Discovering High-level BPMN Process Models from Event Data

A.A. Kalenkova[1], A. Burattin[2], M. de Leoni[3], W.M.P. van der Aalst[3,4], and A. Sperduti[5]

[1]*National Research University Higher School of Economics, Russia*
[2]*Technical University of Denmark, Denmark*
[3]*Eindhoven University of Technology, The Netherlands*
[4]*RWTH Aachen University, Germany*
[5]*The University of Padua, Italy*

akalenkova@hse.ru, andbur@dtu.dk, m.d.leoni@tue.nl,
wvdaalst@pads.rwth-aachen.de, sperduti@math.unipd.it

1 Introduction

Process-Aware Information Systems (PAISs) are increasingly used by organizations to support their businesses. All these systems record the execution of process instances in so-called event logs. These logs thus capture information about activities performed. Each event records the execution of an activity instance by a given resource at a certain point in time along with the output produced. Analyzing event logs, understanding and improving processes based on facts are the primary objectives of *process mining* (van der Aalst 2016).

In a relatively short time span, this discipline has proven to be capable of extracting from event logs in-depth insights into process-related problems that contemporary enterprises face. The lion's share of attention in process mining is *process discovery*. Process discovery aims to discover the actual processes that are executed within organizations. Typically, the output is a model that is a description of the process. Process models produced through process discovery are based on facts and are objective, in contrast with those hand-made, which are subjective and biased toward the designer's belief.

What does one need a model for? Models can be used for multiple purposes. They can not only be used to configure a PAIS but also to discuss responsibilities, analyze compliance, reason over bottlenecks, resource utilization, costs, risks, and other performance-related aspects of processes.

To ensure an efficient communication with the company's stakeholders, the process model notation plays an important role. Several notations exist but the evidence is showing that, during the last years, BPMN (Business Process Model and Notation) 2.0

(Object Management Group 2013) is becoming the defacto standard for modeling business processes in industry. Therefore, it seems crucial that process discovery techniques produce models in this notation. Furthermore, models need to be of a high quality: they need to be accurate, i.e., represent various perspectives of event data, and not to underfit the reality. To achieve this, the models need to integrate the control-flow perspective, i.e., the allowed sequences of activities, with the resource and data perspective (the latter a.k.a. case perspective). The organization perspective focuses on which actors (people, systems, roles, organizational units) are involved and how: who can execute what part of the process. The data perspectives focus on the properties of the process instance executions (e.g., the age or gender of loan applicants) and how these properties affect the process executions, e.g., the applications of gold customers are managed differently from those of silver customers, which are, in turn, different from normal customers.

The BPMN notation allows one to integrate these perspectives into a single model. If process mining focuses on discovering integrated models, one can discover that, e.g., requests from certain customer types are managed by given organizational units or do not need to be assessed by managers.

As mentioned, models need to be readable even when the processes consist of dozens of activities and involve several organizational units. To tackle this, it is worth using a "divide-et-impera" solution: the model is split into several sub-models, each of which is a different, e.g., BPMN model. These sub-models are then connected to illustrate how the executions iterate over them. In other words, readability is achieved by making the model hierarchically structured into a main model and sub-models.

This paper illustrates a methodology to exploit event logs to generate hierarchical BPMN models that integrate the different perspectives mentioned above. The paper starts from analyzing and reporting the BPMN meta-model (Object Management Group 2013), from which we derived the main modeling elements, which can be discovered using different process mining techniques. We identified three main types of BPMN models, which inherit core BPMN modeling constructs: *BPMN models with data*, *hierarchical BPMN models*, and *BPMN models with resources*. After that, we report on process discovery techniques that mine these three types of models. Then, we illustrate how these techniques can work in concert to integrate different views into a single hierarchical BPMN model, which combines data, resource, and control-flow perspectives.

This integrated discovery approach was implemented as a plugin for ProM, the most widespread open-source process mining framework. We conducted three thorough real-life case studies with processes enacted in several Dutch municipalities, in an online sales shop, and in a banking system. The event logs contained information related to the control-flow, the resource and the data perspective, which enabled us to discover rich, integrated BPMN models. In addition to mining different perspectives, these case studies illustrate the importance of mining hierarchical models where the model is broken down into several subprocesses. Indeed, the event logs recorded the execution of dozens of process activities, which would have led to gigantic and unreadable process models if we focused on mining "flat" process models.

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. This allowed us to show that the models that we automatically discovered 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. Section 4 demonstrates the applicability of the existing process discovery techniques to mine different types of BPMN models. The integrated discovery approach is presented in Section 4. Section 5 demonstrates the experimental results. Finally, Section 6 concludes the paper.

2 Related Work

There is a large body of work on process discovery. In (van der Aalst 2016), van der Aalst discusses a large repertoire of process discovery techniques that mainly focus on discovering the control-flow, i.e., the sequences of executions of process activities. In addition, there are several research works to discover the other process perspectives or some subprocess hierarchies, e.g., (de Leoni & van der Aalst 2013, Conforti et al. 2014, 2016, Bazhenova et al. 2016, De Smedt et al. 2017, Kalenkova et al. 2017). However, these works are often limited to a single perspective, such as the data or the resource perspective. Although some approaches to the discovery and conformance checking of multi-perspective process models were proposed earlier (Rozinat 2010, Rozinat & van der Aalst 2006, Mannhardt et al. 2016, Mannhardt 2018), this work leverages the expressive possibilities of the BPMN language. In (De Weerdt et al. 2014) an approach for discovering BPMN models represented by control and resource perspectives is suggested, but it is limited to these perspectives only. Furthermore, it is based on a specific process discovery algorithm, while we present a flexible discovering methodology where diverse discovery techniques can be integrated in a flexible manner. The aim is to discover a single process model with hierarchies integrating the data-flow, resource, and control-flow perspectives.

We also believe that the evaluation of real-life case studies is certainly of higher quality than the evaluations in previous works. No existing works have ever conducted a significant effort to illustrate how all perspectives can be mined and put together into an integrated model, certainly not for case studies of a complexity comparable to what is proposed in this paper. On average, the complexity of the event logs and of the underlying processes is certainly far higher than any previous studies.

A related topic is the recent introduction of the DMN standard (*Decision Model and Notation (DMN) V1.1* 2016) by the OMG group. BPMN can be complemented with DMN, where data-related decisions can be represented. In DMN, the data perspective is mainly modeled outside the BPMN process model as separate tables. In (Batoulis et al. 2015), the authors argue that this separation of concerns would increase the readability, especially when models are of significant size. Indeed, the separation of concerns might theoretically increase the readability, even though more investigation is certainly necessary to support this statement. However, it is trivial to extract the data-related decisions from the integrated BPMN models and to represent them as separate

DMN tables. Indeed, in (Bazhenova et al. 2016), the authors mine DMN tables using an approach that largely coincides with what is proposed in this paper. Compared with the latter, we propose an integrated framework where discovering the data perspective (in the form of, e.g., rules attached to BPMN arcs or DMN tables) is just one of the ingredients.

