Noname manuscript No.
(will be inserted by the editor)

Discovering High-level BPMN Process Models from Event Data

**A. A. Kalenkova · A. Burattin · M. de Leoni ·
W. M. P. van der Aalst · A. Sperduti**

Received: / Accepted: date

Abstract Process mining is a well established research discipline comprising approaches for process analysis based on the history of process executions. One of the main directions in the process mining field is process discovery. Process discovery aims to develop methods for constructing process models from the event logs. The ultimate goal of process discovery is to obtain readable process models, which represent process behavior in the best possible way. Process discovery techniques rarely use higher-level process modeling notations like BPMN and tend to focus on the control-flow perspective. However, end-users are more familiar with notations like BPMN and are interested in the data and resource perspectives. This paper gives an overview of existing process discovery techniques and presents a BPMN meta-model, which describes models that can be obtained from the event logs using existing discovery techniques. Besides that the paper presents an integrated discovery approach for the construction of high-level BPMN models that comply with the BPMN meta-model. The proposed integrated approach was applied to real-life event logs and it was shown that it allows for obtaining readable process models, which reflect the process behavior.

Keywords process mining, process discovery, BPMN (Business Process Model and Notation), process modeling perspectives

This work was supported by the Basic Research Program at the National Research University Higher School of Economics, funded by RFBR and Moscow city Government according to the Research project № 15-37-70008 "mol_a_mos" and RFBR (Project № 15-37-21103) and was supported by the Eureka-Eurostars PROMPT project (E!6696).

A. A. Kalenkova
National Research University Higher School of Economics, Moscow, Russia
E-mail: akalenkova@hse.ru

A. Burattin
University of Innsbruck, Innsbruck, Austria
E-mail: andrea.burattin@uibk.ac.at

M. de Leoni · W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven, Netherlands
E-mail: m.d.leoni@tue.nl,w.m.p.v.d.aalst@tue.nl

A. Sperduti
The University of Padua , Padua, Italy
E-mail: sperduti@math.unipd.it

1 Introduction

Information systems in different domains, such as healthcare, tourism, banking, government and others, record operational behavior in the form of event logs. The process mining discipline [1] offers dozens of techniques to discover, analyze, and visualize processes running in information systems basing on the event logs. The *representational bias* (the language for processes representation) plays an important role in the process discovery. In this work we take BPMN language [24] as a representational bias and as a starting point for process discovery. BPMN is a common process modeling language widely used by analysts and programmers in various domains. This work aims to bridge the gap between existing process mining techniques and BPMN – a commonly used industrial process modeling standard. Although there exist techniques [9, 10, 14, 16] for discovering some of the BPMN modeling constructs, these are often limited to a single perspective, e.g., just the control flow, subprocesses, or just resources. The goal of this work was to formalize and project the BPMN specification onto the process mining domain and suggest a unified integrated approach allowing for the discovery of multiperspective BPMN models. Such an approach gives an ability not only to analyze and visualize discovered processes in BPMN-complaint editors, but even automate their executions using one of the BPMN engines.

First, we analyzed the BPMN meta-model [24], extracting the main modeling elements, which can be discovered using process mining techniques. Thus, three main types of BPMN models, which inherit core BPMN modeling constructs were identified: *BPMN models with data*, *hierarchical BPMN models* and *BPMN models with resources*. Formal execution semantics for these types of models were defined. After that we analyzed the applicability of the existing process discovery techniques to mine these specific models. For hierarchical BPMN models a novel algorithm, which uses a well-defined method for constructing composite process models from localized event logs [4], was adopted and justified. It was proven that each log trace that can be replayed by a composite process model, constructed from a localized event log, can be replayed by the target hierarchical BPMN model. Then an integrated approach for mining *integrated BPMN models*, which combines different perspectives, including data, resource, and hierarchical BPMN models, was developed. The integrated discovery approach was implemented as plugin for ProM (Process Mining Framework) [13]. It was evaluated using real-life event logs taken from government, online sales and banking domains. The underlying approach for discovering composite process models from localized event logs [4] allows us to apply some of the techniques, such as resource discovery and alignment construction, to each of the submodels, thus significantly reducing the overall discovery time. The behavioral and structural characteristics of the discovered BPMN models were also evaluated. The structural characteristics of BPMN models discovered from real-life event logs were compared with the structural characteristics of BPMN models taken from the Signavio model collection, and thus, it was shown that the models discovered automatically using the integrated approach resemble manually created models, which are common for the analysts.

The remainder of this paper is organized as follows. Section 2 gives an overview of related work. In Section 3 all notions, including event logs, Petri nets, BPMN modeling constructs are introduced. Besides that Section 3 presents semantics for various types of BPMN modeling constructs. Section 4 demonstrates the applicability of the existing process discovery techniques to mine different types of BPMN models. Moreover, Section 4 introduces and justifies an approach for the discovery of hierarchical BPMN models, using techniques described in [4]. An integrated discovery approach is presented in Section 4 as well. Section 5 describes implementation aspects of the integrated discovery approach. All experimental results are presented in Section 6. Section 7 concludes the paper.

2 Related Work

Recently a few techniques for discovering BPMN models from event logs were proposed. In [9, 14] a novel method for mining hierarchical BPMN models with subprocesses is presented. Paper [10] suggests an approach for discovering BPMN models represented by control and resource perspectives.

Our approach is an extension of the previous work limited to flat BPMN models discovery [16, 17]. In [16] a conversion algorithm for relating Petri nets and flat BPMN models is introduced and justified. This algorithm gives a solid background for applying advanced discovery techniques proposed in this paper. In [17] technical aspects of its implementation are presented.

In contrast to the approaches [10], which are limited to just the control and resource perspectives, and based on a rather specific process discovery algorithm, we present a flexible discovering framework based on the BPMN meta-model tailored towards the process mining field. Thus, our approach incorporates various discovery techniques in a flexible manner, allowing discovering data flows, subprocesses and other modeling elements.

The subprocess discovery technique presented in this paper is based on a robust approach for discovering composite process models from localized event logs enriched with additional information on the so-called "location" of events (enabling the discovery of hierarchical models) [4]. The approach [4] gives formal guarantees on replay of the log traces, is flexible and can be potentially used for the discovery of different types of communication constructions. In this work we used it for mining subprocesses with multiple arbitrary outgoing flows modeled as cancellations, while in [9, 14] only global cancellations, which lead to a termination event of the entire process model are discovered. Moreover, in comparison with [9, 14] due to the formal nature of the approaches [4] and [16] the language inclusion property of the proposed subprocess discovery technique was verified.

3 Event Logs, Petri Nets, and BPMN Modeling Constructs

This section defines main concepts, including event logs and process modeling formalisms, which will be referred later in this paper.

3.1 Event Logs

Event logs, containing information systems' behavior, are considered as a starting point for the process discovery algorithms.

The definition of event logs reflects the definition reported in [4]. Let $\mathcal{U}_A$ be a set of possible activity names, $\mathcal{U}_{Attr}$ and $\mathcal{U}_{Val}$ be sets of event attributes and their values.

A tuple $L = (E, Tr, act, attr)$ is called an *event log*, if E is a set of events, $Tr \subseteq E^{\star}$ is a set of traces (each trace is a sequence of events), $act : E \to \mathcal{U}_A$ maps events onto activity names, and $attr : E \to (\mathcal{U}_{Attr} \nrightarrow \mathcal{U}_{Val})$[1] defines per event the attributes and their values.

Consider the fragment of an event log presented in Tab. 1. This event log contains infor-

[1] $\nrightarrow$ denotes a partial function, which is defined for a subset of domain elements.

	Case ID	Activity Name	Resource	Time (minutes)
e_1	1	"receive application"	"John"	-
e_2	1	"send acknowledgment of receipt"	"John"	1
e_3	2	"receive application"	"Mary"	-
e_4	1	"process application"	"Kate"	124
e_5	2	"send acknowledgment of receipt"	"Mary"	5
e_6	2	"forward to competent authority"	"Jane"	25

Table 1: An event log of an application processing procedure.

mation about executions of an application processing procedure. The arrival of an application initiates a process instance. After the application is received an acknowledgment is sent back to the applicant and the application is either processed or forwarded to a competent authority. Each case (identified by the attribute *Case ID*) corresponds to the processing of a concrete application and represents a sequence of events (a trace). The values of *Resource* and *Time* attributes are given for these events. The *resource* attribute sets an event performer, while *Time* attribute presents time in minutes needed for the execution of a corresponding operation. More formally, this event log can be defined as a tuple $L = (E, \mathit{Tr}, \mathit{act}, \mathit{attr})$, where $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $\mathit{Tr} = \{\langle e_1, e_2, e_4\rangle, \langle e_3, e_5, e_6\rangle\}$, $\mathit{act}(e_1) = \mathit{act}(e_3) =$ *"receive application"*, $\mathit{act}(e_2) = \mathit{act}(e_5) =$ *"send acknowledgment of receipt"*, $\mathit{act}(e_4) =$ *"process application"*, $\mathit{act}(e_6) =$ *"forward to competent authority"*. Event attributes are defined in the following way: $\mathit{attr}(e_1)(\text{"Resource"}) = \text{"John"}$, $\mathit{attr}(e_2)(\text{"Time"}) = \text{"1"}$, where $\mathcal{U}_A = \{$ *"receive application"*, *"send acknowledgment of receipt"*, *"process application"*, *"forward to competent authority"* $\}$, $\mathcal{U}_{\mathit{Attr}} = \{$ *"Resource"*, *"Time"* $\}$, $\mathcal{U}_{\mathit{Val}} = \{$ *"John"*, *"Mary"*, *"Kate"*, *"Jane"*, *1*, *5*, *25*, *124* $\}$.

Suppose that $E' \subseteq E$ is a subset of events, then the *projection* $L_{\uparrow E'}$ of the log $L = (E, \mathit{Tr}, \mathit{act}, \mathit{attr})$ on set E' is defined as follows: $L_{\uparrow E'} = (E_{\uparrow E'}, \mathit{Tr}_{\uparrow E'}, \mathit{act}_{\uparrow E'}, \mathit{attr}_{\uparrow E'})$, where $E_{\uparrow E'} = E \cap E'$, $\mathit{act}_{\uparrow E'} = \mathit{act} \cap (E' \to \mathcal{U}_A)$, $\mathit{attr}_{\uparrow E'} = \mathit{attr} \cap (E' \to (\mathcal{U}_{\mathit{Attr}} \nrightarrow \mathcal{U}_{\mathit{Val}}))$, and $\mathit{Tr}_{\uparrow E'} = \{t_{\uparrow E'} | t \in \mathit{Tr}\}$, where $t_{\uparrow E'}$ is defined inductively: if $t = \langle\rangle$, then $t_{\uparrow E'} = \langle\rangle$, if $t = \langle e\rangle \cdot t'$, and $e \in E'$, then $t_{\uparrow E'} = \langle e\rangle \cdot t'_{\uparrow E'}$, if $e \notin E'$, then $t_{\uparrow E'} = t'_{\uparrow E'}$.

3.2 Petri Nets

Petri nets is the most popular low-level process modeling formalism used in the context of process mining. A *Petri net* is a tuple $PN = (P, T, F, M_{\mathit{init}}, M_{\mathit{final}})$, where P is the set of places, T is the set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and $M_{\mathit{init}}, M_{\mathit{final}}$ are initial and final markings respectively. A *marking* is a function $M \in P \to \mathbb{N}$, which maps places to natural numbers. Thus, $M_{\mathit{init}}, M_{\mathit{final}} \in P \to \mathbb{N}$. In a marking M place p contains $M(p)$ tokens represented as black dots. In addition, we will use the following notation: by $[p_1, p_2^3]$ we will denote a marking, in which place p_1 contains one token, place p_2 contains three tokens, while other place are empty.

For transition t sets of input and output places are defined as: ${}^\bullet t = \{p \in P | (p, t) \in F\}$, and $t^\bullet = \{p \in P | (t, p) \in F\}$ correspondingly. Places are represented by circles, transitions by boxes, and the flow relation by directed arcs.

Transition t is *enabled* in marking M iff $\forall p \in {}^\bullet t : M(p) \geq 1$, i.e., each input place 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 is added to each of output places $t^\bullet$.

A *labeled Petri net* $PN = (P, T, F, M_{init}, M_{final}, l)$ is defined as a Petri net $(P, T, F, M_{init}, M_{final})$ and a labeling function $l \in T \nrightarrow \mathcal{U}_A$, which maps transitions to a universe of activity names. If l is not defined for $t \in T$, then t is called *invisible* and represented as a black box. Function l can be applied to transition sequences using the following inductive definition: $l(\langle\rangle) = \langle\rangle$, $l(\langle t\rangle \cdot \sigma')$ equals $l(t) \cdot l(\sigma')$, if $l(t)$ is defined, and $l(\sigma')$, otherwise.

