Enhanced Transformation of BPMN Models with Cancellation Features

Giorgi Lomidze[1], Daniel Schuster[2,1][0000−0002−6512−9580], Chiao-Yun Li[2,1], and Sebastiaan J. van Zelst[2,1][0000−0003−0415−1036]

[1] RWTH Aachen University, Aachen, Germany
[2] Fraunhofer Institute for Applied Information Technology, Sankt Augustin, Germany

Abstract. Canceling ongoing process instances is a natural phenomenon in practice. As such, modeling cancellation behavior is supported in the Business Process Model and Notation (BPMN) via exception events. Event-data-driven analysis techniques using such process models, e.g., conformance checking, require converting the BPMN model into a formal process modeling representation, i.e., Petri nets. However, the existing transformation of BPMN models with exception events renders a classical Petri net, with various additional modeling constructs to mimic the exception behavior. Using such a model in a subsequent analysis renders an infeasible computational complexity. Hence, this paper presents a novel conversion of BPMN models with exception events into reset nets, significantly reducing the number of required invisible transitions in the corresponding transformation. Our results show that the enhanced conversion reduces the computational effort of using the converted models for conformance checking.

Keywords: Business Process Management · Business Process Modeling · BPMN · Petri nets.

1 Introduction

Several business process modeling languages exist that allow modeling the behavior of the processes in an organization [19]. The Business Process Model and Notation (BPMN) [20] language is a commonly used standard for business process modeling. BPMN is a graph-based language and includes over 50 distinct modeling elements. Among these elements are *exception events*, which allow one to specify exceptional behavior w.r.t. the default process behavior, e.g., a sensor value exceeding a certain threshold may require a special recovery procedure.

Whereas exceptions refer to *non-standard behavior*, adequate handling of an exception that occurred is vital. To verify whether the appropriate exception handling procedure is followed, *conformance checking techniques* [8] can be applied. Such techniques compare historical process executions stored in the information system of a company, i.e., referred to as *event data*, with a reference process model describing the main control flow of a process (and corresponding exception handling procedures).

Most conformance checking techniques are not designed to use BPMN as an input, i.e., they use *Petri nets* [23]. When applying conformance checking techniques on a process modeled in BPMN, the model is first transformed into a Petri net, after which the conformance checking algorithm is invoked. However, in the case of BPMN models with exception events, such a translation renders a Petri net with many additional modeling constructs to mimic the exception behavior. Applying conformance checking on such a transformed Petri net model is often infeasible in the context of computational complexity.

Transforming BPMN models with cancellation features to Petri nets is partly addressed in the literature (see Tab. 1 for an overview) and applying algorithms on the resulting Petri nets lacks efficiency. Hence, we propose an enhanced transformation that solves these limitations. Our transformation uses the concept of *reset nets* (R-nets) [14], i.e., Petri nets enhanced with means to "reset behavior". The main advantage of our proposed transformation is that the number of elements required to model the same behavior as the corresponding BPMN model is significantly reduced w.r.t. the existing transformation procedures. Such a smaller model size, in turn, is beneficial for the computational complexity of model-driven analysis techniques, e.g., conformance checking.

We conducted experiments with BPMN models, including exception events, comparing the existing transformation approach to our newly proposed transformation. Our experiments explicitly focus on conformance checking analysis of the transformed models. Our experiments confirm that the resulting R-nets, i.e., obtained by applying our newly proposed transformation rules, yield significantly reduced computational complexity in the context of conformance checking.

The remainder of this paper is structured as follows. In Section 2, we discuss related work. In Section 3, we present background concepts, i.e., BPMN models and R-nets. Section 4 presents our method. In Section 5, we present the evaluation of our proposed transformation. Section 6 concludes this paper.

2 Related Work

A general overview of the BPMN language is presented in [20]. Here, we discuss mappings from BPMN to other modeling languages. In Tab. 1, we present an overview of these approaches, including our approach. The table shows the supported constructs in the conversion, i.e., sub-process support, internal/external exceptions, terminations, timeouts, and OR-join constructs. Note that timeouts and OR-Joins are easily supported in our proposed transformation. However, for simplicity and brevity, they are not described in this work.

Arbab et al. [4] propose to convert BPMN models to the *Reo language* [3] and focus on model verification and compliance analysis. Kheldoun et al. [16] suggest mapping BPMN models with cancellation to *RECAT nets*, i.e., a tree-based structure that utilizes Petri nets. Kherbouche et al. [17] propose model checking for BPMN by exploiting a transformation to *Kripke structures*. Various transformations of BPMN to *Yet Another Worfklow Language (YAWL)* exist [10,11,26,27]. The transformations support sub-processes, internal exceptions,

Table 1: Overview of BPMN transformations to other modeling languages. Per construct, a ✓-symbol indicates full support; a (✓)-symbol indicates restrictions or missing information on the exact mapping of the corresponding construct.