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 to 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 (van der Aalst et al. 2015). Let A be a set of activity names, $\tau \in A$ be a *silent activity name*, $\hat{A} = A \setminus \{\tau\}$ – a set of *activity names* without the *silent activity*, *Attr* – a finite set of *attributes*, and *Val* – a set of *values*.

An *event* is defined as a pair $e = (a, f)$, where $a \in \hat{A}$ is a name of the event, and $f : \mathit{Attr} \to \mathit{Val}$ is an attribute function, which maps attributes to their values. By E we denote a set of events. E^* denotes the set of all finite sequences (including the empty sequence) of elements from E. A sequence of events $\mathit{tr} \in E^*$ is called a *trace*, and an *event log* L is a multiset of traces, i.e., $L \in \mathcal{B}(E^*)[1]$. A *concatenation* of two traces $\langle e_1, e_2, ..., e_k \rangle$ and $\langle e'_1, e'_2, ..., e'_l \rangle$ is denoted by $\langle e_1, e_2, ..., e_k \rangle \cdot \langle e'_1, e'_2, ..., e'_l \rangle$ and equals $\langle e_1, e_2, ..., e_k, e'_1, e'_2, ..., e'_l \rangle$.

Let $E' \subseteq E$ be a subset of events, then a *projection* $\mathit{tr}_{\uparrow E'}$ of a trace $\mathit{tr} \in E^*$ on the set E' is defined inductively: $\mathit{tr}_{\uparrow E'} = \begin{cases} \langle\rangle, & \text{if } \mathit{tr} = \langle\rangle, \\ \mathit{tr}'_{\uparrow E'}, & \text{if } \mathit{tr} = \langle e \rangle \cdot \mathit{tr}', e \notin E', \\ \langle e \rangle \cdot \mathit{tr}'_{\uparrow E'}, & \text{if } \mathit{tr} = \langle e \rangle \cdot \mathit{tr}', e \in E'. \end{cases}$

A projection $L_{\uparrow E'}$ of an event log $L \in \mathcal{B}(E^*)$ on a set $E' \subseteq E$ is obtained by projecting on E' all the traces from L.

Consider the fragment of an event log presented in Tab. 1.

This event log contains information about the procedure of application processing. 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 employee.

Each case (identified by *Case ID*) corresponds to the processing of a concrete application and represents a sequence of events (a trace). Timestamps define the order of events within the traces. The *Resource* attribute indicates employees performing the activities.

	Case ID	Activity Name	Resource	Timestamp
e_1	1	"receive application"	"John"	2017-08-15 11:23:17
e_2	1	"send acknowledgment of receipt"	"John"	2017-08-15 11:45:54
e_3	2	"receive application"	"Mary"	2017-08-15 14:12:54
e_1	3	"receive application"	"John"	2017-08-15 14:21:32
e_4	1	"process application"	"Kate"	2017-08-15 14:36:17
e_2	3	"send acknowledgment of receipt"	"John"	2017-08-15 14:41:15
e_4	3	"process application"	"Kate"	2017-08-15 14:45:59
e_5	2	"send acknowledgment of receipt"	"Mary"	2017-08-15 14:48:12
e_6	2	"forward to competent authority"	"Jane"	2017-08-15 15:22:07

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

Formally, this event log can be defined as a multiset of traces $L = [\langle e_1, e_2, e_4\rangle^2, \langle e_3, e_5, e_6\rangle^1]$, where each event is identified by its name and the value of the *Resource* attribute: $e_1 = (\text{"}receive\ application\text{"}, f_1)$, $f_1(Resource) = \text{"}John\text{"}$, $e_2 = (\text{"}send\ acknowledgment\ of\ receipt\text{"}, f_2)$, $f_2(Resource) = \text{"}John\text{"}$, $e_3 = (\text{"}receive\ application\text{"}, f_3)$, $f_3(Resource) = \text{"}Mary\text{"}$, $e_4 = (\text{"}process\ application\text{"}, f_4)$, $f_4(Resource) = \text{"}Kate\text{"}$, $e_5 = (\text{"}send\ acknowledgment\ of\ receipt\text{"}, f_5)$, $f_5(Resource) = \text{"}Mary\text{"}$, $e_6 = (\text{"}forward\ to\ competent\ authority\text{"}, f_6)$, $f_6(Resource) = \text{"}Jane\text{"}$. As it was mentioned before, *Case ID* is used to define traces, *Timestamp* forms the order of events within a trace.

3.2 Classical Petri Nets and Petri Nets with Data

Petri nets are the most popular low-level process modeling formalism used in the context of process mining. A *labeled Petri net* is a tuple $PN = (P, T, F, M_{init}, M_{final}, l)$, where P is a set of places, T is a set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, M_{init}, M_{final} are initial and final markings respectively, and $l \in T \to A$ is a labeling function which maps transitions to a set of activity names. A *marking* is a function $M \in P \to \mathbb{N}$ mapping places to natural numbers. In a marking M place p contains $M(p)$ tokens. 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 places 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 place from ${}^\bullet t$ and one token is added to each place from $t^\bullet$.

If $l(t) = \tau$, then t is called *invisible* and represented as a black box. Function l can be applied to transition sequences using the following inductive definition:

$$l(\sigma) = \begin{cases} \langle\rangle, & \text{if } \sigma = \langle\rangle, \\ l(\sigma'), & \text{if } \sigma = \langle t\rangle \cdot \sigma', l(t) = \tau, \\ \langle l(t)\rangle \cdot l(\sigma'), & \text{if } \sigma = \langle t\rangle \cdot \sigma', l(t) \neq \tau. \end{cases}$$

A trace $\langle (a_1, f_1), ..., (a_k, f_k)\rangle \in E^*$ 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 $l(\sigma) = \langle a_1, ..., a_k \rangle$[2].

Now let us extend Petri nets and define Petri nets with data introduced in (de Leoni & van der Aalst 2013). 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. The function U defines possible values for each variable v, such that $U(v) = D_v$, where D_v is the domain for v. The functions $R : T \to 2^V$ and $W : T \to 2^V$ define sets of variables, which are read and written by the transitions. The guard function $G : T \nrightarrow \mathcal{G}_{\mathcal{V}}$[3] associates some of the transitions with guards. A transition can fire only if a corresponding guard evaluates to true and all the input places contain at least one token[4]. A state of a Petri net with data is represented by a pair (M, Val), where M is a marking, and *Val* is a function, which maps variables to their values, i.e., $Val : V \to D \cup \{\perp\}$, where $D = \cup_{v \in V} U(v)$. The 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. This Petri net describes a booking pro-

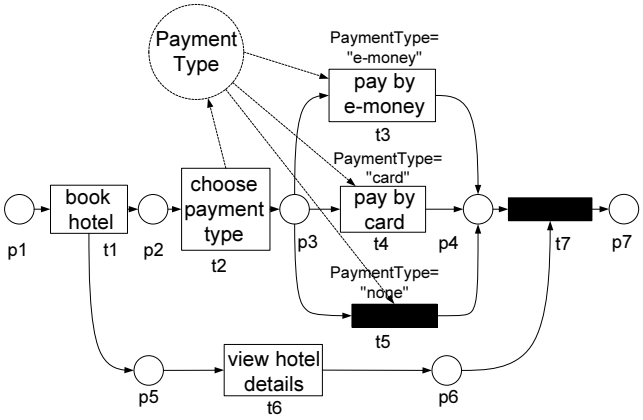

Figure 1: An example of a Petri net with data.