Let us consider an event log $L = (E, Tr, act, attr)$. We say that trace $\langle e_1, ..., e_k\rangle \in E$ can be *replayed* by a labeled Petri net $PN = (P, T, F, M_{init}, M_{final}, l)$ if there is a sequence of transition firings σ, leading from the initial marking M_{init} to the final marking M_{final}, such that $l(\sigma) = \langle act(e_1), ..., act(e_k)\rangle$.[2]

Now let us extend Petri nets and define Petri nets with data, which were introduced in [22]. A *Petri net with data* is a tuple $DPN = (PN, V, U, R, W, G)$, where PN is a labeled Petri net, V is a set of variables, U is a domain function, which defines possible values for each variable v. By $D = \cup_{v \in V} U(v)$ we denote a set of all possible variables' values. Functions $R : T \rightarrow 2^V$ and $W : T \rightarrow 2^V$ define sets of variables read and written by each transition respectively. Guard function $G : T \nrightarrow \mathcal{G}_V$, where $\mathcal{G}_V$ is a set of guards, associates transitions with guards. Variables are shown as circles with dashed boarders, transitions, which read or write variables, are connected with them by incoming or outgoing dashed arrows respectively.

A transition is enabled iff a corresponding guard (in case it was defined for that transition) evaluates to true and all the input places contain at least one token. A state of a Petri net with data is a pair (M, Val), such that M is a marking, and Val is a function which maps variables to its values, i.e., $Val : V \rightarrow D \cup \{\perp\}$. Sign $\perp$ denotes that the variable does not have a value.

An example of a Petri net with data $DPN = (PN, V, U, R, W, G)$, where $PN = (P, T, F, M_{init}, M_{final}, l)$ is presented in Fig. 1.

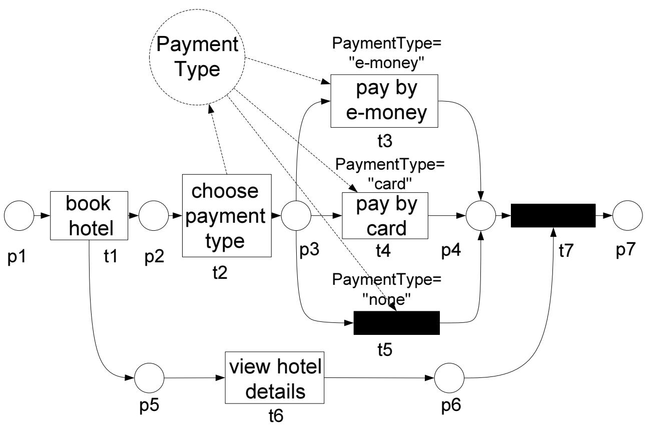


Fig. 1: An example of a Petri net with data.

This Petri net describes a booking process, where the user first books a hotel, chooses the payment type, pays for the reservation (using one of the payment types) or skips the payment, and views the hotel details (before, after, or during the pay-

[2]Note that invisible steps in the process model do not need to be observed in the event log.

ment). Formally, $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, $F = \{(p_1, t_1), (t_1, p_2), (t_1, p_5), (p_2, t_2), (t_2, p_3), (p_3, t_3), (p_3, t_4), (p_3, t_5), (t_3, p_4), (t_4, p_4), (t_5, p_4), (p_4, t_7), (p_5, t_6), (t_6, p_6), (p_6, t_7), (t_7, p_7)\}$, $l(t_1) =$ *"book hotel"*, $l(t_2) =$*"choose payment type"*, $l(t_3) =$*"pay by e-money"*, $l(t_4) =$*"pay by card"*, $l(t_6) =$*"view hotel details"*, for transitions t_5 and t_7 the value of l is not defined. Place p_1 contains one token in the initial marking, formally $M_{init} = [p_1]$, the final marking can be set as $M_{final} = [p_7]$. The set of variables V is represented by a variable *PaymentType*, i.e., $V = \{PaymentType\}$, the set of its possible values is defined as $U(PaymentType) = \{$*"e-money","card","none"*$\}$. Transition t_2 writes the variable *PaymentType*, transitions t_3, t_4, and t_5 read it, formally, $W(t_2) = \{PaymentType\}$, $R(t_3) = R(t_4) = R(t_5) = \{PaymentType\}$. Moreover, guard conditions for t_3, t_4, and t_5 depend on the value of *PaymentType*: $G(t_3)$, $G(t_4)$, and $G(t_5)$ are defined as *"PaymentType="e-money"*, *"PaymentType="card"*, and *"PaymentType="none"* respectively, for other transitions the guard function is not specified. This Petri net can replay 8 traces, represented by distinct sequences of activity names. For instance, $\langle e_1, e_2, e_3, e_4\rangle$, where $act(e_1) =$ *"book hotel"*, $act(e_2) =$ *"choose payment type"*, $act(e_3) =$ *"view hotel details"*, $act(e_4) =$ *"pay by card"*, is an example of such a trace.

3.3 BPMN Modeling Constructs

In this subsection we will introduce BPMN modeling constructs defined on the basis of BPMN 2.0 specification [24]. BPMN offers a wide range of modeling elements, but not all of them are frequently employed [23]. In contrast to the existing formal BPMN semantics [11, 18, 27] in this paper we consider all key BPMN constructs, which cover the main workflow perspectives: control, resource, and data. We will restrict ourselves to *private* BPMN diagrams, which are used to model internal business processes without interaction with the environment. Modeling and discovering of interacting processes is out of the scope of this paper and can be considered as a direction for the future work.

We iteratively introduce and formalize various types of BPMN diagrams. To show their relations with the BPMN standard corresponding meta-models extracted from [24] enumerating all BPMN diagram elements and their relations used in the context of a particular diagram type are presented. Native classes of modeling elements and abstract classes are shown in white and gray respectively. Classes of BPMN diagrams added in this work are highlighted in blue. For each native BPMN class a parent package from [24] is specified.

3.3.1 Core BPMN Models

Core BPMN models are used to formalize flat processes represented by control flow perspective.

A meta-model, which describes elements of core BPMN models, is presented in Fig. 2. It shows various types of *flow nodes*, including *tasks*, *gateways*, and *events*. *Tasks* stand for atomic process steps, *gateways* are used to model routing constructions, *start* and *end events* denote the beginning and completion of the process respectively. The nodes can be connected via directed sequence flows independently of their type.

Graphical notations of BPMN elements used within core models are presented in Fig. 3. Let us define core BPMN models formally. A *core BPMN model* is a tuple $BPMN_{core} = (FN, A, G_{XOR}, G_{OR}, G_{AND}, e_{start}, E_{end}, E_{cancel}, SF, \lambda_A)$, where

- FN is a set of flow nodes,

- $A \subseteq FN$ is a set of activities,[3]
- $G_{XOR}, G_{OR}, G_{AND} \subseteq FN$ are sets of exclusive, inclusive and parallel gateways respectively,
- $e_{start} \in FN$, $E_{end} \subseteq FN$, and $E_{cancel} \subseteq E_{end}$ are a start event, a set of end events, and a set of cancellation end events respectively, such that $E_{end} \setminus E_{cancel} \neq \emptyset$,
- sets A, G_{XOR}, G_{OR}, G_{AND}, $\{e_{start}\}$, E_{end} form a partition of FN,
- $SF \subseteq FN \times FN$ is a set of sequence flows,
- $\lambda_A \in A \to \mathcal{U}_A$, is a labeling function, which maps activities to the universe of activity names.

$\mathcal{U}_{BPMN_C}$ denotes the universe of core BPMN models.

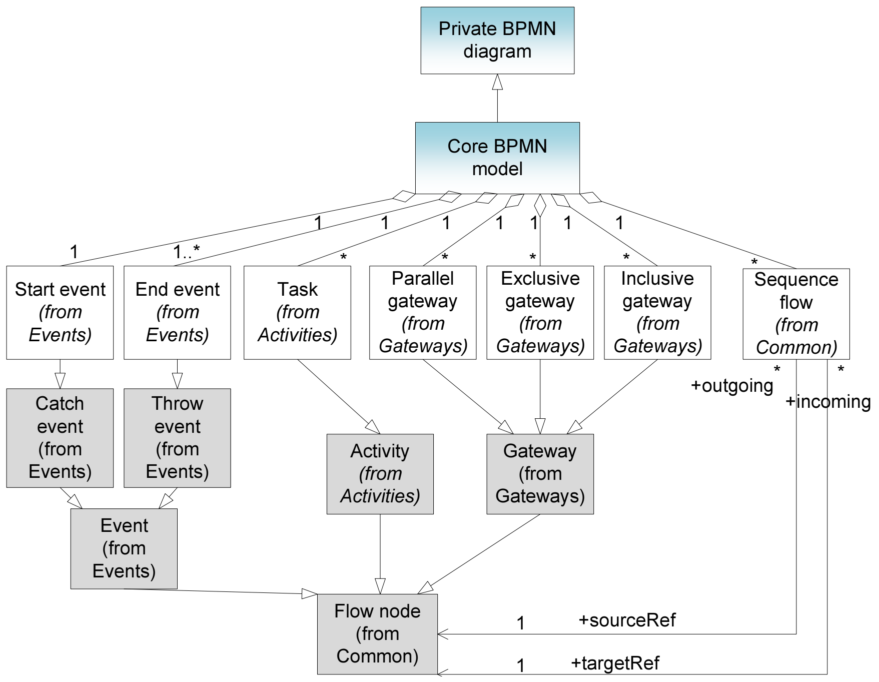

Private BPMN diagram
Core BPMN model
1
1..*
1
1
1
1
1
1
1
*
*
*
*
*
Start event (from Events)
End event (from Events)
Task (from Activities)
Parallel gateway (from Gateways)
Exclusive gateway (from Gateways)
Inclusive gateway (from Gateways)
Sequence flow (from Common)
+outgoing
*
*
+incoming
Catch event (from Events)
Throw event (from Events)
Activity (from Activities)
Gateway (from Gateways)
Event (from Events)
Flow node (from Common)
1
+sourceRef
1
+targetRef

Fig. 2: A meta-model for core BPMN models.

We will restrict core BPMN models to be oriented graphs with one start and multiple end events, where the start event and end events do not have incoming and outgoing sequence flows respectively, and each node of the graph lies on a path from the start to an end event.

Similarly to Petri nets, core BPMN models have an operational semantics based on the model states (or markings). In each state, *sequence flows* (Fig. 3h) may carry *tokens*. In the initial state each outgoing sequence of a *start event* (Fig. 3a) contains a token, while other sequence flows do not. Each node (except start event) can be enabled and may fire.[4] *Activities* (Fig. 3d) and *exclusive gateways* (Fig. 3f) are enabled if at least one of the incoming sequence flows contains a token. When an activity fires it takes a token from one of the incoming sequence flows and adds a token to each outgoing sequence flow. While an exclusive gateway consumes a token from one of the incoming sequence flows and passes a token

[3]We will call "tasks" more generally "activities" in order to extend their semantics in the case of hierarchical BPMN models.

[4]Note that in order to avoid multiple firings of the start event, we assume that it can't be enabled.

to one of the outgoing sequence flows. A *parallel gateway* (Fig. 3e) is enabled only if each of the incoming sequence flows contains at least one one token. When an enabled parallel gateway fires, it takes a token from each incoming sequence flow and produces a token to each outgoing sequence flow. The semantics of *inclusive gateways* (Fig. 3g) is *non-local*. An inclusive gateway fires if some of the incoming sequence flows contain tokens and it is not possible to reach a marking from the current marking, in which currently empty incoming sequence flow will contain tokens, without firing this gateway. Inclusive gateway produces tokens for some of the outgoing sequence flows.

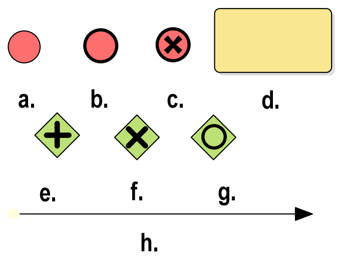


Fig. 3: Elements of core BPMN model: a. start event, b. end event, c. cancellation end event, d. task, e. parallel gateway, f. exclusive gateway, g. inclusive gateway, h. sequence flow.

End event (Fig. 3b) consumes all the tokens as they arrive. Beyond ordinary end events we also consider *cancellation end events* (Fig. 3c), which terminate the entire process, consuming all the tokens from its sequence flows. The BPMN notation contains a wide range of event constructs, the semantics of which involves cancellation. These could be error, signal, cancel, and other types of events. In this paper we combine all of them together conceptually as one type called *cancel* events.