Author	**Year**	**Target**	**Supported BPMN Constructs**					
			Sub-process	Int. Exception	Ext. Exception	Termination	Timeout	OR-Join
van der Aalst et al. [1]	2002	Petri net	(✓)					
Dijkman et al. [12]	2007	Petri net	✓	(✓)	✓	(✓)	(✓)	✓
Raedts et al.[22]	2007	Inhibitor net	✓				✓	
Arbab et al. [4]	2008	Reo	✓	✓	✓	(✓)	✓	(✓)
Decker et al.[10]	2008	YAWL	✓	✓	(✓)	(✓)	✓	✓
Ou-Yang and Lin [21]	2008	Colored Petri net	✓					(✓)
Ye et al.[27]	2008	YAWL	✓	(✓)			✓	✓
Ye and Song [26]	2010	YAWL	✓	(✓)	✓		✓	✓
Decker et al.[11]	2010	YAWL	✓	✓	(✓)	(✓)	✓	✓
Kherbouche et al.[17]	2013	Kripke structure	✓					✓
Kheldoun et al. [16]	2015	RECAT net	✓	✓	✓		✓	
Dechsupa et al. [9]	2019	Colored Petri net	✓					
This work	2022	reset net	✓	✓	✓	✓	(✓)	(✓)

timeouts, and OR-joins. The YAWL model is projected to a reset net, where backward reasoning is required to handle OR-joins. The concept of cancellation is directly translated to reset arcs. Additional transformations supporting cancellation behavior exist for YAWL models, e.g., to Inhibitor nets [24] and Generalized Stochastic Petri nets with inhibitor arcs [6].

Several authors focus on transforming BPMN models to Petri nets (or subclasses thereof). A general overview of transformations from arbitrary process modeling languages (including BPMN) to Petri nets is presented in [18]. Most existing mappings have in common that the essential BPMN workflow components, i.e., start/intermediate/end events, sequence flows, tasks, and AND-/XOR-/OR-gateways, are mapped using the same approach. In [1] van der Aalst et al. propose a complete set of transformation rules for *Workflow graphs* (which can be seen as a generalization of simple BPMN models). In [22], a mapping from BPMN to Petri nets with inhibitor arcs is presented for model checking. The inhibitor arcs are used to model timeout events. Dechsupa et al. [9] map a BPMN model to a Colored Petri Net (CPN). An effective way of transforming block-structured OR-joins and OR-splits is provided. In both works mentioned above, general cancellation is not supported. The transformation approach presented in [12] is most complete w.r.t. cancellation behavior. The presented algorithm first transforms the basic workflow patterns to Petri net elements with equivalent behavior. Subsequently, sub-process features are mapped. External and internal sub-process cancelation events and nested sub-processes are transformed. Since Petri nets cannot reset certain areas by just firing one transition, the tokens are bypassed to the end of the cancellation area once the exception event has been fired.

In summary, support for the conversion of BPMN models that include cancelation behavior to Petri net-oriented models is limited. To obtain a Petri net from a BPMN model with advanced cancellation constructs, a mapping via YAWL is required. To the best of our knowledge, the transformation proposed in this paper is the first direct mapping to R-nets that covers all relevant BPMN cancellation features.

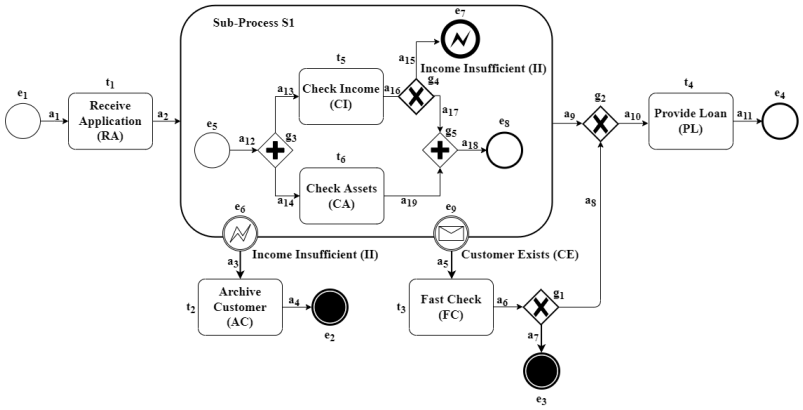


Fig. 1: Example BPMN model containing the core modeling concepts assumed in this paper. We use the model as a running example.

3 Background

In this section, we present the concepts used in this paper. We specify the class of supported BPMN models in Section 3.1. In Section 3.2, we present reset nets.