cess, where the user first books a hotel, chooses a 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 payment). 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*", $l(t_5) = l(t_7) = \tau$. Place p_1 contains one token in the initial marking, formally $M_{init} = [p_1]$, the final is defined 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 to the variable $PaymentType$, transitions t_3, t_4, and t_5 read its values, formally, $W(t_2) = \{PaymentType\}$, $R(t_3) = R(t_4) = R(t_5) = \{PaymentType\}$. Moreover, the 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*", $PaymentType =$ "*none*" respectively, for other transitions the guard function is not defined. This Petri net can replay 8 traces, represented by distinct sequences of activity names. Sequence $\langle (a_1, f_1), (a_2, f_2), (a_3, f_3), (a_4, f_4) \rangle$, where $a_1 =$ "*book hotel*", $a_2 =$ "*choose payment type*", $a_3 =$ "*view hotel details*", $a_4 =$ "*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 the BPMN 2.0 specification (Object Management Group 2013). BPMN offers a wide range of modeling elements, but not all of them are frequently employed (Muehlen & Recker 2008). In contrast to the existing formal BPMN semantics (Dijkman et al. 2008, Kheldoun et al. 2015, Ye et al. 2008) 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 future work.

We iteratively introduce and formalize various types of BPMN diagrams. To show their relations with the BPMN standard, corresponding meta-models were extracted from the specification (Object Management Group 2013). 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 (Object Management Group 2013) is specified.

3.3.1 Core BPMN Models

Core BPMN models are used to formalize flat processes represented by a 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 *activities*, *gateways*, and *events*. *Activities* stand for process steps (in separate core BPMN models they correspond to *tasks*), *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.

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 flow of a *start event* (Fig. 3a) contains a token, while other sequence flows do not. Each node (except the start event) can be

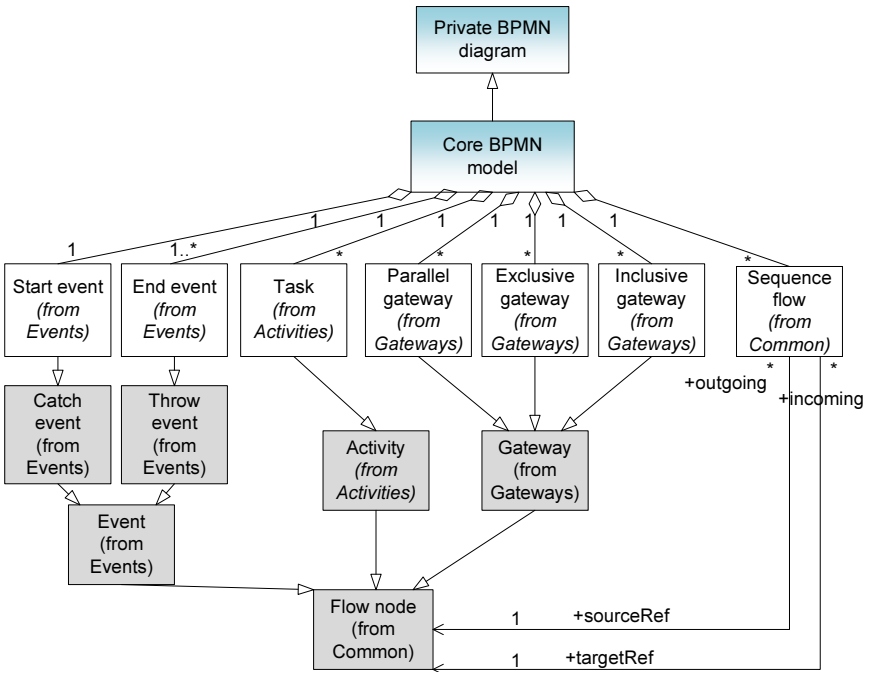

Figure 2: A meta-model for core BPMN models.

enabled and may fire[5]. *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 to one of the outgoing sequence flows.

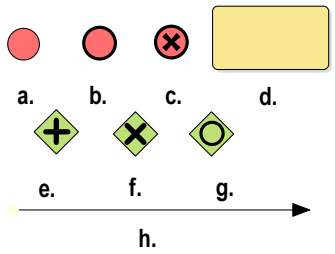

Figure 3: Elements of core BPMN model: a. start event, b. end event, c. cancellation end event, d. activity, e. parallel gateway, f. exclusive gateway, g. inclusive gateway, h. sequence flow.

A *parallel gateway* (Fig. 3e) is enabled only if each of the incoming sequence flows contains at least 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 flows will contain tokens, without firing this gateway. An inclusive gateway produces tokens for some of the outgoing sequence flows.

An *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 can be error, signal, cancel, and other types of events. In this paper we combine all of them together conceptually as one type called *cancel* event.

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*.

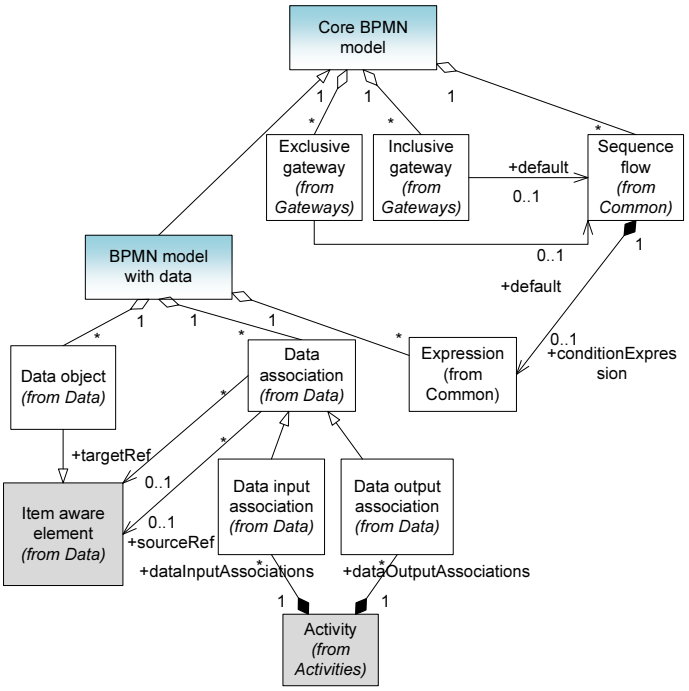

Figure 4: A meta-model for BPMN models with data.

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 conditional expressions 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 consider them in BPMN models with data only, since without conditional expressions default sequence flows do not influence the model execution.

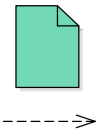

Figure 5: Data object, data association.

In contrast to the BPMN specification (Fig. 4), where for any arbitrary sequence flow a corresponding *conditional expression* can be determined, we will assume that conditional expressions are set only for outgoing sequence flows of exclusive and inclusive gateways. Also we will assume that for each exclusive or inclusive gateway there is exactly one outgoing *default sequence flow*.