3.3.2 BPMN Models with Data

In this subsection we will extend core BPMN modeling constructs by adding the data perspective. As Fig. 4 shows a BPMN model with data may contain *data objects*. Activities, which read or write data are connected with corresponding data objects via input or output *data associations* respectively. Fig. 5 shows a graphical representation of a data object and a data association.

Also a BPMN model with data may incorporate *conditional expressions*. The values of condition expression are calculated on the basis of data object values and define conditions for passing tokens to the corresponding sequence flows.

Despite the fact that according to the meta-model (Fig. 4) *default sequence flows* can be used within core BPMN models, we will add them to BPMN models with data only, since without conditional expressions default sequence flows do not influence the model execution.

In contrast to the BPMN specification (Fig. 4), where for any arbitrary sequence flow a corresponding *condition expression* can be determined, we will assume that condition expressions are set only for outgoing sequence flows of exclusive and inclusive gateways.

Formally, *BPMN model with data* is a tuple $BPMN_{data} = (BPMN_{core}, DO, DA, Expr, SF_{default})$, where:

- $BPMN_{core} = (FN, A, G_{XOR}, G_{OR}, G_{AND}, e_{start}, E_{end}, E_{cancel}, SF, \lambda_A)$ is a core BPMN model,
- DO is a set of data objects,
- $DA \subseteq (DO \times A) \cup (A \times DO)$ is a set of input and output data associations,
- $Expr \in (SF \cap ((G_{XOR} \cup G_{OR})) \times FN) \nrightarrow U_{Expr}$, where U_{Expr} is a universe of conditional expressions, is a function, which defines conditional expressions for some of the outgoing sequence flows of exclusive and inclusive gateways,

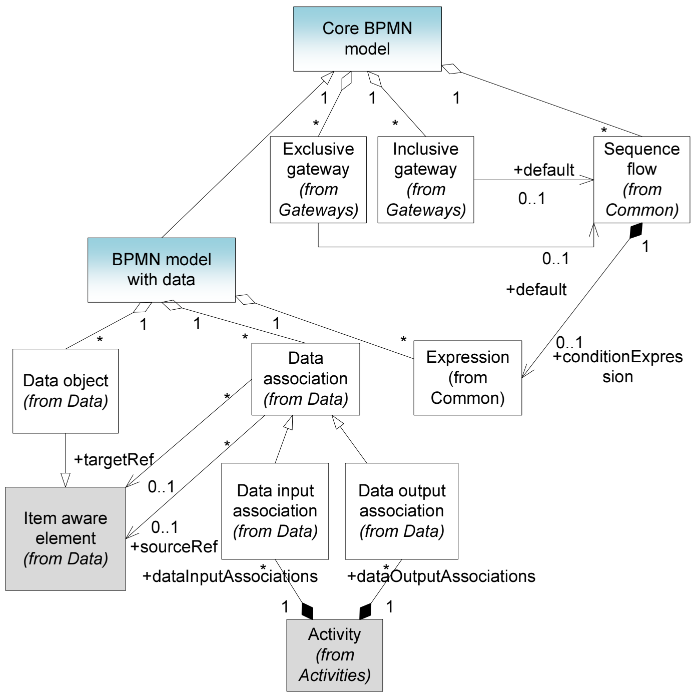

Core BPMN model
Exclusive gateway (from Gateways)
Inclusive gateway (from Gateways)
Sequence flow (from Common)
+default
0..1
0..1
+default
1
BPMN model with data
Data object (from Data)
Data association (from Data)
Expression (from Common)
0..1
+conditionExpression
+targetRef
0..1
Item aware element (from Data)
0..1
+sourceRef
Data input association (from Data)
Data output association (from Data)
+dataInputAssociations
+dataOutputAssociations
Activity (from Activities)

Fig. 4: A meta-model for BPMN models with data.

– $SF_{default} \in (G_{XOR} \cup G_{OR} \to FN)$ is a function, which sets default sequence flows mapping each exclusive or inclusive gateway $g \in (G_{XOR} \cup G_{OR})$ to its outgoing sequence flow $(g, n) \in SF$, $n \in FN$.

By $\mathcal{U}_{BPMN_D}$ we will denote the universe of BPMN models with data. Function $D_{core} : \mathcal{U}_{BPMN_D} \to \mathcal{U}_{BPMN_C}$ defines the underlying core BPMN model for each BPMN model with data.

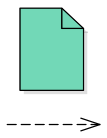


Fig. 5: Data object, data association.

An exclusive gateway passes a token to one of the outgoing sequence flows, for which condition expression is not defined or evaluates to *true*. Inclusive gateway produces tokens for all outgoing sequence flows, those conditional expression are not defined or *true*. If conditional expressions of all outgoing sequence flows evaluate to *false*, then a token is added to a *default sequence flow* of the exclusive or inclusive gateway.

3.3.3 BPMN Models with Resources

Resource is a business entity, which executes or is responsible for business process activities. These could be programs, human beings, departments, or even organizations. Usually, in private BPMN models resources are represented as lanes.[5] An example of a BPMN model with lanes is presented in Fig. 6.

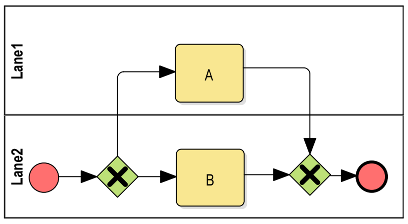


Fig. 6: An example of a BPMN model with lanes.

A meta-model for BPMN models with resources is shown in Fig. 7. As one may see from this meta-model, each lane belongs to a lane set, which in turn can be contained by a lane. Here we will consider only one level of granularity. Lanes may contain flow nodes, such as activities, gateways, events. Note that sequence flows may cross lane's boundaries.

Now let us give a formal definition of BPMN models with resources, which incorporates a set of lanes and a mapping function.

BPMN model with resources is a tuple $BPMN_{res} = (BPMN_{core}, Lanes, map)$, where

- $BPMN_{core} = (FN, A, G_{XOR}, G_{OR}, G_{AND}, e_{start}, E_{end}, E_{cancel}, SF, \lambda_A)$ is a core BPMN model,
- $Lanes$ is a set of lanes representing resources,
- $map : FN \nrightarrow Lanes$ is a partial function which maps some of the nodes onto set of lanes.

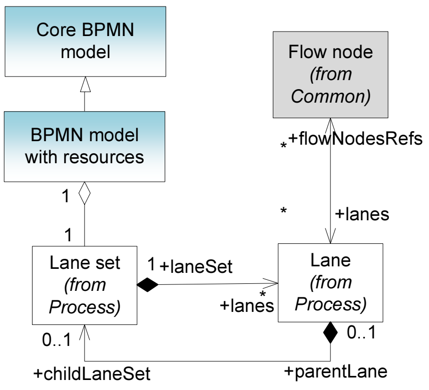


Fig. 7: A meta-model for BPMN models with resources.

$\mathcal{U}_{BPMN_R}$ denotes the universe of BPMN models with resources, function $R_{core} : \mathcal{U}_{BPMN_R} \rightarrow \mathcal{U}_{BPMN_C}$ specifies underlying core BPMN models for BPMN models with resources.

3.3.4 Hierarchical BPMN Models

A hierarchical BPMN model represents a nested structure of a process by adding subprocesses and intermediate cancellation events (Fig. 8).

As it follows from Fig. 8 a subprocess is an activity, which considered as a container with inner flow nodes, such as start/end events, tasks, inner subprocesses and gateways. Thus, subprocesses can be represented as core BPMN models.

The behavior of hierarchical BPMN models builds on the behavior of core BPMN models and extends their semantics in execution of *non-task* activities, i.e., subprocesses.

Each subprocess can be activated if and only if one of the incoming sequence flows contains a token and there are no tokens inside the subprocess and its child subprocesses. An activated subprocess consumes a token from an incoming sequence flow and produces a token to each outgoing sequence flow of the inner start event. If the subprocess terminates normally (all tokens were consumed by non-cancellation end events), then a token is passed to each regular outgoing sequence flow. In case a cancellation end event terminates the

[5] Despite the fact that an assignment of lanes is not strictly defined in the BPMN 2.0 specification, most frequently they are used to model resources.

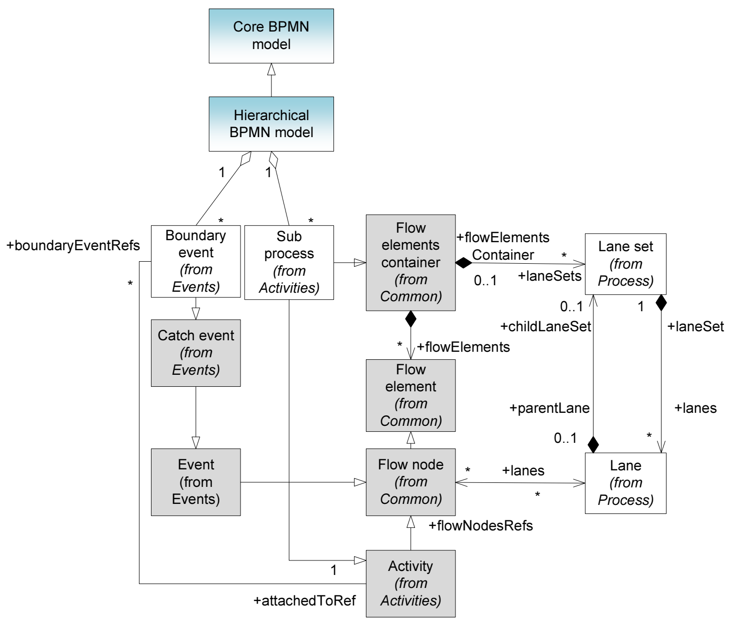


Fig. 8: A meta-model for hierarchical BPMN models.

subprocess, then a control is passed to an outgoing sequence flow marked by a corresponding *boundary event*. We will assume that boundary events are attached to subprocesses only. Namely, we will not consider boundary events attached to tasks.

An example of a subprocess is presented in Fig. 9. The end cancellation event and the corresponding boundary event are marked with an additional “x” sign.

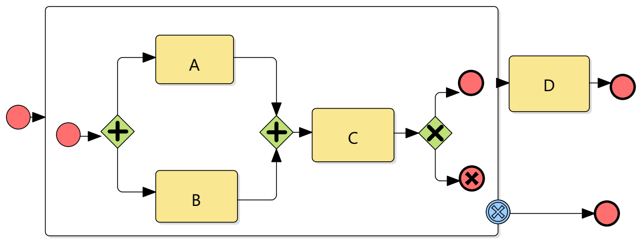


Fig. 9: An example of a BPMN model with subprocesses.

Now let us define hierarchical models formally. A *hierarchical BPMN model* is a tuple: $BPMN_h = (BPMN_{SubProc}, BPMN_0, H, ref, cancel)$, where

– $BPMN_{SubProc} = \{BPMN_1, ..., BPMN_m\}$[6] is a set of subprocesses, presented by core BPMN models,

[6]Let $A = A_0 \cup A_1 \cup ... \cup A_m$, $SF = SF_0 \cup SF_1 \cup ... \cup SF_m$, and $E_{cancel} = E_{cancel_0} \cup E_{cancel_1} \cup ... \cup E_{cancel_m}$, where A_0, A_1,..., A_m, SF_0,SF_1,..., SF_m, and E_{cancel_0},E_{cancel_1},..., E_{cancel_m} are sets of activities, sequence flows, and cancellation events of core models $BPMN_0$,$BPMN_1$,...,$BPMN_m$ respectively. Note that these sets are assumed to be mutually disjoint.

- $\mathit{BPMN_0}$ is a core BPMN model, which represents the root process model, such that $\mathit{BPMN_0} \notin \mathit{BPMN_{SubProc}}$,
- $H : \mathit{BPMN_{SubProc}} \times (\mathit{BPMN_{SubProc}} \cup \{\mathit{BPMN_0}\})$ is a tree relation, which specifies a parent model for each model from $\mathit{BPMN_{SubProc}}$, where $\mathit{BPMN_0}$ acts as a root of this tree,
- $\mathit{ref} : A \nrightarrow \mathit{BPMN_{SubProc}}$, is an injective function, such that $\forall i \in \overline{0,m} \forall j \in \overline{1,m}$ the following condition: $\exists a \in A_i : \mathit{ref}(a) = \mathit{BPMN_j}$ iff $(\mathit{BPMN_j}, \mathit{BPMN_i}) \in H$, holds,
- $\mathit{cancel} : E_{\mathit{cancel}} \to \mathit{SF}$ is an injective function, which maps cancellation end events of each $\mathit{BPMN_{child}} \in \mathit{BPMN_{SubProc}}$ to outgoing sequence flows of activity a marked by corresponding boundary events, where $\mathit{ref}(a) = \mathit{BPMN_{child}}$.