3.1 Business Process Model and Notation

The *Business Process Model and Notation* (BPMN) [20] language is commonly used for business process modeling. We use a strict subset of modeling constructs, i.e., start/end events, tasks (allowed to represent sub-models), gateways, exception events, and some additional modeling assumptions. Consider Fig. 1, in which we depict an example BPMN model describing a simplified loan-application process, including all modeling elements considered in this paper.

The model in Fig. 1 describes that first the *Receive Application* (RA) activity is executed. After this, sub-process $S1$ is invoked in which an *income check* (CI) and *asset check* (AC) are performed. The client's income can be insufficient, triggering an *Income Insufficient* (II) exception. After such an exception occurs, the *customer record is archived*, and the process is terminated. During sub-process $S1$, at any point in time, a background check can reveal that the customer already exists in the system. In this case, the sub-process is canceled, and a *fast check* is performed. After the fast check, the global process either continues or is canceled. For the two exceptions described, different *resolution strategies* are modeled, i.e., exception *Income Insufficient* is always followed by task t_2 (*Archive Customer*) and subsequently the entire process is terminated whereas external exception *Customer Exists* is always followed by task t_3 (*Fast Check*), after which we either terminate the process, or, we continue to process at gate g_2. Observe that event *Customer Exists* can occur at any point in time, whereas the end event *Income Insufficient* can only happen after the task *Check Income*. We formally define BPMN models, i.e., as considered in this paper, as follows.

Definition 1 (BPMN Model). *Let Σ denote the universe of activity labels. A* BPMN Model *is a tuple $M{=}(N, T, G^{\times}, G^{+}, e^{s}, e^{t}, E^{ie}, E^{ii}, F, \ell, \xi)$, where*[3] *:*

- N *is a set of* nodes,
- $T{\subseteq}N$ *is a set of* tasks,
- $G^{\times}{\subseteq}N$ *is a set of* exclusive gateways,
- $G^{+}{\subseteq}N$ *is a set of* parallel gateways,
- $e^{s}{\in}N$ *is a* regular start event,
- $e^{t}{\in}N$ *is a* regular end event,
- $E^{ie}{\subseteq}N$ *is a set of* irregular end events,
- $E^{ii}{\subseteq}N$ *is a set of* interrupting intermediate events,
- $F{\subseteq}N{\times}N$ *is the sequence flow relation,*
- $\ell\colon T{\nrightarrow}\Sigma$ *is a partial labeling function,*
- $\xi\colon E^{ie}{\rightarrow}T$ *is a function assigning an intermediate event to a task, signaling an exception and thus interrupting the execution of the task.*

Compared to the standard BPMN specification, we assume exactly one "regular end event" is present per (sub-)model (e.g., e_4 and e_8 in Fig. 1), i.e., representing *normative termination*. We assume that the set of tasks can be further divided in a set of labeled activities (i.e., those tasks $t{\in}T$ with $\ell(t){\in}\Sigma$) and a set of sub-models (those tasks $t{\in}T$ with $\ell(t){=}\perp$). If an activity refers to a sub-model, we assume that the underlying model recursively adheres to Definition 1, e.g., consider task $S1$ in Fig. 1. However, for simplicity, we do not formally define this. We assume that some hierarchy H is present that captures whether a model is contained in another model. Further, we assume that any node that is part of a sub-model is not explicitly contained in the set of nodes N of its parent model. The root model is assumed to be at hierarchy level H_0, and any model that is contained in a task of the root model is at level H_1. In general, a model that is contained in a task of a model at level H_i is at level H_{i+1}, e.g., in Fig. 1 all elements in sub-model $S1$ are at level H_1, all other models are at level H_0. Let $M_i{=}(N_i, T_i, G_i^{\times}, G_i^{+}, e_i^{s}, e_i^{t}, E_i^{ie}, E_i^{ii}, F_i, \ell_i, \xi_i)$ be a (sub)-BPMN model at some level $i{\geq}0$ and let $M_{i+1}{=}(N_{i+1}, T_{i+1}, G_{i+1}^{\times}, G_{i+1}^{+}, e_{i+1}^{s}$, $, e_{i+1}^{t}, E_{i+1}^{ie}, E_{i+1}^{ii}, F_{i+1}, \ell_{i+1}, \xi_{i+1})$ be any model at hierarchy level H_{i+1}, i.e., it is contained in one of the nodes in T_i. We assume that there exists a partial function $\delta\colon E_{i+1}^{ie}{\nrightarrow}E_i^{ii}$ that associates certain *irregular end events* at hierarchy level H_{i+1} to a corresponding *intermediate event* in E_i^{ii}, e.g., consider end event e_7 in $S1$ which is linked to intermediate event e_6. Note that, in case M_{i+1} is represented by some $t{\in}T_i$, then for every $e{\in}E_{i+1}^{ie}$ s.t. $\delta(e){\neq}\perp$ we have $\xi_i(\delta(e)){=}t$.