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

3.3.3 BPMN Models with Resources

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

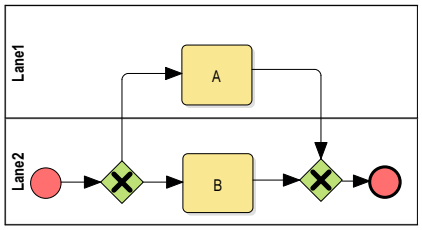

Figure 6: An example of a BPMN model with lanes.

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. In this work we will consider only one level of granularity. Lanes may contain flow nodes, such as activities, gateways, events. Note that sequence flows may cross a lane's boundaries.

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 forms a container with inner flow nodes, such as start/end events, other activities, and gateways. It means that an activity can correspond to another core BPMN model, which in turn can also contain compound activities.

The behavior of hierarchical BPMN models is built on top of the behavior of core

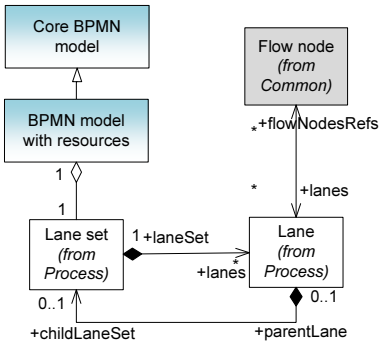

Figure 7: A meta-model for BPMN models with resources.

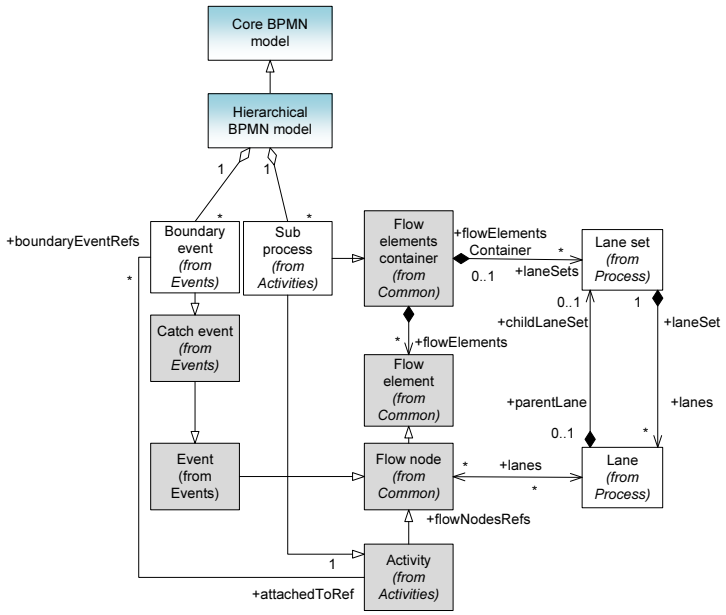

Figure 8: A meta-model for hierarchical BPMN models.

BPMN models and extends their semantics to execute *non-atomic* activities, i.e., subprocesses.

Each subprocess can be activated iff one of the incoming sequence flows contains a token and there are no tokens inside the subprocess and its nested 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 are consumed by non-cancellation end events), then a token is passed to each regular outgoing sequence flow. In cases of a cancellation, the

subprocess is terminated and the 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. In other words, we will not consider boundary events attached to atomic activities without an inner structure.

An example of a subprocess is presented in Fig. 9. An end cancellation event and a corresponding boundary event are marked with a *cross sign*.

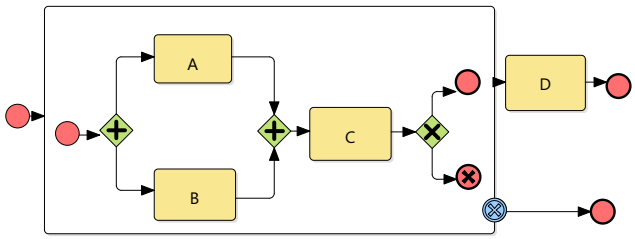

Figure 9: An example of a BPMN model with a subprocess.

3.3.5 Integrated BPMN Models

Integrated BPMN models incorporate all process modeling perspectives introduced above (see Fig. 10 below).

We will assume that an integrated BPMN model consists of core BPMN models, and each core BPMN model representing a subprocess or a root process, can be extended by both resources and data. A lane set can 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. That also holds for data, since (according to the BPMN specification) each subprocess may have its own variables.

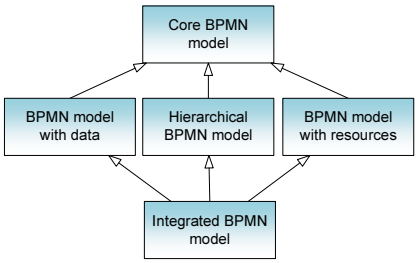

Figure 10: A meta-model for integrated BPMN models.

4 A Framework for Discovering Integrated BPMN Models

This section presents a framework for discovering multi-perspective hierarchical BPMN models. Firstly, techniques for mining various modeling perspectives are discussed, and then a novel integrated discovery approach is introduced.

4.1 Transforming Flat Process Models to BPMN

Flat process models, such as Petri nets, causal nets (van der Aalst et al. 2011) and process trees (Leemans et al. 2014) can be obtained from event logs using existing process discovery techniques.

To take advantages of the vast number of existing process discovery techniques producing Petri nets (Leemans et al. 2014, van der Aalst et al. 2003, Bergenthum et al. 2007, van der Aalst et al. 2008), an approach for the transformation of labeled Petri nets to BPMN, is used as a basis in this work. This transformation approach is introduced by examples in this subsection. The detailed description of the control-flow transformations can be found in (Kalenkova et al. 2017). These transformations were implemented as plugins (Kalenkova et al. 2014) for ProM (van Dongen et al. 2005) – an open-source framework for developing process mining algorithms.

As an example, consider a labeled Petri net, which models a simple booking process (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 transitions highlighted in black are invisible.

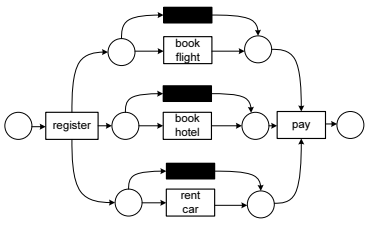

Figure 11: A labeled Petri net of a booking process.

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

This labeled Petri net can be automatically transformed to a BPMN model (Fig. 12), using an existing transformation algorithm (Kalenkova et al. 2017). This algorithm converts a labeled Petri net to a core BPMN model in such a way that for each visible transition there exists one, and only one, activity with the same label. It was proven (Kalenkova et al. 2017) 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 to an end event.

The resulting core BPMN model (Fig. 12) complies with the meta-model presented in Fig. 2. It contains activities, a start and an end event, exclusive/parallel gateways, and sequence flows. Note that sequence flows can connect arbitrary flow nodes, thus, invisible transitions are not added to the process model (Fig. 12).

This BPMN model can be simplified and transformed to a model with inclusive gateways (Fig. 13), which compactly represents routing constructions.