3.3.5 Integrated BPMN Models

Since the aim of this work is to discover integrated BPMN models, consisting of various perspectives (Fig. 10), we provide the following definition.

An *integrated BPMN model* is a tuple $\mathit{BPMN_i} = (\mathit{BPMN_h}, \mathcal{F_D}, \mathcal{F_R})$, where

- $\mathit{BPMN_h} = (\mathit{BPMN_{SubProc}}, \mathit{BPMN_0}, H, \mathit{ref}, \mathit{cancel})$ is a hierarchical BPMN model,
- $\mathcal{F_D} \in (\mathit{BPMN_{SubProc}} \cup \{\mathit{BPMN_0}\}) \nrightarrow \mathcal{U}_{\mathit{BPMN_D}}$ is a function, which maps core BPMN models to BPMN models with data, such that if $\mathcal{F_D}(\mathit{BPMN_{core}}) = \mathit{BPMN_{data}}$, then $D_{\mathit{core}}(\mathit{BPMN_{data}}) = \mathit{BPMN_{core}}$, i.e., $\mathit{BPMN_{data}}$ extends $\mathit{BPMN_{core}}$,
- $\mathcal{F_R} \in (\mathit{BPMN_{SubProc}} \cup \{\mathit{BPMN_0}\}) \nrightarrow \mathcal{U}_{\mathit{BPMN_R}}$ maps core BPMN models to BPMN models with resources, if $\mathcal{F_R}(\mathit{BPMN_{core}}) = \mathit{BPMN_{res}}$, then $R_{\mathit{core}}(\mathit{BPMN_{res}}) = \mathit{BPMN_{core}}$, i.e., $\mathit{BPMN_{res}}$ incorporates $\mathit{BPMN_{core}}$ and extends it with resources.

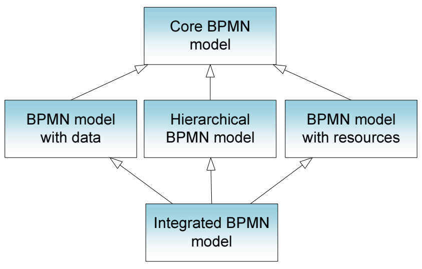


Fig. 10: A meta-model for integrated BPMN models.

As it follows from the definition, each core BPMN model, even if it represents a subprocess, can be extended by both resources and data. Each lane set may belong to a flow elements container, which is represented by a process or a subprocess (Fig. 8). That means, each lane set may be contained by a subprocess or a process itself. That also holds for data, since (according to the BPMN specification) each subprocess may have its own variables.

4 A Framework for Discovering Integrated BPMN Models

4.1 Transforming Flat Process Models to BPMN

Flat process models, such as Petri nets, causal nets and process trees can be obtained from event logs by using existing process discovery techniques.

In this subsection approaches for the transformation of flat process models to BPMN, used as a basis for the discovering technique presented in this paper, are introduced by examples. Their detailed formal description can be found in [16]. Moreover, these approaches were implemented as plugins [17] for ProM [13]. ProM is an open-source framework for developing process mining algorithms.

4.1.1 Converting labeled Petri nets

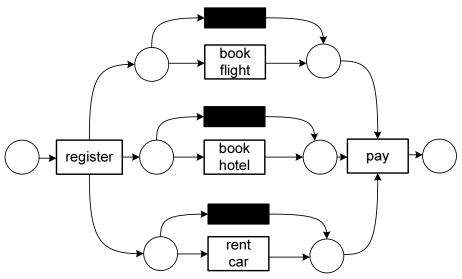


Fig. 11: A labeled Petri net of a booking process.

As an example consider the labeled Petri net which models a simple booking process in Fig. 11. In this process people use an information system to register, book a flight, a hotel, rent a car, and pay. Labeled transitions represent actions, while transition highlighted in black are invisible.

According to the Petri net semantics, users can perform booking actions in any order, moreover, they can skip some of the actions (in that case the corresponding invisible transition is fired).

This labeled Petri net can be automatically transformed to a BPMN model (Fig. 12), represented by core elements only, using the existing transformation algorithm [16]. This algorithm converts a labeled Petri net $PN = (P, T, F, M_{init}, M_{final}, l)$ to a core BPMN model $BPMN_{core} = (FN, A, G_{XOR}, G_{OR}, G_{AND}, e_{start}, E_{end}, E_{cancel}, SF, \lambda_A)$ in such a way that for each visible transition $t \in T$ exists one and only one activity $a \in A$ with the same label, i.e., $l(t) = \lambda_A(a)$. It was proven [16] that the target core BPMN model has the same behavior as the initial labeled Peri net. Moreover, the target BPMN model is a connected graph with nodes lying on paths from the start event to an end event.

The resulting core BPMN model (Fig. 12) complies with the meta-model presented in Fig. 2. It contains activities, start/end events, exclusive and parallel gateways, and sequence flows. Note that sequence flows can connect arbitrary flow nodes, thus, invisible transitions do not need to be added as activities to Fig. 12 because of the explicit gateways.

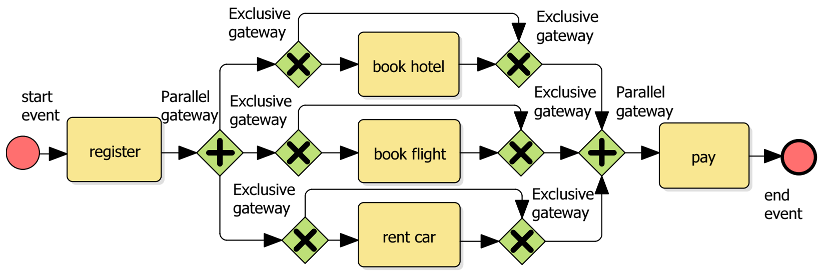


Fig. 12: A BPMN model constructed from the labeled Petri net presented in Fig. 11.

4.1.2 Converting causal nets

Now let us consider an example of a causal net (Fig. 13), which models the booking process. Causal nets are represented by activities and their input and output bindings, which stand for pre- and post-conditions of these activities. In this particular example, activity *register* can be followed by any combination of the booking activities from the set $\{book\ flight, book\ hotel, rent\ car\}$. In other words activity *register* can be followed by all of the booking activities, or just two of them, or only one of booking activities. Similarly, any combination of the booking activities precede the *pay* activity.

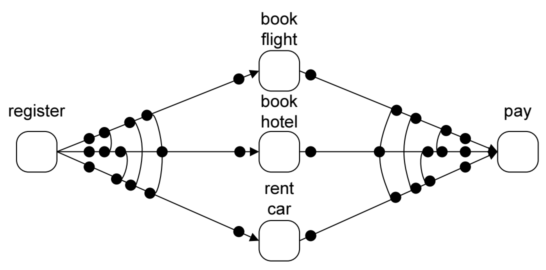


Fig. 13: A causal net, which models the booking process.

Each activity of a causal net can be transformed to a corresponding BPMN activity.

Exclusive and parallel gateways can be used to model bindings. If an activity has several input or output bindings a corresponding exclusive gateway is added. Parallel gateways are used model bindings consisting of multiple elements. In contrast to exclusive and parallel gateways inclusive gateways compactly represent bindings within BPMN models.

The causal net presented in Fig. 13 can be transformed to a BPMN model with inclusive gateways (Fig. 14), which naturally model overlapping input and output bindings.

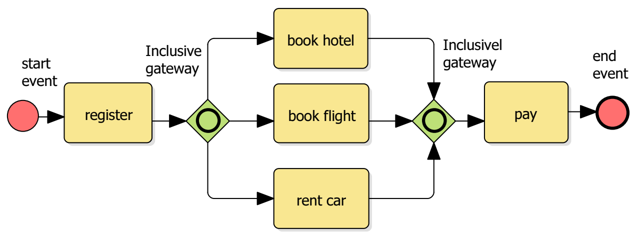


Fig. 14: A BPMN model with inclusive gateways constructed form the causal net presented in Fig. 13.

4.1.3 Converting process trees

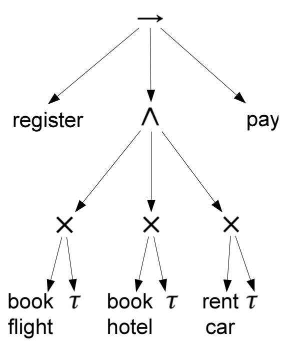


Fig. 15: Process of a booking process.

Process trees (Fig. 15) are yet another formalism for process modeling. They are often obtained as a result of applying process discovery algorithms (e.g. Inductive miner [19] or Genetic miner [6]). They were proposed in [19] and defined as directed trees with root, branch and leaf nodes. Each branch node is considered to be an operator node, leaf nodes stand for atomic activities, while a root node denotes an entire process model.

Process tree is defined inductively:

- $a \in \mathcal{U}_A \cup \{\tau\}$ is a process tree, representing an atomic activity (τ denotes the silent activity);
- Let M_1,...M_n, where $n \geq 1$ be process trees, and $\bigoplus$ be a process tree operator, then $\bigoplus(M_1, ...M_n)$ is a process tree.

There can the following types of process tree operators: $\times$ denotes the exclusive choice between one of the subtrees, $\longrightarrow$ is a sequential execution of all subtrees, $\circlearrowleft$ means the structured loop, where M_1 is a loop body and M_2, ..., M_n are alternative loop back paths ($n \geq 2$), $\bigwedge$ denotes parallel execution of all subtrees.

An example of a process tree shown in Fig. 15 is inductively converted to a labeled Petri net presented earlier in Fig. 11. Note that any process tree can be converted to a corresponding Petri net, while the opposite is not always true, since there is no guarantee that a given Petri net is structured.

4.2 Converting Petri Nets with Data to BPMN Models with Data

In this subsection we will introduce an approach for transforming Petri nets with data to data BPMN models. Note that this approach can be extended to other process models, such as causal nets and process trees enhanced with data.

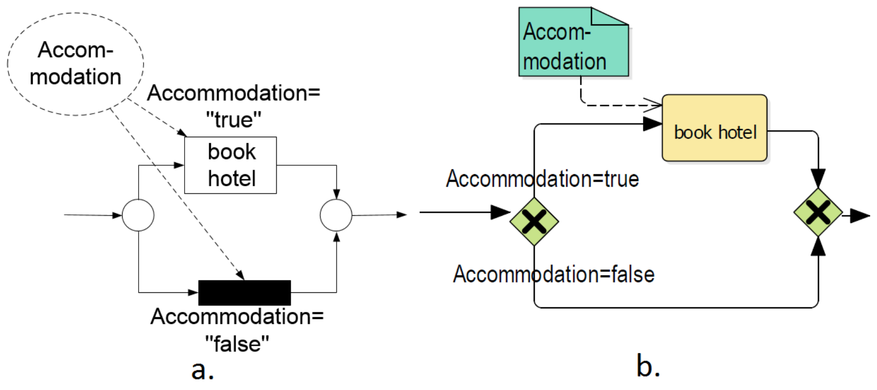


Fig. 16: Conversion of data objects, data associations and guards.

Suppose that $DPN = (PN, V, U, R, W, G)$ is an initial Petri net with data. In order to transform this model to a target BPMN model with data $BPMN_{data} = (BPMN_{core}, DO, DA, Expr, SF_{default})$, where $BPMN_{core} = (FN, A, G_{XOR}, G_{OR}, G_{AND}, e_{start}, E_{end}, E_{cancel}, SF, \lambda_A)$, the following steps are to be performed:

1. labeled Petri net $PN = (P, T, F, M_{init}, M_{final}, l)$ is converted to a core BPMN model $BPMN_{core}$, using the algorithm described in [16] (without removing activities, which correspond to invisible transitions); by $M_A : T \to A$, we denote a function, which maps transitions to activities;
2. for each variable from V a data object do is created and added to the target BPMN model, the mapping of variables to data objects is defined as a function $M_D : V \to DO$;
3. for each non-invisible $t \in T$, for each $v \in R(t)$, a data input association $(M_D(v),M_A(t))$ is added to DA; similarly, for each $t \in T$, for each variable v from $W(t)$, a data output association $(M_A(t),M_D(v))$ is created and added to DA;
4. for each $t \in T$ with a guard $G(t)$ an expression for each incoming sequence flow of activity $M_A(t)$ is set to $G(t)$, if the source node of this sequence flow is an exclusive or inclusive gateway, [7] default condition expressions are chosen in an arbitrary way;

[7] According to the guard mining algorithm presented in [22] guards are specified for those transitions only, preceding places of which form decision points. Thus, these places will be transformed to exclusive BPMN gateways using existing conversion algorithms.

