., García, F., Mendling, J., Ruiz, F., Piattini, M.: Prediction of business process model quality based on structural metrics. In: ER, Lecture Notes in Computer Science. vol. 6412, pp. 458–463. Springer (2010)
17. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdiscp. Rev. Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
18. van der Aalst, W.M.P.: The application of Petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998)
19. OMG: OMG Unified Modeling Language 2.5. OMG. http://www.omg.org/spec/UML/2.5/ (2013)
20. Kindler, E.: On the semantics of EPCs: a framework for resolving the vicious circle. In: International Conference on Business Process Management (BPM 2004), Lecture Notes in Computer Science, vol. 3080, pp. 82–97. Springer (2004)
21. Mendling, J., van der Aalst, W.M.P.: Towards EPC semantics based on state and context. In: Proceedings of fifth workshop on event-driven process chains (WI-EPK 2006), pp. 25–48. Gesellschaft für Informatik, Bonn, Vienna (2006)
22. van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.: An alternative way to analyze workflow graphs. In: Proceedings of the 14th international conference on advanced information systems engineering (CAiSE'02), vol. 2348, pp. 535–552 (2002)
23. Favre, C., Fahland, D., Völzer, H.: The relationship between workflow graphs and free-choice workflow nets. Inf. Syst. **47**, 197–219 (2015)
24. Conforti, R., Dumas, M., Garcia-Baneulos, L., La Rosa, M.: Beyond tasks and gateways: discovering BPMN models with subprocesses, boundary events and activity markers. In: International conference in business process management (BPM), pp. 101–117. Springer, Haifa (2014)
25. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs: a constructive approach. In: Application and Theory of Petri Nets and Concurrency, Lecture Notes in Computer Science. vol. 7927, pp. 311–329. Springer (2013)
26. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: IEEE symposium on computational intelligence and data mining (CIDM 2011), pp. 310–317. IEEE, Paris, France (2011)
27. Weerdt, J.D., vanden Broucke, S.K.L.M., Caron, F.: Bidimensional process discovery for mining bpmn models. In: Fournier, F., Mendling, J. (eds.) Business Process Management Workshops, Lecture Notes in Business Information Processing, vol. 202, pp. 529–540. Springer, (2014)
28. de Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: discovering decisions in processes using alignments. In: ACM symposium on applied computing (SAC 2013), pp. 1454–1461. ACM Press (2013)
29. Burattin, A., Sperduti, A., Veluscek, M.: Business models enhancement through discovery of roles. In: IEEE symposium on computational intelligence and data mining, CIDM 2013, Singapore, pp. 103–110 (2013)
30. Song, M., van der Aalst, W.M.P.: Towards comprehensive support for organizational mining. Decis. Support Syst. **46**(1), 300–317 (2008)
31. de Leoni, M., van der Aalst, W.M.P., van Dongen, B.F.: Data- and resource-aware conformance checking of business processes. In: BIS, Lecture Notes in Business Information Processing. vol. 117, pp. 48–59. Springer (2012)
32. van der Aalst, W.M.P., Rubin, V.A., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87–111 (2010)
33. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: International conference on business process management (BPM 2007), Lecture Notes in Computer Science. vol. 4714, pp. 375–383. Springer (2007)
34. Carmona, J., Cortadella, J.: Process mining meets abstract interpretation. In: ECML/PKDD 210, Lecture Notes in Computer Science, vol. 6321, pp. 184–199. Springer (2010)
35. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining: adaptive process simplification based on multi-perspective metrics. In: International conference on business process management (BPM 2007), Lecture Notes in Computer Science, vol. 4714, pp. 328–343. Springer (2007)
36. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow control-flow patterns: a revised view. Tech. Rep. BPM-06-22, BPM Center. http://bpmcenter.org/wp-content/uploads/reports/2006/BPM-06-22.pdf (2006)
37. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A Genetic algorithm for discovering process trees. In: IEEE Congress on evolutionary computation (CEC 2012), pp. 1–8. IEEE Computer Society (2012)
38. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, New York (1995)
39. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
40. Wynn, M., Verbeek, H., van der Aalst, W.M.P., Hofstede, A., Edmond, D.: Reduction rules for YAWL workflows with cancellation regions and OR-join. Inf. Softw. Technol. **51**(6), 1010–1020 (2009)
41. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: EDOC, pp. 55–64. IEEE Computer Society (2011)

42. WFMC: XML Process Definition Language Version 2.2. In: Technical Report WFMC-TC-1025, Workflow Management Coalition, Lighthouse Point, Florida, USA (2012)
43. Kalenkova, A.A., de Leoni, M., van der Aalst, W.M.P.: Discovering, analyzing and enhancing BPMN models using ProM. In: Business Process Management—12th International Conference, BPM 2014, Haifa, Israel, September 7–11, 2014. Proceedings (2014). In press
44. Buijs, J.C.A.M.: Environmental Permit Application Process. CoSeLoG project (2014)
45. Buijs, J.C.A.M.: Receipt phase of an environmental permit application process. CoSeLoG project (2014)
46. van der Aalst, W.M.P.: A general divide and conquer approach for process mining. In: Federated conference on computer science and information systems (FedCSIS 2013), pp. 1–10 (2013)
47. Kalenkova, A.A., Lomazova, I.A.: Discovery of cancellation regions within process mining techniques. Fundam. Inform. **133**(2–3), 197–209 (2014)
48. Kalenkova, A.A., Lomazova, I.A., van der Aalst, W.M.P.: Process model discovery: A method based on transition system decomposition. In: Application and Theory of Petri Nets and Concurrency, Lecture Notes in Computer Science, pp. 71–90. Springer (2014)

Anna A. Kalenkova was graduated in applied mathematics from the Lomonosov Moscow State University; in 2011 she completed her PhD in the Computing Center of Russian Academy of Sciences. Currently she is working in the Laboratory of Process Information Systems Higher School of Economics, Moscow, Russian Federation as a senior researcher. Her main topics of interest are Petri nets, workflow nets, business process modeling languages, business process management systems and process mining.

Wil M. P. van der Aalst is a full professor of Information Systems at the Technische Universiteit Eindhoven (TU/e) where he is also scientific director of the Data Science Center Eindhoven (DSC/e). His personal research interests include workflow management, process mining, Petri nets, business process management, process modeling, and process analysis. Wil van der Aalst has published more than 180 journal papers, 18 books (as author or editor), 400 refereed conference/workshop publications, and 60 book chapters. He is a member of the Royal Netherlands Academy of Arts and Sciences, Royal Holland Society of Sciences and Humanities, and Academia European.

Irina A. Lomazova is a Professor at the Faculty of Computer Science, National Research University Higher School of Economics (Moscow, Russia), where she also heads the Laboratory for Process-Aware Information Systems. Her research interests include process modeling and analysis, formal models of concurrency, Petri nets, and process mining.

Vladimir A. Rubin During more than 14 years he has been working and researching in the area of software engineering in a variety of projects in different consulting and technology companies (msg systems AG, Capgemini, sd&m, NetCracker) and scientific institutions in Germany, Russia, USA, Holland, and Switzerland. He graduated in computer science from the Moscow State University of Railway Transport and completed his PhD in a graduate school of dynamic intelligent systems in Paderborn, Germany. Currently he is working as an independent Lead IT Architect and Consultant in Frankfurt am Main, Germany.