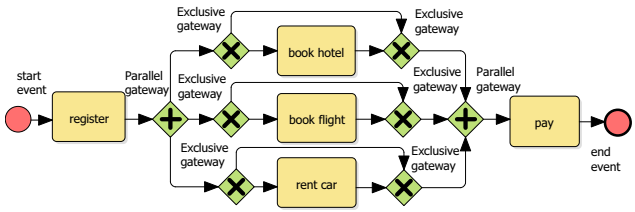

Figure 12: A BPMN model constructed from the labeled Petri net presented in Fig. 11.

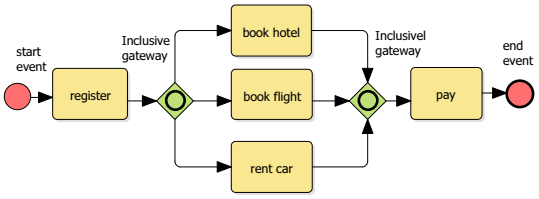

Figure 13: A BPMN model with inclusive gateways obtained form the BPMN model presented in Fig. 12.

4.2 Converting Petri Nets with Data to BPMN Models with Data

In order to transform a Petri net with data to a target BPMN model, first this Petri net should be converted to a core BPMN model, using an algorithm described in (Kalenkova et al. 2017).

After that, variables, read and write functions, and transition guards are trivially transformed to data objects, input and output data associations, and conditional expressions respectively.

To illustrate the conversion of transition guards let us consider a fragment of a Petri net presented in Fig. 14 a. and a corresponding fragment of a data BPMN model shown in Fig. 14 b.

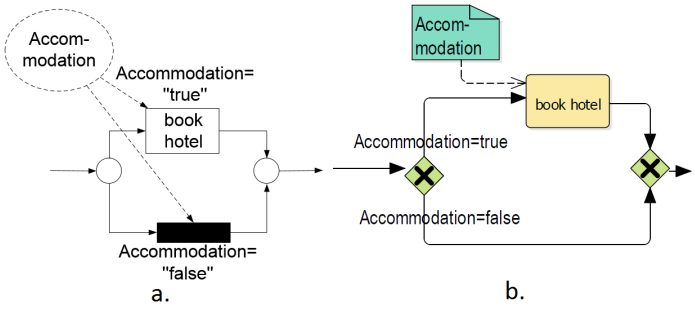

Figure 14: Conversion of data objects, data associations and guards.

This example shows that each transition guard is transformed to a conditional expression of a corresponding sequence flow.

4.3 Enhancing Core BPMN Models by Adding Resources and Subprocesses

In this subsection an approach for the enhancement of core BPMN models by adding resources and subprocesses will be introduced.

The resource-based enhancement algorithm takes a core BPMN model and an event log as input parameters and enhances the BPMN model with resources by specifying a function, which maps activities and other elements to lanes, producing a BPMN model with a set of resources. This approach puts each activity of a core BPMN model to an aggregate resource (or lane), taking into account the resource attributes of corresponding events. Note that different corresponding events in a log can be associated with different resources. We assume that activities connected by sequence flows (possibly through a series of gateways) which have a certain number of common resources belong to the same lane. All nodes of other types (gateways, events) are attached to one of the resources of the "neighboring" activities. For a detailed description of the approach please refer to (Burattin et al. 2013).

Similarly to resources, subprocesses are defined using additional information presented in event logs. In this work we construct hierarchical BPMN models with subprocesses using the *localized logs* process discovery technique proposed earlier in (van der Aalst et al. 2015).

Each event in the event log can be assigned to a so-called *region*. As it will be shown later in Section 5, such event logs can be found in many application domains. Moreover, if there is no information on the events' localization, then the event log can be enriched with additional data using expert knowledge.

Let us give a definition of a localized event log (van der Aalst et al. 2015). A *localized event log* is a triplet $L_L = (L, Reg, loc)$, where $L \in \mathcal{B}(E^*)$ is an event log, *Reg* is a set of regions (or localizations), and $loc : E \to \mathcal{P}_{NE}(Reg)$[7]. We will consider only stable localized event logs. A localized event log $L_L = (L, Reg, loc)$ with $L \in \mathcal{B}(E^*)$ is called *stable* iff $\forall e_1, e_2 \in E$, such that $e_1 = (a_1, f_1)$, $e_2 = (a_2, f_2)$, and $a_1 = a_2$, holds $loc(e_1) = loc(e_2)$.

The localized logs discovery approach performs a construction of a target labeled Petri net from a localized event log $L_L = (L, Reg, loc)$, $Reg = \{reg_1, ..., reg_k\}$, $L \in \mathcal{B}(E^*)$ in three steps:

1. For each region $reg_i \in Reg$ a labeled Petri net $PN_i = (P_i, T_i, F_i, M_{init_i}, M_{final_i}, l_i)$ is discovered from a projection of the log $L_{\uparrow E_i}$ on a set of events $E_i = \{e \in E | reg_i \in loc(e)\}$, 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;

2. A resulting labeled Petri net $PN_U = (P, T, F, M_{init}, M_{final}, l)$ is defined as a union of all discovered labeled Petri nets $PN_1, ..., PN_k$, where transitions with the same labels are merged, these transitions define start and end points of the nested subprocesses;

3. All structurally redundant hanging places are removed.

After the above, using the approach (Kalenkova et al. 2017), the resulting Petri net is converted to a BPMN model, where all subgraphs, corresponding to concrete regions, form subprocesses. Compared with the approaches discussed in (Conforti et al. 2014, 2016), we provide language inclusion properties of the subprocesses that are discovered. In the next section we will demonstrate how to integrate all the introduced discovery techniques to learn hierarchical multi-perspective BPMN models.

4.4 Integrated Discovery Approach

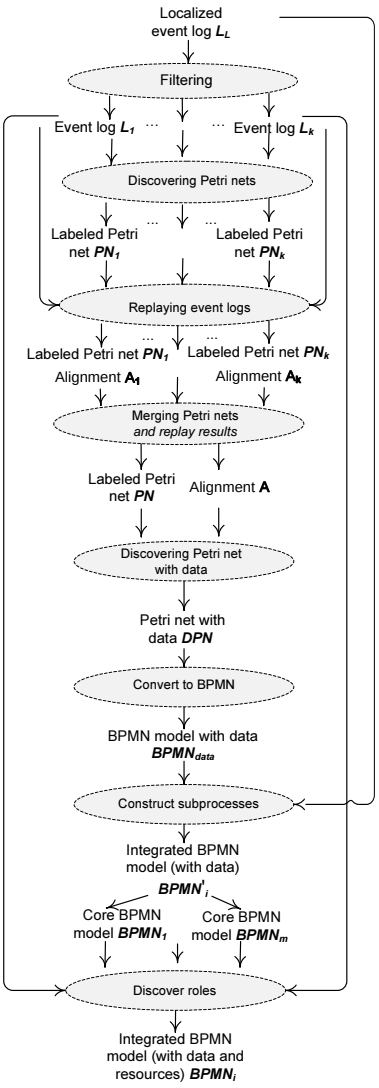

Figure 15: The integrated discovery approach presented in this paper.

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. 15. 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 any of the existing discovery techniques (Leemans et al. 2014, van der Aalst et al. 2003, Bergenthum et al. 2007, van der Aalst et al. 2008).