5. each activity corresponding to an invisible transition is removed from the target BPMN diagram along with incoming and outgoing sequence flows in accordance with the simplification technique presented in [16]; all outgoing sequence flows of exclusive gateways added instead of removed sequence flows inherit condition expressions.

Variables, read and write functions of a data Petri net are trivially transformed to data objects, input and output data associations respectively.

To illustrate the conversion of guards let us consider a fragment of a Petri net presented in Fig. 16 a. and a corresponding fragment of a data BPMN model shown in Fig. 16 b. This example shows that each guard is transformed to condition expressions of corresponding sequence flows.

4.3 Enhancing Core BPMN Models by Adding Resource Perspective

In this subsection an approach for the enhancement of core BPMN models by adding resources will be introduced. The enhancement algorithm takes a core BPMN model $BPMN_{core} = (FN, A, G_{XOR}, G_{OR}, G_{AND}, e_{start}, E_{end}, E_{cancel}, SF, \lambda_A)$, an event log $L = (E, Tr, act, attr)$ as input parameters and enhances $BPMN_{core}$ with resources by specifying $map : FN \nrightarrow Lanes$ function, which maps tasks and other elements to lanes, producing a BPMN model $BPMN_{res} = (BPMN_{core}, Lanes, map)$, where $Lanes$ is a set of resources.

The algorithm is illustrated using an example.[8] Let us consider a fragment of the event log $L = (E, Tr, act, attr)$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $Tr = \{\langle e_1, e_2, e_4\rangle, \langle e_3, e_5, e_6\rangle\}$, presented in Tab. 1. For each event a resource performing this event is specified. For example, $attr(e_1)("Resource") = attr(e_2)("Resource") = "John"$, $attr(e_3)("Resource") = "Mary"$, etc. The initial set of resources, which perform activities is defined as: $R = \cup_{e \in E} \{attr(e)("Resource")\}$. Thus, $R = \{"John", "Mary", "Kate", "Jane"\}$.

Now let us consider a core BPMN model discovered from this log (Fig. 17), using the Inductive mining algorithm [20] and the Petri net to BPMN conversion technique presented in Section 4.1.

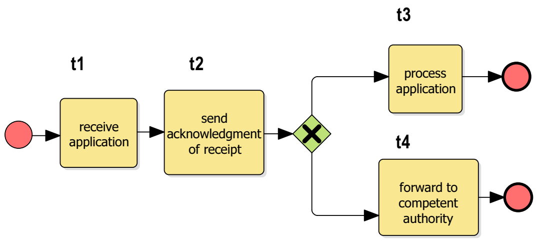

t1
receive application
t2
send acknowledgment of receipt
t3
process application
t4
forward to competent authority

Fig. 17: A core BPMN model without lanes.

The approach we apply for the further model enhancement [8] assigns each task of a BPMN model to an aggregate resource (lane), while corresponding events in a log may be associated with different resources. First, we consider pairs of source and target tasks, which are connected via sequence flow (with possible intermediate gateways), such that for at least one of them $t \in A$ exists a corresponding event $e \in E$, for which

[8]For the detailed description of the approach please refer to [8].

$act(e) = \lambda_A(t)$ and $attr(e)("Resource")$ is defined.[9] In our example these pairs are: $(t_1, t_2), (t_2, t_3), (t_2, t_4)$. Then for each such a pair we define *degree of no handover*: $w_{t_i t_j} = \frac{|\mathcal{U}^{t_i}_{t_i \to t_j}(L) \cap \mathcal{U}^{t_j}_{t_i \to t_j}(L)| + |\sigma_{=}(\mathcal{U}_{t_i \to t_j}(L))|}{|\mathcal{U}^{t_i}_{t_i \to t_j}(L)| + |\mathcal{U}^{t_j}_{t_i \to t_j}(L)|}$, where $\mathcal{U}^{t_i}_{t_i \to t_j}(L)$ is a multiset over R, such that $r \in R$ belongs to it n number of times, where $n = |\{e_i \in E | \exists tr \in Tr, \exists e_j \in E, \exists \alpha, \beta \in E^* : tr = \alpha \cdot \langle e_i \rangle \cdot \langle e_j \rangle \cdot \beta \wedge act(e_i) = \lambda_A(t_i) \wedge act(e_j) = \lambda_A(t_j) \wedge attr(e_i)("Resource") = r\}|$, i.e., it specifies the number of times task t_i followed by t_j will be executed by resource r. Similarly, $\mathcal{U}^{t_j}_{t_i \to t_j}(L)$ defines the number of times task t_j preceded by t_i will be executed by resource r, and $\sigma_{=}(\mathcal{U}_{t_i \to t_j}(L))$ specifies the number of times tasks t_i and t_j, where t_j is preceded by t_i, are both executed by resource r.

Thus, $w_{t_1 t_2} = 1$, $w_{t_2 t_3} = 0$, and $w_{t_2 t_4} = 0$, hence handover will be defined only for pairs (t_2, t_3), (t_2, t_4). The resulting BPMN model with resources is presented in Fig. 18.

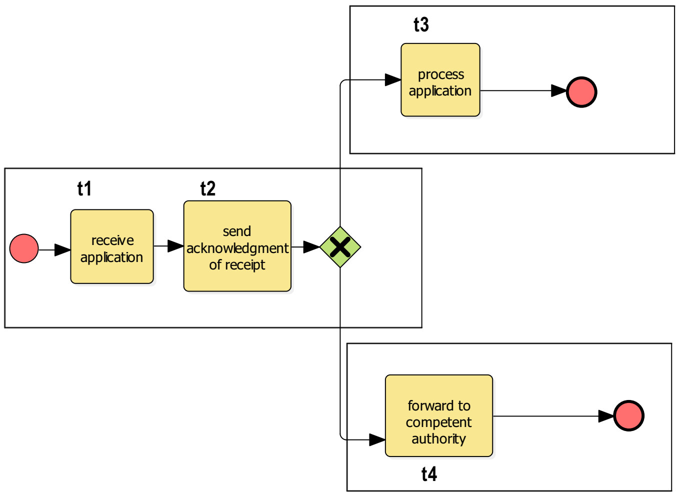

t3
process application
t1
t2
receive application
send acknowledgment of receipt
forward to competent authority
t4

Fig. 18: Enhancing a core BPMN model by adding lanes.

If the *degree of no handover* is not zero or one, decision on whether to add new lanes depends on a predefined threshold value. In the second step of the algorithm [8] lanes with a high degree of common resources are merged. In our example lanes will not be merged, since they do not share any resources. And finally all nodes of other types (gateways, events) are attached to one of the resources of "neighbor" activities.

4.4 Constructing BPMN Models with Subprocesses

Subprocesses can be used to create a process model that has a hierarchical structure. This subsection presents an approach for constructing hierarchical BPMN models with subprocesses using the *localized logs* process discovery technique proposed earlier in [4]. Each event in the event log can be assigned to a so-called *region*. As it will be shown later in Section 6 such event logs can be found in many application domains. Moreover, if there is no

[9]We will assume that all the tasks are uniquely labeled, and thus, there is an unambiguous correspondence between tasks and events.

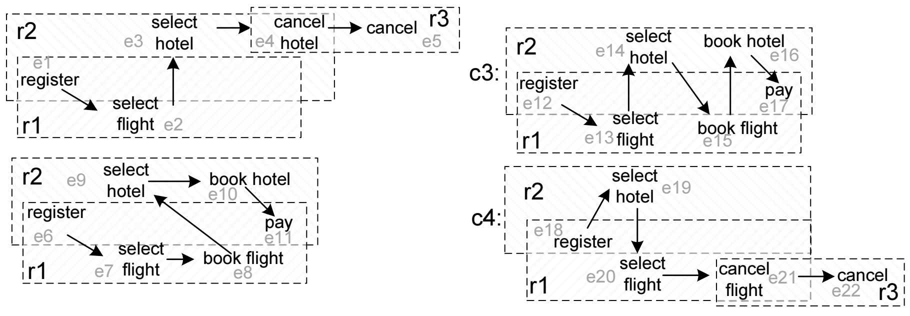


Fig. 19: Localized event log of a booking process. Regions r_1, r_2, and r_3 correspond to booking flight, booking hotel, and cancellation procedures respectively.

information on events' localization, then in most of the cases an event log can be enriched with additional data on a basis of expert knowledge.

Now let us give a definition of a localized event log [4]. A *localized event log* is a triplet $L_L = (L, R, loc)$, where $L = (E, Tr, act, attr)$ is an event log, R is a set of regions (or localizations), and $loc : E \to \mathcal{P}_{NE}(R)$.[10] We will consider only stable localized event logs. A localized event log $L_L = (L, R, loc)$ with $L = (E, Tr, act, attr)$ is called *stable* iff $\forall e_1, e_2 \in E$, such that $act(e_1) = act(e_2)$, holds $loc(e_1) = loc(e_2)$.

An example of a localized event log is presented in Fig. 19. This localized event log contains four traces and three regions. Events with names "select flight", "book flight" and "select hotel", "book hotel" belong to regions $r1$ and $r2$ corresponding to the booking flight and booking hotel procedures respectively. Events named "register", "pay" are attached to both regions $r1$ and $r2$, since these events correspond to both booking procedures. Events labeled by "cancel flight" and "cancel hotel" belong to a specific booking region ($r1$ or $r2$ respectively) and a cancellation region $r3$, while event with the name "cancel" belongs to the cancellation region only.

The localized logs discovery approach performs a construction of a target labeled Petri net from a localized event log $L_L = (L, R, loc)$, $R = \{r_1, ..., r_k\}$, where $L = (E, Tr, act, attr)$, in three steps:

1. For each region $r_i \in R$ a labeled Petri net $PN_i = (P_i, T_i, F_i, l_i, M_{init_i}, M_{final_i})$ is discovered from a projection of the log $L_{\uparrow E_i} = (E_{\uparrow E_i}, Tr_{\uparrow E_i}, act_{\uparrow E_i}, attr_{\uparrow E_i})$ on a set of events $E_i = \{e \in E | loc(e) \in r_i\}$, using one of the existing process discovery techniques in such a way that $\forall t_1, t_2 \in T_r$ if $l_i(t_1) = l_i(t_2)$, then $t_1 = t_2$, i.e., all transitions are uniquely labeled[11];
2. A resulting labeled Petri net $PN_U = (P, T, F, l, M_{init}, M_{final})$ is defined as a union of all discovered labeled Petri nets $PN_1, ..., PN_k$, where transitions with the same labels are merged;

[10] $\mathcal{P}_{NE}(X)$ defines a set of non-empty subsets of X.

[11] Note that most of the discovery methods allow for constructing such models

3. All structurally redundant hanging places and redundant arcs leading to hanging places are removed.

For the description of the localized logs discovery algorithm please refer to [4].

A labeled Petri net constructed from the localized event log (Fig. 19) on the basis of the Inductive mining approach [20] is shown in Fig. 20. All transitions are highlighted in the colors of regions they belong to (if a transition belongs to several regions it is highlighted in several colors).

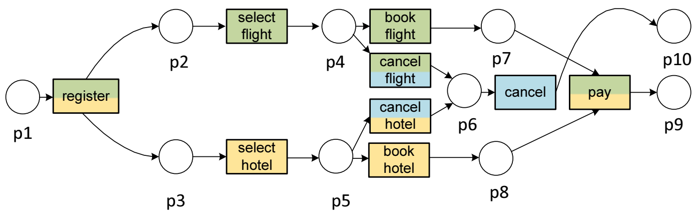


Fig. 20: A labeled Petri net constructed from the localized event log presented in Fig. 19.

After the labeled Petri net is constructed, it is transformed into a core BPMN model using algorithms presented in [16]. Then the core BPMN is converted to a hierarchical BPMN model. For that aim groups af activities corresponding to the same region are identified, and for each such a group dominating and postdominating activities are found (for the formal definitions and an algorithm for finding these activities please refer to [21]). Having dominating and postdominating activities it is verified whether these activities can serve as star and end subprocess's points (it is checked that the subprocess either contains all the activities solely belonging to a region or do not contain any of them, i.e., subprocesses do not intersect and can be only nested into each other). Also it is verified that the subprocess has only one start activity and if there are multiple end activities it is additionally checked that all of them except one correspond to the events marked with a special *cancellation* attribute. In our example we will assume that the events named "cancel" are marked with a *cancellation* attribute.