Then each event log L_i is replayed on the corresponding labeled Petri net PN_i and an alignment A_i (replay sequences of steps, including synchronous log and model moves, log only and model only moves) is constructed. These Petri nets and alignments are merged to a unified labeled Petri net PN and a corresponding *pseudo-alignment* A', which in the general case may contain replay sequences with merge conflicts (Verbeek & van der Aalst 2016). All the replay sequences containing merge conflicts are filtered out from A' or (if it is possible) resolved, and an alignment A without conflicts is obtained.

Next a method for enriching

Petri nets with data recorded in the event logs using corresponding alignments (de Leoni & van der Aalst 2013) is applied, and as a result a Petri net with data, e.g., *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, thoroughly presented in (Kalenkova et al. 2017, 2014).

After that, on the basis of the localization information contained in the initial event log, subprocesses are constructed within $BPMN_{data}$, and an integrated model $BPMN'_i$, containing core models $BPMN_1$, ..., $BPMN_m$ enriched with data, is produced.

Then the core models $BPMN_1$, ..., $BPMN_m$ are enriched with resources on the basis of information presented in the corresponding event logs. And finally, the target integrated BPMN model $BPMN_i$ is constructed.

Note that log partitioning allows to reduce the total computational time. Log partitioning works especially well for algorithms of enriching Petri nets with data, since they involve replay techniques, which are known to be time consuming for large models and logs.

5 Case Studies

In this section we evaluate the integrated discovery approach developed as a plugin within the MultiPerspectiveMiner package for ProM (Process mining) framework (van Dongen et al. 2005) – an open source extensible platform, widely used for the analysis of event logs. 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.

5.1 Discovering Multiperspective BPMN Models

Subsections 5.1.1, 5.1.2, and 5.1.3 contain information on the event logs of municipal, booking, and banking systems respectively as well as multiperspective BPMN models discovered from these event logs.

5.1.1 Discovering Municipal Processes

First, we took event logs from building permission administrative processes of five Dutch municipalities (van Dongen 2015) (containing 1,199, 832, 1,409, 1,053, and 1,156 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. This fragment of the log describes

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.

one case of the building permission 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 the *DRZ* subprocess resource *560912* decides not to forward the application to the competent authority (note that the value of the *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 the *BPT* subprocess. Then, after several steps of the application processing, another resource *560890* publishes a document with a final decision.

The resulting BPMN model describes a building permission process and contains 13 subprocesses with control-flow obtained on the basis of the inductive miner (Leemans et al. 2014) using the *Subprocess name* attribute for the subprocesses' identification.

The data and resource perspectives were discovered by (de Leoni & van der Aalst 2013) and (Burattin et al. 2013) 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 technique (de Leoni & van der Aalst 2013) which relies on the model and log alignment (and thus known to be possibly time consuming as logs and models get bigger).

Now let us consider the discovered BPMN model in details. A fragment of the *BPT* subprocess is presented in Fig. 16.

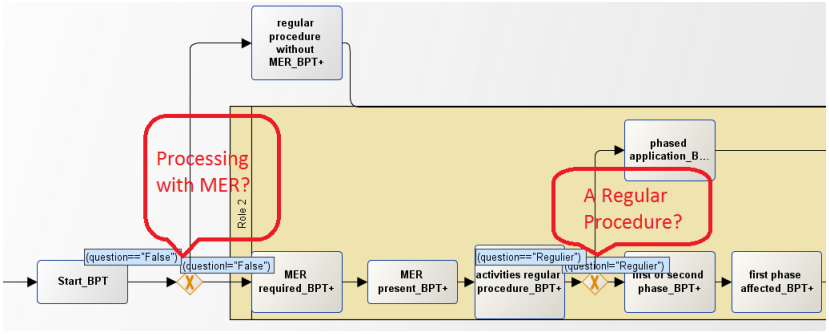

Figure 16: A fragment of the *BPT* subprocess.

As it follows from the diagram, the decision whether or not to process an 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 the diagram. According to the log (Tab. 2) the choice was made (the variable assigned a value) in the previous step, when a performer decided whether the application should be forwarded to a competent authority. The other exclusive choice gateway (Fig. 16) also has a guard depending on the value of the *question* data variable. Cases are split according to the procedure type (*Regular* or not). 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*).

Another fragment of the diagram, containing the *OPS* subprocess, is presented in Fig. 17. It illustrates the applicability of the resource discovery approach.

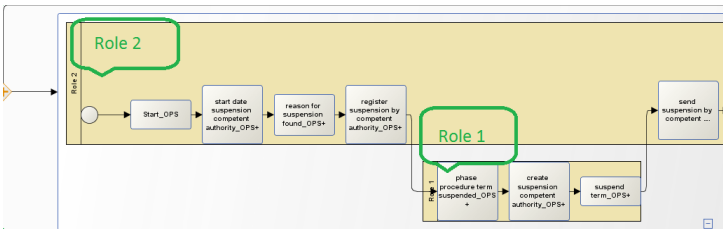

Figure 17: A fragment of the *OPS* subprocess.

The subprocess OPS describes a procedure of suspending an application and is performed by two roles. As it follows from the diagram, *Role 2* is responsible for the technical steps, such as registration of suspension, finding a reason for suspension and forwarding the application to a competent authority. While *Role 1* is a role of a competent authority, who defines the terms.

The basic approach constructs subprocesses with one entry and one exit. However, in some cases it may be feasible to construct subprocesses with multiple exits. One of these exits may represent a normal subprocess termination, while the other exists stand for exceptional terminations. The localized logs approach can be extended to construct subprocesses with multiple exits. This requires to enrich the event log with additional

information about terminating events.

Figure 18 presents a fragment of the EIND subprocess, constructed for the event log enriched with additional cancellation attributes. This subprocess describes a termination procedure. If the result of *terminate on request* activity is *true* (i.e., it was decided to terminate the entire process), then a cancellation occurs and a token is produced to an outgoing sequence flow marked with a corresponding boundary cancellation event, leading to the final process activity. Whereas, it was decided not to terminate the entire process, the subprocess EIND terminates normally, and a token is produced to an ordinary outgoing sequence flow, resuming execution of the entire process.

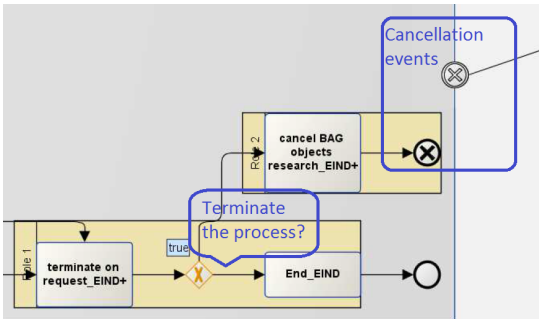

Figure 18: A subprocess with a cancellation event.

5.1.2 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 the flights. Each trace of the log represents 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-in the form with personal data, choose an insurance type, choose a type of payment, and pay.

A hierarchical BPMN diagram discovered from the event log contains *personal data entry*, *registration of insurance*, and *payment* subprocesses. A fragment of the *personal data entry* subprocess is presented in Fig. 19.

The resource perspective 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-in/check-out the document expiry date checkbox. Another important observation is that 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 field of the personal data form.

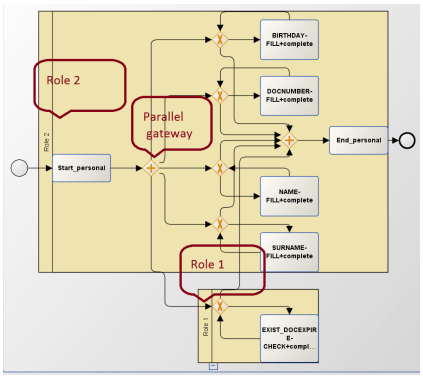

Figure 19: A fragment of the *personal data entry* subprocess.

5.1.3 Discovering a Banking Process

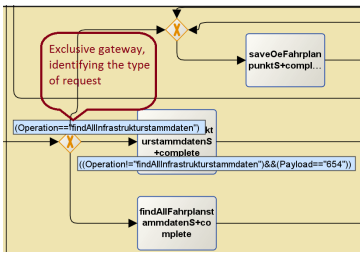

Figure 20: A fragment of the service subprocess.

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*, *database*) and handles the user requests. First, a user request is received by the front layer and transmitted to the next service layer.

The service layer implements a business logic of the process, interacting with the database layer to store and retrieve data. Each program layer was discovered as a subprocess. Fig. 20 shows a fragment of the service layer: an exclusive gateway with guards identifies types of operations (types of requests) and passes the control to corresponding outgoing sequence flows.

5.2 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 permission 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 5.1.1, 5.1.2, and 5.1.3 respectively. For mining data and resource perspectives, approaches (de Leoni & van der Aalst 2013) and (Burattin et al. 2013) were used. To discover subprocesses, all the event logs were preprocessed (the information needed for the events localization was learned from the

event attributes) and then a discovery approach (van der Aalst et al. 2015) was applied. The inductive miner technique (Leemans et al. 2014) was used as an underlying control-flow discovery method. The discovered Petri nets were converted to BPMN models using an algorithm presented in (Kalenkova et al. 2017).

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 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* (van der Aalst 2016). *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 the fitness function can be found in (van der Aalst et al. 2012). In this work we calculate the fitness value of each subprocess, merge alignments using the approach presented in (Verbeek & van der Aalst 2016), and then semi-automatically resolve merge conflicts to obtain the overall alignments and corresponding fitness values.

Having a fitting process model is not sufficient, because it is easy to construct a process model that can replay any trace, but overfits or underfits behavior observed in the event log. Thus, additional process metrics: *precision* and *generalization* are needed (van der Aalst 2016). The *precision* shows if the model does not allow too much behavior, i.e., it does not underfit the event log. The *generalization* metric indicates whether the model is general enough, i.e., the model is not overfitting.

Tab. 3 compares behavioral characteristics of the models discovered using localization information and models discovered directly from *M1-M5* event logs. For the behavioral characteristics of the models discovered from *TS* and *BS* event logs please refer to (van der Aalst et al. 2015). The process models constructed from the event logs *M1-M3* using the localized logs approach are more fitting than the corresponding process models constructed directly. However, the models discovered from the event logs *M4-M5* using localization information are less fitting, because root subprocesses in these models do not agree with the event logs on activities, which correspond to the beginning of child subprocesses. Skipping a shared event when aligning an event log and a parent process may lead to skipping all events of the corresponding subprocess. This can decrease the total fitness value.

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 the number of atomic activities, XOR (exclusive) gateways, AND (parallel) gateways, data objects, subprocesses, and swimlanes; number of control-flows; density (ratio of the number, 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 (Sánchez-González et al. 2010), the number of nodes, density, diameter, and depth have a negative correlation with the process model

Log traces	Fitness	Precision	Genera-lization
M1	0.94 / 0.88	0.14 / 0.15	1.00 / 0.99
M2	0.96 / 0.88	0.14 / 0.12	1.00 / 0.99
M3	0.94 / 0.93	0.12 / 0.11	0.99 / 0.99
M4	0.80 / 0.89	0.13 / 0.12	0.99 / 0.99
M5	0.75 / 0.93	0.14/ 0.11	1.00 / 1.00

Table 3: Behavioral characteristics of process models discovered from the event logs with and without application of the localized logs approach. The inductive miner was used as an underlying discovery technique.

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 activities, XOR, AND gateways	Num-ber of flows	Number of swim-lanes and sub-processes	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[8].

These structural characteristics of the discovered process models were compared with the characteristics of the BPMN models constructed manually. For that reason the existing Signavio 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 activities, XOR, AND gateways	Num-ber of flows	Number of swim-lanes and sub-processes	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[8].

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 a swimlane resembles a 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, in contrast to the approach of discovering large flat BPMN models from real-life event logs (Kalenkova et al. 2017), we showed that the proper identification of subprocesses, helps to discover readable and convenient process models.

Moreover, the decomposition (extraction of subprocesses) significantly reduces the discovery time. Thus, using a laptop with 2.4GHz Processor and 4GB RAM, it takes less than a minute to discover a BPMN model from any of the real-life event logs. In the meantime, the discovery of multi-perspective BPMN models without identification of subprocesses cannot be performed in a reasonable amount of time, using a computer with the same characteristics.

6 Conclusion

This paper reported on a methodology to discover BPMN models that integrate several perspectives and support the "divide-et-impera" paradigm, where the model is hierarchically broken down into several subprocesses.

Section 2 discusses the related work. In Section 3 the main concepts, such as Petri nets and BPMN modeling constructs, including data, resources and subprocesses, are introduced. Section 4 presents a novel integrated approach to discover hierarchical multi-perspective process models. Case studies are thoroughly described in Section 5.

The overall methodology can be summarized as follows. First, the event log is split into smaller event logs. The case studies reported in this paper illustrate that this is often possible by leverage of domain knowledge. For each sub-log, we can discover the control-flow structure of the process, which is used as a sort of process backbone, initially in the form of a labeled Petri net. Subsequently, the projected log is replayed on these individual, generally, much smaller, labeled Petri nets. Through the 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 hierarchical BPMN model and resource information can be added at each level of granularity. This results in a hierarchical BPMN model that integrates the different perspectives.

The work was thoroughly validated with real-life event logs. The validation illustrates the pros of our framework in giving insights that are more accurate, incorporating several process perspectives, and more understandable, with the process model structured in sub-models. An added bonus is that the splitting of the logs also improves performance.

Compared with the past research this framework is the first attempt to integrate several existing approaches with the aims of discovering multi-perspective hierarchical BPMN models. Note that, while the framework is showcased with several techniques, it is possible in the future to integrate other techniques and also extend to other perspective, for example, the time.

In fact, the time perspective is certainly an avenue for future work. Another important direction of work is the discovery of interacting processes in a form of pools and messages communicating these pools. It is also worth extending the framework to allow process analysts to compare the discovered BPMN models (a.k.a. "as-is models") with the supposed models (a.k.a. "to-be models") that encode the knowledge of the

stakeholders. By comparing the "as-is model" with the "to-be model", one can better pinpoint where deviations lie between what process participants should do and what they actually do.