The result of applying conversions and subprocesses identification techniques to the Petri net (Fig 20) is shown in Fig. 21. This hierarchical BPMN model contains one subpro-

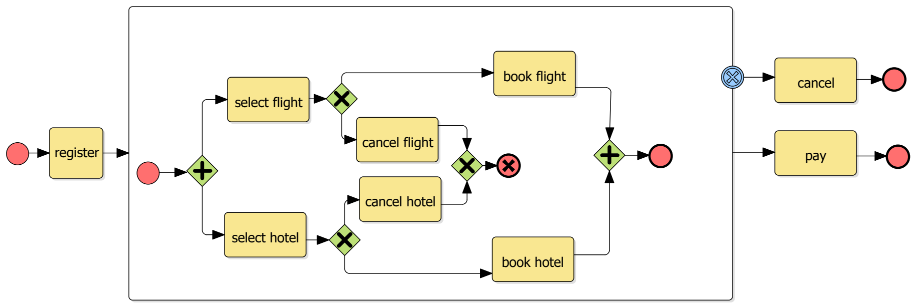


Fig. 21: A hierarchical BPMN model constructed from the labeled Petri net presented in Fig. 20.

cess, which incorporates all the activities solely belonging to $r1$ and $r2$ regions, while the "cancel" activity, solely belonging to the region $r3$, is outside of its boundaries. The subprocess has one start point represented by the "register" activity and two end points represented by the "pay" and "cancel" activities. Since the corresponding events of the "cancel" activity are marked with a *cancellation* attribute the subprocess is connected with the "cancel" activity via an intermediate cancellation event attached to its boundary. Activation of the end cancellation event leads to the termination of the subprocess (all the tokens are removed) and yields a token to the outgoing sequence flow marked with the intermediate cancellation event. If the subprocess terminates normally (all the tokens are consumed by the normal end event), a token is passed to the regular outgoing sequence flow, activating the "pay" activity.

In contrast to the initial Petri net (Fig. 20), where the traces with cancellations can't be replayed (assuming that $[p_9]$ is the final state), this hierarchical BPMN model accepts all the log traces (all the tokens are consumed by the end events). In case one of the booking procedures completes successfully, while the other is canceled, the entire BPMN subprocess will be canceled as well. Moreover, in the hierarchical BPMN model no booking can be performed after the cancellation, while in the labeled Petri net a booking transition can fire after the other booking procedure was canceled. Thus, the hierarchical BPMN model (Fig. 21) describes the booking process accurately than the corresponding Petri net (Fig. 20).

To show the importance of the regions selection we will consider an event log of the same booking process with differently chosen regions (Fig. 22). Note that this event log

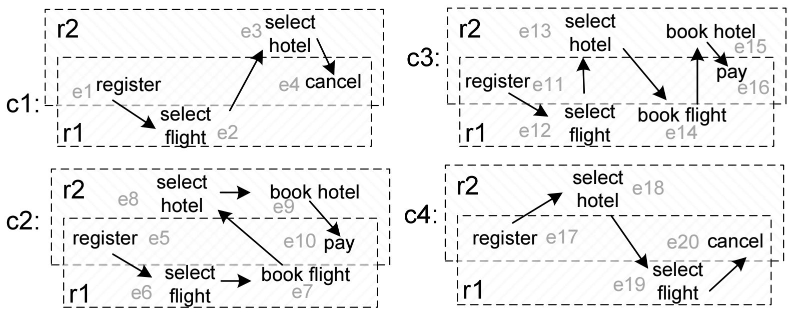


Fig. 22: Localized event log of a booking process. Regions r_1 and r_2 correspond to booking flight and booking hotel procedures respectively.

in contrast to the previous one (Fig. 19) does not contain specific cancellation events, and cancellations are represented by an aggregate cancellation event only. Also note that both localized event logs can be easily constructed one from another during the preprocessing. A Petri net discovered from this event log using the localized logs approach [4] on the basis of the Inductive miner technique [20] is presented in Fig. 23.

This labeled Petri net can replay any trace from the event log (Fig. 22), assuming that $[p_1]$ and $[p_8]$ are the initial and the final markings of this Petri net respectively. In [4] it was proven that: if for each region $r_i \in R$ trace $t_{\uparrow E_i}$ can be replayed in PN_i, then trace t can be replayed by the entire model PN_U. That is holds for the labeled Petri net (Fig. 23), since for each region $r_i \in R$ every trace $t_{\uparrow E_i}$ can be replayed on a corresponding labeled Petri net PN_i.

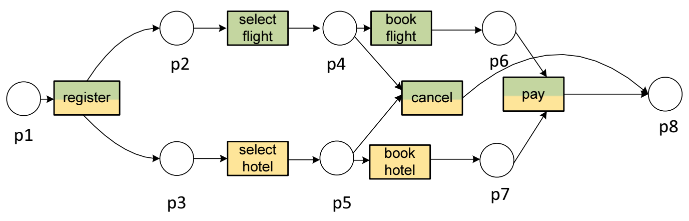


Fig. 23: A labeled Petri net constructed from the localized event log presented in Fig. 22.

A hierarchical BPMN model constructed from the labeled Petri (Fig. 23) is shown in Fig. 24. Again we assume that events named "cancel" are marked with a *cancellation* attribute. This BPMN model contains two booking subprocesses with unique start and two final activities, one of which corresponds to the events marked with a *cancellation* attribute. The BPMN model accepts all the traces of the localized event log (all tokens are consumed by the end event). At the same time in contrast to the initial labeled Petri net (Fig. 23) this model may reach a so-called *dead state*, in which no node is enabled, while some sequence flows contain tokens. This can happen if one of the subprocesses is canceled, while the other finished its execution in a regular way. Hence taking this localized event log (Fig. 22) as a starting point the labeled Petri net (Fig. 23) is more preferable as a model for the booking process description than the corresponding hierarchical BPMN model (Fig. 24).

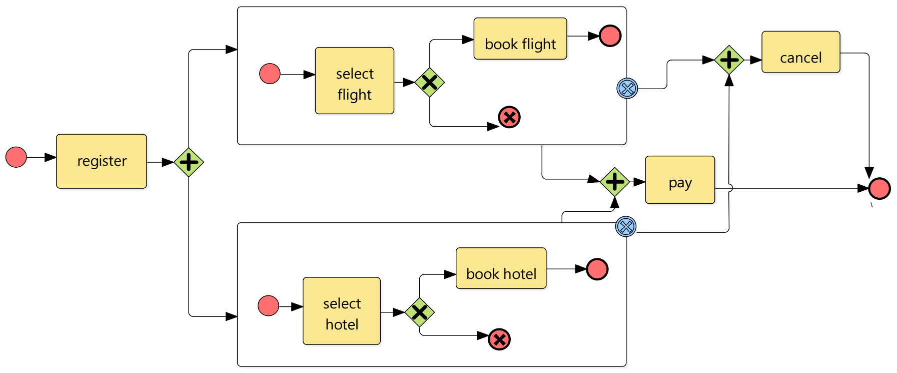


Fig. 24: A hierarchical BPMN model constructed from the labeled Petri net presented in Fig. 23.

Thus, considering these two examples of localized event logs and corresponding process models, one may conclude that characteristics of synthesized labeled Petri nets and hierarchical BPMN models dramatically depend on localization of events.

4.5 Integrated Discovery Approach

This subsection presents an integrated discovery approach for constructing hierarchical multiperspective BPMN models. This approach incorporates all methods introduced above.

The entire schema of the approach is presented in Fig. 25. First, a localized event log L_L is filtered and logs $L_1, ..., L_k$ corresponding to regions are extracted. After that labeled Petri nets $PN_1, ..., PN_k$ are discovered from these event logs using one of the existing discovery techniques.

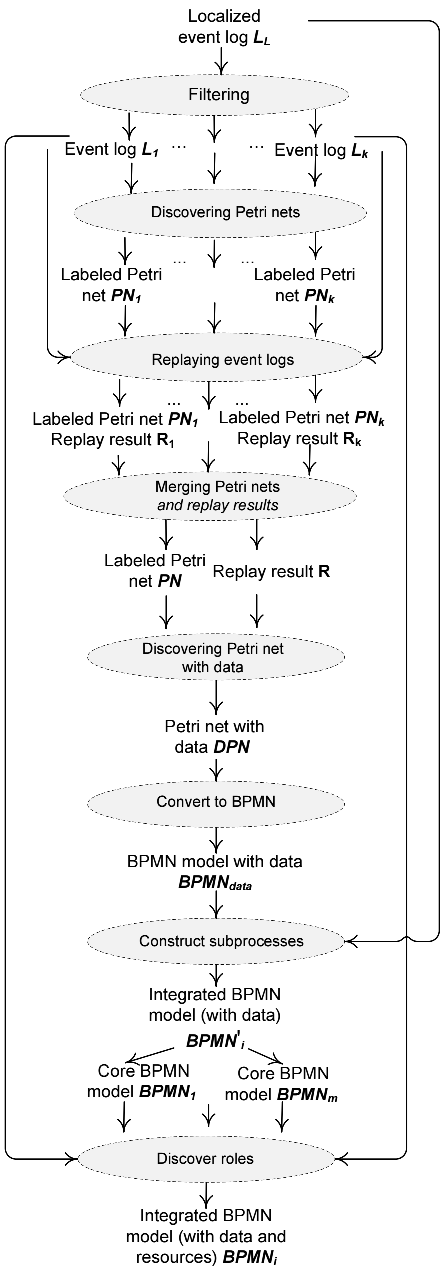

Localized event log L_L
Filtering
Event log L_1 ...
... Event log L_k
Discovering Petri nets
Labeled Petri net PN_1
Labeled Petri net PN_k
Replaying event logs
Labeled Petri net PN_1 Replay result R_1
Labeled Petri net PN_k Replay result R_k
Merging Petri nets and replay results
Labeled Petri net PN
Replay result R
Discovering Petri net with data
Petri net with data DPN
Convert to BPMN
BPMN model with data $BPMN_{data}$
Construct subprocesses
Integrated BPMN model (with data) $BPMN'_i$
Core BPMN model $BPMN_1$
Core BPMN model $BPMN_m$
Discover roles
Integrated BPMN model (with data and resources) $BPMN_i$

Fig. 25: The integrated discovery approach presented in this paper

Then each event log L_i is replayed on a corresponding labeled Petri net PN_i and an alignment R_i (a sequence of replay steps, including synchronous log and model moves, log only and model only moves) is constructed. These labeled Petri nets and alignments are merged to a unified Petri net *PN* and a corresponding alignment R using the techniques presented in [4] and [26] correspondingly.

Next a method for enriching Petri nets with data recorded in the event logs using a corresponding alignment [22] is applied, and as a result a Petri net with data, i.e., *DPN*, is obtained.

Then the Petri net with data is converted to a BPMN model with data $BPMN_{data}$ using existing conversion techniques introduced above and thoroughly presented in [16, 17].

On the basis of localization information contained in the initial event log subprocesses are constructed within $BPMN_{data}$ using Algorithm 1 described above. Although the procedure of constructing subprocesses is defined for core BPMN models, BPMN models with data can also be transformed to integrated BPMN models with subprocesses and data perspective. In that case common data variables are duplicated.

The resulting integrated model $BPMN'_i = (BPMN_h, \mathcal{F}_\mathcal{D}, \mathcal{F}'_\mathcal{R})$ is represented by a hierarchical BPMN model $BPMN_h$, which in its turn contains a set of core BPMN models: $BPMN_0, BPMN_1, ..., BPMN_m$. Function $\mathcal{F}_\mathcal{D}$ defines a data perspective for each of these models.

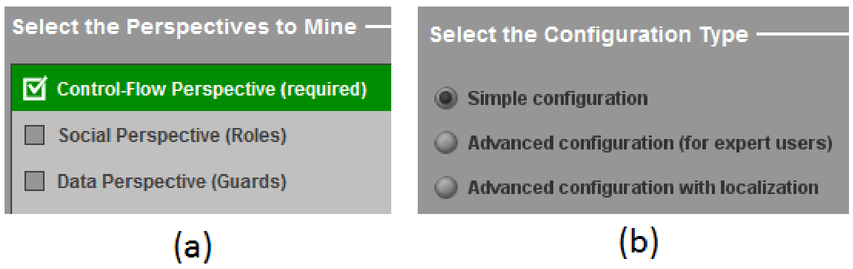
After that the core BPMN models are enriched with resources on the basis of information presented in the corresponding event logs. The target integrated BPMN model can be represented as $BPMN_i = (BPMN_h, \mathcal{F}_{\mathcal{D}}, \mathcal{F}_{\mathcal{R}})$, where function $\mathcal{F}_{\mathcal{R}}$ defines a resource perspective of the core BPMN models.

Note that log partitioning allows to dramatically reduce the total time computational complexity. Log partitioning works especially well for the algorithms of enriching Petri nets with data, since they involve replay techniques, which are known to be time consuming for large models and logs. Moreover, discovery of models from the real-life event logs presented in Section 6 cannot be performed in any reasonable amount of time without log partitioning.

5 Tool Support

The tool called *MultiPerspectiveMiner*[12] was developed as a plugin for ProM (Process mining) framework [13] – an open source extensible platform, widely used for the analysis of event logs. The *MultiPerspectiveMiner* implements the integrated discovery approach and functionally depends on other mining and analysis plugins called during the integrated discovery.

Figure 26 shows screenshots of mining parameters selection. Thus, in Fig. 26(a) the selection of process modeling perspectives is presented. Besides the control flow perspective, which is mandatory for mining, the user can choose data, resources perspectives or both of them. Fig. 26(b) shows configuration types. In case the user chooses *Simple configuration*, the Inductive miner [20] will be selected as an underlying control flow mining algorithm. In *advanced configuration* the user can choose from a variety of control flow mining algorithms. For mining process models with subprocesses the user is to select *advanced configuration with localization*.



Select the Perspectives to Mine

- [✓] Control-Flow Perspective (required)
- [ ] Social Perspective (Roles)
- [ ] Data Perspective (Guards)

Select the Configuration Type

- [✓ Simple configuration]
- Advanced configuration (for expert users)
- Advanced configuration with localization

Fig. 26: Parameters of the Multiperspective Miner

6 Case Studies

In this section we evaluate of the integrated discovery approach presented in the earlier sections. First, we show that our discovery approach can assist in extracting in-depth knowledge from real-life event logs and represent them in terms of convenient BPMN models. Then behavioral and structural characteristics of BPMN models discovered from the real-life event logs are obtained. Finally, using these structural characteristics the discovered BPMN models are compared to the manually created BPMN models from the Signavio model collection.

[12] See https://svn.win.tue.nl/repos/prom/Packages/MultiPerspectiveMiner/

6.1 Discovering Multiperspective BPMN Models

6.1.1 Discovering Municipal Processes

First we took event logs from building permit administrative processes of five Dutch municipalities [12] (containing 1199, 832, 1409, 1053, and 1156 traces respectively) and analyzed them. These event logs contain information on processes managed by an information system and performed by human resources. A fragment of one of the event logs after preprocessing is shown in Table 2. Each row represents an event occurrence and contains a *case id*, an *activity name*, a *timestamp*, a *resource identifier*, a *subprocess name* (derived from an original event code), and a value of additional *question* parameter.

Case ID	**Activity Name**	**Timestamp**	**Resource ID**	**Subprocess name**	**Question**
5772892	application received	2012-09-04 T13:12:34	560912	HOOFD	EMPTY
5772892	send confirmation	2012-09-04 T13:12:36	560912	HOOFD	true
5772892	enter date acknowledgment	2012-09-04 T13:16:20	560912	HOOFD	EMPTY
5772892	forward to the competent authority	2012-09-04 T13:16:24	560912	DRZ	false
5772892	start regular procedure without MER	2012-09-04 T13:16:24	560912	BPT	true
...	...	...	...	...	...
5772892	publish document	2012-10-23 T15:48:36	560890	HOOFD	true

Table 2: Event log of a Dutch municipality processes.

This fragment of the log describes one case of the building permit process. First, resource *560912* receives an application, then sends a confirmation of receipt and enters a date of acknowledgment (all these activities are performed within the main subprocess *HOOFD*), after that within *DRZ* subprocess resource *560912* decides not forward the application to the competent authority (note that the value of 'question' parameter is set to *false*), and finally he or she starts a regular procedure of application processing without MER (assessment of the impact on the environment) within *BPT* subprocess. After several steps of the application processing another resource *560890* publishes a document with a final decision.

The result of applying the integrated discovery approach to one of the event logs is presented in Fig. 27.

The resulting BPMN model describes building permit process and contains 13 subprocesses with control flow obtained on the basis of the Inductive miner [20] using *Subprocess* attribute as a localization information for [4] approach. The data and resource perspectives were discovered by [22] and [8] algorithms respectively. Constructing resources within subprocesses allowed to build a more detailed diagram and significantly reduce time costs for the resource discovery. Moreover, the division of the model into subprocesses made it possible to apply the data perspective mining [22] (which relies on the model and log align-

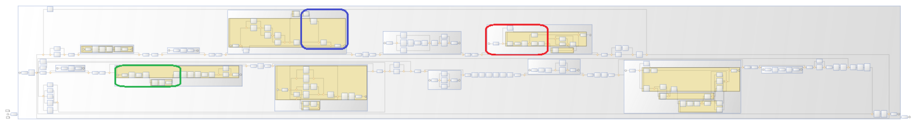


Fig. 27: An entire BPMN model obtained by the integrated discovery approach from the Dutch municipality event log. All the fragments of subprocesses described later in this section are marked with boxes of corresponding colors.

ment and thus known to be possibly time consuming as logs and models get bigger), using the divide and conquer approach [2].

Now let us consider the discovered BPMN model in details. A fragment of the *BPT* subprocess is presented in Fig. 28.

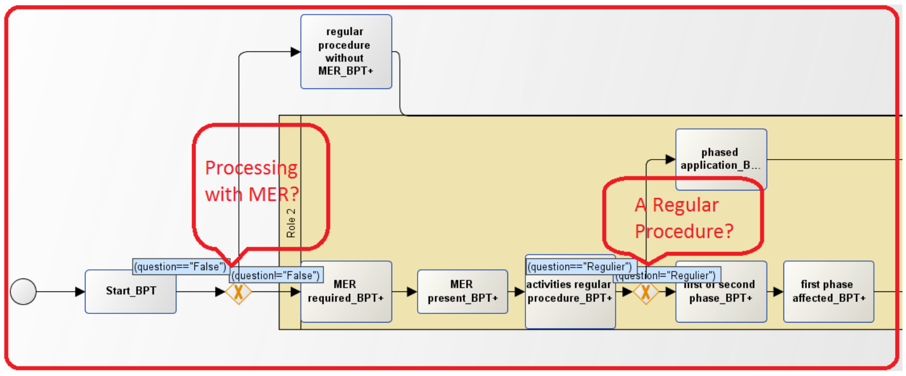


Fig. 28: A fragment of the *BPT* subprocess

As it follows from the diagram the decision whether or not to process the application with MER (assessment of the impact on the environment) depends on the value of the *question* data variable. This dependency was automatically discovered and represented within a diagram. According to the log (Tab. 2) that choice is made (the variable assigned a value) in the previous process step, when the performer decides if the application should be forwarded to the competent authority. The other exclusive choice gateway (Fig. 28) has a guard depending on the value of the *question* data variable as well. Cases are being split according to the type (*Regular* or not) of the application processing procedure. Note, that just like in the previous case the value of the variable is defined in the preceding step (here it is defined by *activities regular procedure* task).

Another fragment of the diagram, containing *OPS* subprocess, is presented in Fig. 29. It illustrates the applicability of the resource discovery approach. Subprocess OPS describes a procedure of suspending the application and is performed by two roles. As it follows from the diagram, *Role 2* is responsible for technical processing steps, such as registration of suspending, finding a reason for suspending and forwarding the application to the competent authority. While *Role 1* is a role of a competent authority, who defines terms.

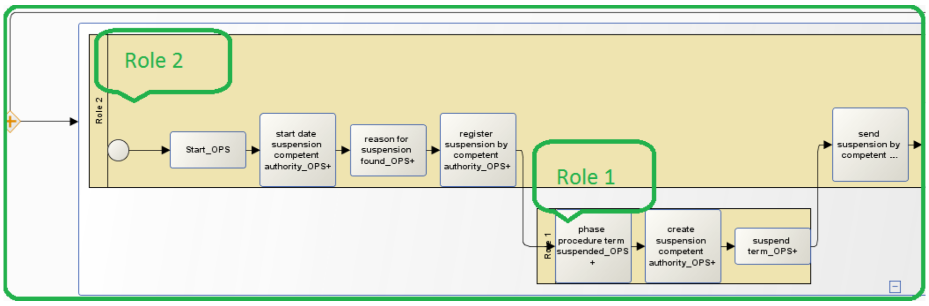

Role 2
Role 1
Start_OPS
start date suspension competent authority_OPS+
reason for suspension found_OPS+
register suspension by competent authority_OPS+
send suspension by competent ...
phase procedure term suspended_OPS +
create suspension competent authority_OPS+
suspend term_OPS+

Fig. 29: A fragment of the *OPS* subprocess

Figure 30 presents a fragment of EIND subprocess, constructed in case the event log was enriched with additional cancel attributes.[13] This subprocess describes the termination procedure. If the result of *terminate on request* task execution is *true* (it was decided to terminate the entire process), then the cancellation occurs and a token is produced to an outgoing sequence flow marked with a corresponding boundary cancellation event, leading to the final process task. Whereas it was decided not to terminate the entire process, subprocess EIND terminates normally and a token is produced to an ordinary outgoing sequence flow of the subprocess, resuming execution of the entire process.

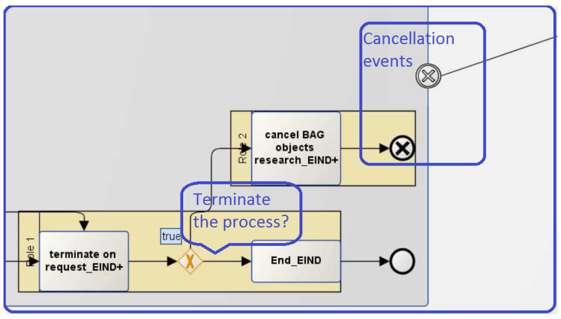


Fig. 30: A subprocess with a cancellation event

6.1.2 Discovering a Booking Process

A real-life event log of a ticketing system was analyzed as well. This log contains 774 traces and describes the behavior of a web-based system used for searching and booking flights. Each trace of the log represents the user interactions with the ticketing system. The user can buy a flight and get an insurance. For that purpose he or she needs to fill the form with personal data, choose an insurance type, choose a type of payment, and pay.

A hierarchical BPMN diagram of the online booking process discovered from the event log contains subprocesses describing filling personal data, registration of insurance, and payment procedures.

[13]Note that in the overall diagram (Fig. 27) the fragment of this subprocess is shown without cancellation event.

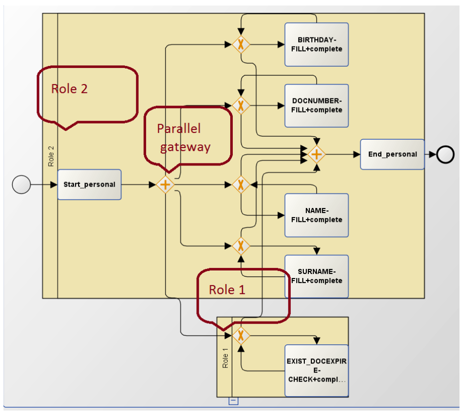


Fig. 31: A fragment of the filling personal data subprocess

A fragment of the filling personal data subprocess is presented in Fig. 31. The resource perspective in the diagram shows that different groups of users perform different steps within the subprocesses. As it follows from this diagram there are two groups of users: the first group fills the forms in order to buy a flight, while users from the other group just check/uncheck the document expiry date checkbox. The other important observation is that the users fill the form fields in an arbitrary order: the parallel gateway splits the control flow into several subflows, each of which corresponds to a concrete filed of the personal data form.

6.1.3 *Discovering a Banking Process*

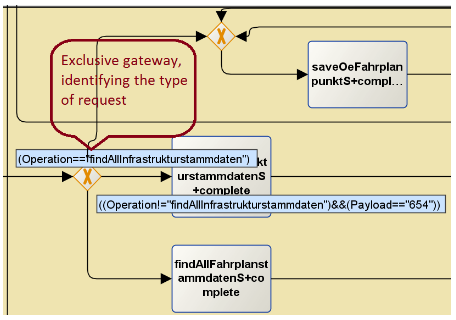

Exclusive gateway, identifying the type of request
saveOeFahrplanpunktS+compl...
(Operation=="findAllInfrastrukturstammdaten")
urstammdatenS+complete
((Operation!="findAllInfrastrukturstammdaten")&&(Payload=="654"))
findAllFahrplanstammdatenS+complete

Fig. 32: A fragment of the service subprocess

Another event log being analyzed is a small (66 traces) software log of a banking information system, which consists of three program layers (*front*, *service*, and *database*) and handles user requests. First, the user request is received by the front layer and transmitted to the next service layer. The service layer implements the business logics of the process, calling the database layer methods to store and retrieve data. Each program layer was represented as a subprocess. Fig. 32 shows a fragment of the service layer, it contains an exclusive gateway with guards, identifying types of operations (types of requests for a reference information) and passing the control to corresponding outgoing sequence flows.