Notes

1. $\mathcal{B}(X)$ denotes a set of multisets over X.
2. Note that invisible steps in the process model do not need to be observed in the event log.
3. $\nrightarrow$ denotes a partial function, which is defined for a subset of domain elements.
4. If transition t is not explicitly associated with a guard, we assume that $G(t) = true$.
5. Note that in order to avoid multiple firings of the start event, we assume that it cannot be enabled.
6. 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.
7. $\mathcal{P}_{NE}(X)$ defines a set of non-empty subsets of X.
8. The maximal, minimal and average values are separated by a slash, the values of metrics for different types of elements are separated by comma.

References

Batoulis, K., Meyer, A., Bazhenova, E., Decker, G. & Weske, M. (2015), Extracting decision logic from process models, *in* 'CAiSE 2015', Vol. 9097 of *LNCS*, Springer, pp. 349–366.

Bazhenova, E., Bülow, S. & Weske, M. (2016), Discovering decision models from event logs, *in* 'BIS 2016', Vol. 255 of *LNBIP*, Springer, pp. 237–251.

Bergenthum, R., Desel, J., Lorenz, R. & Mauser, S. (2007), Process mining based on regions of languages, *in* 'Business Process Management', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 375–383.

Burattin, A., Sperduti, A. & Veluscek, M. (2013), Business models enhancement through discovery of roles, *in* 'IEEE Symposium on Computational Intelligence and Data Mining 2013', pp. 103–110.

Conforti, R., Dumas, M., García-Bañuelos, L. & La Rosa, M. (2014), Beyond tasks and gateways: Discovering BPMN models with subprocesses, boundary events and activity markers, *in* 'Business Process Management: 12th International Conference 2014, Proceedings', Springer, pp. 101–117.

Conforti, R., Dumas, M., García-Bañuelos, L. & La Rosa, M. (2016), 'BPMN miner: Automated discovery of BPMN process models with hierarchical structure', *Information Systems* **56**, 284 – 303.

de Leoni, M. & van der Aalst, W. (2013), Data-aware process mining: Discovering decisions in processes using alignments, *in* 'Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013', pp. 1454–1461.

De Smedt, J., Hasic, F., vanden Broucke, S. & Vanthienen, J. (2017), Towards a holistic discovery of decisions in process-aware information systems, *in* 'BPM 2017', Vol. 10445 of *LNCS*, Springer.

De Weerdt, J., van den Broucke, S. & Caron, F. (2014), Bidimensional process discovery for mining BPMN models, *in* 'BPM 2014 International Workshops', Springer, pp. 529–540.

Decision Model and Notation (DMN) V1.1 (2016).
URL: *http://www.omg.org/spec/DMN/1.1/*

Dijkman, R., Dumas, M. & Ouyang, C. (2008), 'Semantics and analysis of business process models in BPMN', *Information & Software Technology* **50**(12), 1281–1294.

Kalenkova, A., de Leoni, M. & van der Aalst, W. (2014), Discovering, analyzing and enhancing BPMN models using ProM, *in* 'Proceedings of the BPM Demo Session 2014 Co-located with BPM 2014', pp. 36–40.

Kalenkova, A., van der Aalst, W., Lomazova, I. & Rubin, V. (2017), 'Process mining using BPMN: Relating event logs and process models', *Software & Systems Modeling* **16**(4), 1019–1048.

Kheldoun, A., Barkaoui, K. & Ioualalen, M. (2015), Specification and verification of complex business processes - A high-level Petri net-based approach, *in* 'Business Process Management: 13th International Conference 2015, Proceedings', Springer, pp. 55–71.

Leemans, S., Fahland, D. & van der Aalst, W. (2014), Discovering block-structured process models from incomplete event logs, *in* 'Application and Theory of Petri Nets and Concurrency', Vol. 8489 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 91–110.

Mannhardt, F. (2018), Multi-perspective process mining, PhD thesis, TUE: Department of Mathematics and Computer Science.

Mannhardt, F., de Leoni, M., Reijers, H. & van der Aalst, W. (2016), Measuring the precision of multi-perspective process models, *in* 'Business Process Management Workshops', Lecture Notes in Business Information Processing, Springer, Germany, pp. 113–125.

Muehlen, M. & Recker, J. (2008), 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', Vol. 5074 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 465–479.

Object Management Group (2013), 'Business Process Model and Notation (BPMN) 2.0.2'.
URL: *http://www.omg.org/spec/BPMN/2.0.2/PDF*

Rozinat, A. (2010), Process mining : conformance and extension, PhD thesis, TUE : Department of Industrial Engineering and Innovation Sciences.

Rozinat, A. & van der Aalst, W. (2006), Decision mining in ProM, *in* 'Business Process Management', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 420–425.

Sánchez-González, L., García, F., Mendling, J., Ruiz, F. & Piattini, M. (2010), Prediction of business process model quality based on structural metrics, *in* '29th International Conference on Conceptual Modeling', Vol. 6412 of *Lecture Notes in Computer Science*, Springer, pp. 458–463.

van der Aalst, W. (2016), *Process Mining – Data Science in Action, Second Edition*, Springer-Verlag Berlin Heidelberg.

van der Aalst, W., Adriansyah, A. & van Dongen, B. (2011), Causal nets: A modeling language tailored towards process discovery, *in* '22nd International Conference on Concurrency Theory', Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 28–42.

van der Aalst, W., Adriansyah, A. & van Dongen, B. (2012), 'Replaying history on process models for conformance checking and performance analysis', *Data Mining and Knowledge Discovery* **2**(2), 182–192.

van der Aalst, W., Kalenkova, A., Rubin, V. & Verbeek, E. (2015), Process discovery using localized events, *in* 'Application and Theory of Petri Nets and Concurrency', Vol. 9115 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 287–308.

van der Aalst, W., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E. & Günther, C. (2008), 'Process mining: a two-step approach to balance between underfitting and overfitting', *Software & Systems Modeling* **9**(1), 87.

van der Aalst, W., Weijter, A. & Maruster, L. (2003), 'Workflow mining: Discovering process models from event logs', *IEEE Transactions on Knowledge and Data Engineering* **16**, 2004.

van Dongen, B. (2015), 'Bpi challenge 2015'.
URL: *https://data.4tu.nl/repository/uuid:31a308ef-c844-48da-948c-305d167a0ec1*

van Dongen, B., Medeiros, A., Verbeek, H., Weijters, A. & van der Aalst, W. (2005), The prom framework: A new era in process mining tool support, *in* 'Application and Theory of Petri Nets 2005', Vol. 3536 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 444–454.

Verbeek, H. & van der Aalst, W. (2016), Merging alignments for decomposed replay, *in* 'Application and Theory of Petri Nets and Concurrency: 37th International Conference 2016, Proceedings', Springer International Publishing, pp. 219–239.

Ye, J., Sun, S., Song, W. & Wen, L. (2008), Formal semantics of BPMN process models using YAWL, *in* 'Second International Symposium on Intelligent Information Technology Application', Vol. 2, pp. 70–74.