6.2 Analyzing Discovered BPMN Models

In the previous subsection we have discussed the capability of the integrated discovery approach to build relevant multiperspective BPMN diagrams from real-life event logs. Now let us consider behavioral and structural characteristics of BPMN models discovered from the real-life event logs.

For the experiments we have chosen 7 real-life event logs: 5 event logs of building permit administrative processes of Dutch municipalities (denoted as *M1*-*M5*), an event log of a ticketing system (*TS*), and an event log of a banking system (*BS*). These event logs are described in the subsections 6.1.1, 6.1.2, and 6.1.3 respectively. For mining data and resource perspectives approaches [22] and [8] were used. To discover subprocesses all the event logs were preprocessed (the information needed for event localization was learned from the event attributes) and then a discovery approach [4] was applied. The Inductive miner technique [20] was used as an underlying control flow discovery method. The discovered Petri nets were converted to BPMN models using an algorithm presented in [16].

Since the aim of the work is to discover readable and convenient process models, the analysis of their structural characteristics is meaningful. Next to these structural characteristics we need to estimate behavior parameters of the models to evaluate their quality with respect to the initial event logs. In order to relate initial event logs and discovered process models, we consider three standard metrics: *fitness*, *precision* and *generalization* [1]. *Fitness* shows if the model can replay a given event log. If all the traces of the log can be replayed by a model then the value of the *fitness* function is 1. If there are non-fitting traces, then corresponding alignments, represented as sequences of synchronous and asynchronous steps performed to replay a trace on a model, are calculated and penalties are estimated in such a way that the value of the *fitness* function will be decreased proportionally to the number of asynchronous log and model moves in the alignment. The detailed description of fitness function can be found in [3]. In this work we calculate the fitness value for each subprocesses, using the approach presented in [26] and then take a weighted sum, where coefficients depend on the number of times subprocesses' traces appear in the initial event log.

Having a fitting process model is not sufficient, because it is easy to construct a process model that can replay any trace, but that has no relation to the log. Thus, an additional process metric: *precision* is needed. The *precision* shows if the model does not allow too much behavior. The *generalization* metric indicates whether the model is general enough. We calculated precision and generalization metrics for the discovered Petri net models, replaying the alignments. For the detailed descriptions of these metrics please refer to [5, 7]. Table 3 presents the behavioral characteristics of the models discovered from the event logs.

Log traces	Fitness	Precision	Generalization
M1	0.90	0.85	0.99
M2	0.72	0.91	0.98
M3	0.82	0.63	0.99
M4	0.66	0.75	0.99
M5	0.76	0.8	0.98
TS	0.77	0.78	0.98
BS	0.99	0.55	0.71

Table 3: Behavioral characteristics of process models discovered from the event logs

Structural metrics of process models highly correlate with their readability. The following structural metrics were considered during the analysis of the discovered models: number of nodes (including number tasks, XOR gateways, AND gateways, data objects, subprocesses, and swimlanes), number of control flows, density (ratio of the number of control flows to the possible maximum number of control flows), diameter (the maximal shortest path from the start node to a graph node), depth (maximal nesting of the graph), number of child nodes for compound nodes. As it was statistically shown in [25], the number of nodes, density, diameter, and depth have a negative correlation with the process model understandability. Thus, we were especially interested in these metrics. Structural metrics of the discovered process models are presented in Table 4.

Log traces	Number of tasks, XOR, AND gateways	Number of flows	Number of swimlanes and subprocesses	Number of child nodes	Density	Diameter	Depth	Number of data objects and guards
M1	180, 24, 60	350	13, 14	124 / 1 / 12.8	0.003	35 / 4 / 10.6	3	2, 5
M2	235, 49, 72, 2	482	11, 22	191 / 1 / 13.2	0.003	44 / 4/ 12.7	3	2, 3
M3	238, 44, 66	494	10, 22	31 / 1 / 7.7	0.003	27 / 4/ 10.5	2	2, 6
M4	265, 44, 72	504	14, 37	134 / 1 / 9.5	0.002	31 / 4 / 13.8	3	2, 5
M5	255, 44, 88	512	13, 42	177 / 1 / 9.5	0.002	35 / 4 / 12.4	3	2, 3
TS	59, 4, 20	134	6, 5	47 / 1 / 12.0	0.009	13 / 2 / 7.3	3	6, 4
BS	74, 4, 10	159	4, 4	31 / 2 / 16	0.1	11 / 2 / 7.0	3	3, 4

Table 4: Structural characteristics of process models discovered from the event logs. [14]

These structural characteristics of the discovered process models were compared with the characteristics of the BPMN models constructed manually. For that reason the existing Sigavio collection of 4781 BPMN models from various domains was analyzed. The results based on the Signavio collection analysis are presented in Table 5. For each parameter maximal, minimal and average values are specified.

Number of tasks, XOR, AND gateways	Number of flows	Number of swimlanes and subprocesses	Number of child nodes	Density	Diameter	Depth	Number of data objects and guards
34/ 0/ 7.2, 16/ 0/ 2, 14/ 0/ 0.66	27/ 0/ 3.4	38/ 0/ 14.6, 14/ 0/ 0.5	68/ 0/ 4.4	0.87/ 0/ 0.1	25/ 1/ 8	8/ 1/ 2.7	20/ 0/ 0.74 6 /0 /0.008

Table 5: Structural characteristics of BPMN models from the Signavio model collection.[14]

Although the total number of elements in the discovered process models is significantly higher that in manually created BPMN models, structural characteristics within containers (subprocesses and swimlanes), such as diameter (only for subprocesses) and number of child nodes, are comparable. Thus, an automatically discovered (sub)model within a subprocess or swimlane resembles manually created model by its structural characteristics. The maximum values for the number of child nodes of the discovered BPMN models were typically found in the main subprocesses whereas other subprocesses were comparable to the manually created models. Thus, the results of the experiments show that the proper identification of subprocesses helps to discover readable and convenient process models fully reflecting process behavior recorded in the event log. Moreover, the identification of subprocesses significantly reduce the time needed for discovery. Thus, it often takes less than a minute to discover a BPMN model from any of the real-life event logs, while construction of BPMN models without identification of subprocesses cannot be performed in a reasonable amount of time.

[14]The maximal, minimal and average values are separated by a slash, the values of metrics for different types of elements are separated by a comma.

7 Conclusion

In this paper we presented an approach to discover BPMN models from event logs leveraging the representational bias of BPMN. The models discovered cover multiple perspectives (next to control-flow also resources and data) and, if possible, have a meaningful hierarchy to improve their readability. The approach was implemented in ProM and resulted in a plug-in where multi-perspective hierarchical process models can be learned in a single step.

The overall approach can be summarized as follows. First the event log is split into smaller event logs. As demonstrated it is often possible to exploit domain knowledge, event and case attributes to create localized event logs. A labeled Petri net is learned per sublog and the projected log is replayed on these individual generally much smaller labeled Petri nets. Through replay we can add information on input and output data and we can even learn guards. The resulting data Petri net can be converted to a BPMN model and also resource information can be added. This results in a hierarchical BPMN model that integrates the different perspectives.

The work was tested on several real-life event logs. The results were evaluated using various metrics and compared with a collection of standard BPMN models. The results indicate that the models are of good quality and definitely better than models obtained using an approach that does not consider hierarchy. An added bonus is that the splitting of the logs also improves performance.

This paper consolidates various lines of research done earlier. However, the different results were never integrated and important steps were missing to go from an event log to a multi-perspective hierarchical BPMN model. Note that wherever possible we made the approach pluggable. For example, one can use any discovery technique to discover the control-flow of the subprocesses.

Future work aims at finding structural differences between discovered BPMN models (as-is) and reference BPMN models (to-be) using the model comparison tool proposed earlier [15].

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag, Berlin (2011)
2. van der Aalst, W.: A General Divide and Conquer Approach for Process Mining. In: Federated Conference on Computer Science and Information Systems (FedCSIS 2013), pp. 1–10. IEEE Computer Society (2013)
3. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Int. Rev. Data Min. and Knowl. Disc. **2**(2), 182–192 (2012). DOI 10.1002/widm.1045. URL http://dx.doi.org/10.1002/widm.1045
4. van der Aalst, W., Kalenkova, A., Rubin, V., Verbeek, E.: Process Discovery Using Localized Events. In: Application and Theory of Petri Nets and Concurrency, *Lecture Notes in Computer Science*, vol. 9115, pp. 287–308. Springer International Publishing (2015)
5. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B., van der Aalst, W.: Alignment Based Precision Checking. In: Workshop on Business Process Intelligence (BPI 2012). Tallinn, Estonia (2012)

6. Buijs, J., van Dongen, B., van der Aalst, W.: A Genetic Algorithm for Discovering Process Trees. In: IEEE Congress on Evolutionary Computation (CEC 2012), pp. 1–8. IEEE Computer Society (2012)
7. Buijs, J., van Dongen, B., van der Aalst, W.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In: OTM Federated Conferences, 20th International Conference on Cooperative Information Systems (CoopIS 2012), *Lecture Notes in Computer Science*, vol. 7565, pp. 305–322. Springer-Verlag, Berlin (2012)
8. 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, 16-19 April, 2013, pp. 103–110 (2013). DOI 10.1109/CIDM.2013.6597224
9. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Business Process Management: 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, chap. Beyond Tasks and Gateways: Discovering BPMN Models with Subprocesses, Boundary Events and Activity Markers, pp. 101–117. Springer (2014). DOI 10.1007/978-3-319-10172-9_7
10. De Weerdt, J., van den Broucke, S.K., Caron, F.: Bidimensional Process Discovery for Mining BPMN Models. In: Business Process Management Workshops, pp. 529–540. Springer (2014)
11. Dijkman, R., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information & Software Technology **50**(12), 1281–1294 (2008)
12. van Dongen;, B.: Bpi challenge 2015 (2015). DOI http://dx.doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
13. van Dongen, B., Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: A New Era in Process Mining Tool Support. In: Application and Theory of Petri Nets 2005, *Lecture Notes in Computer Science*, vol. 3536, pp. 444–454. Springer-Verlag, Berlin (2005)
14. García-Bañuelos, L., van Beest, N., Dumas, M., Rosa, M.L.: Complete and Interpretable Conformance Checking of Business Processes (2015). URL http://eprints.qut.edu.au/91552/
15. Ivanov, S., Kalenkova, A., van der Aalst, W.: BPMNDiffViz: A Tool for BPMN Models Comparison. In: Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015., pp. 35–39 (2015)
16. Kalenkova, A., van der Aalst, W., Lomazova, I., Rubin, V.: Process Mining Using BPMN: Relating Event Logs and Process Models. Software and Systems Modeling pp. 1–30 (2015)
17. Kalenkova, A., de Leoni, M., van der Aalst, W.: 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)
18. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach. In: BPM, *Lecture Notes in Computer Science*, vol. 9253, pp. 55–71. Springer (2015)
19. Leemans, S., Fahland, D., van der Aalst, W.: Discovering Block-structured Process Models from Event Logs: A Constructive Approach. In: J. Colom, J. Desel (eds.) Applications and Theory of Petri Nets 2013, *Lecture Notes in Computer Science*, vol. 7927, pp. 311–329. Springer-Verlag, Berlin (2013)
20. Leemans, S., Fahland, D., van der Aalst, W.: Discovering Block-Structured Process Models from Incomplete Event Logs. In: Application and Theory of Petri Nets and

Concurrency, *Lecture Notes in Computer Science*, vol. 8489, pp. 91–110. Springer International Publishing (2014)

21. Lengauer, T., Tarjan, R.: A Fast Algorithm for Finding Dominators in a Flowgraph. ACM Trans. Program. Lang. Syst. **1**(1), 121–141 (1979). DOI 10.1145/357062.357071
22. de Leoni, M., van der Aalst, W.: Data-aware process mining: Discovering decisions in processes using alignments. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013, pp. 1454–1461 (2013). DOI 10.1145/2480362.2480633
23. Muehlen, M., 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), *Lecture Notes in Computer Science*, vol. 5074, pp. 465–479. Springer-Verlag, Berlin (2008)
24. OMG: Business Process Model and Notation (BPMN). Object Management Group, formal/2013-12-09 (2013)
25. 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)
26. Verbeek, H., van der Aalst, W.: Merging Alignments for Decomposed Replay. In: Application and Theory of Petri Nets and Concurrency: 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings, pp. 219–239. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-39086-4_14
27. Ye, J., Sun, S., Song, W., Wen, L.: Formal Semantics of BPMN Process Models Using YAWL. In: Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on, vol. 2, pp. 70–74 (2008). DOI 10.1109/IITA.2008.68