3.2 Reset Nets

Petri nets [23] are a mathematical model, often used for verification and automation purposes. Their mathematical definition yields Petri nets unambiguous in terms of the language they describe, i.e., as opposed to most business-oriented graphical modeling languages. A Petri net, see Fig. 2 as an example, describes a bipartite graph with two types of vertices, i.e., *places* (visually represented as circles) and *transitions* (visually represented as boxes). Places only connect to

[3] We omit OR gateways, yet, the framework is easily extended with OR-support.

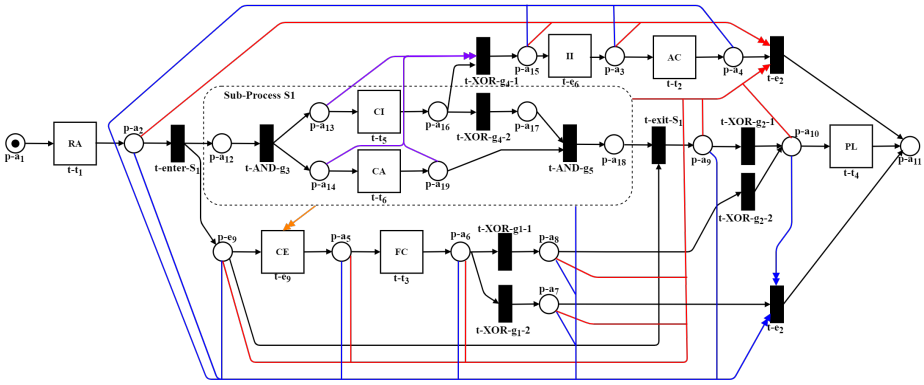


Fig. 2: Example reset-net (R-net) modeling the same behavior as the BPMN model in Fig. 1. Regular arcs are depicted as black single-headed arcs. Reset arcs are shown using double-headed arcs in different colors. Each color represents another type of cancelation behavior. A reset arc directly connecting to sub-process $S1$ is assumed to empty all places in the sub-process.

transitions and vice versa. The places of a net are used to represent the *state* of the net (referred to as the net's *marking*), i.e., using so-called "tokens", the transitions are used to manipulate the state. For example, the place $p{-}a_1$ in Fig. 2 contains *one token*. A transition in a Petri net is *enabled* iff all its "pre-places" connecting to it (places with an outgoing arc from the place to the transition) contain a token. For example, in Fig. 2, transition $t{-}t_1$ is enabled. An enabled transition can *fire*. When we fire an enabled transition, it consumes a token from each "pre-place" and creates a new token for each of its "post-places". For example, if we fire transition $t{-}t_1$ in Fig. 2, we obtain the new marking $[p{-}a_2]$. Reset nets (R-nets) extend Petri nets with an additional arc type. A *reset* arc, i.e., the double-headed arcs in Fig. 2, consumes all tokens from the source place it connects to, upon firing the target transition it connects to. For example, if at least one token resides in $p{-}a_2$ in Fig. 2 and we fire $t{-}e_2$ or $t{-}e_3$, all tokens in $p{-}a_2$ are removed. Unlike regular arcs, the places associated with a reset arc can be empty to fire a transition, i.e., no tokens may exist in the places.

Definition 2 (Reset Net). *Let P denote a set of places, let T denote a set of transitions ($P{\cap}T{=}\emptyset$), let $F{\subseteq}(P{\times}T){\cup}(T{\times}P)$ denote the flow relation, let $R{:}\ T{\to}\mathcal{P}(P)$[4] denote the reset arc relation, let Σ denote the universe of transition labels, let $\tau{\notin}\Sigma$ and let $\ell{:}\ T{\to}\Sigma{\cup}\{\tau\}$ denote a labeling function. A* reset net *N is a tuple (P, T, F, R, ℓ).*

Let $N{=}(P, T, F, R, \ell)$ denote a Reset net which will be called R-net for convenience. A *marking* m of N is a multiset of places, i.e., $m{\in}\mathcal{M}(P)$, tuple (N, m) is referred to as a *marked net*. Given marked net (N, m), a transition $t{\in}T$ is enabled

[4] $\mathcal{P}(X)$ denotes the *power set* of set X.

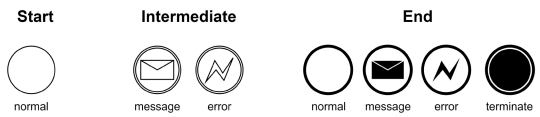


Fig. 3: The start, end, and intermediate events considered in this paper.

iff $\forall p{\in}P\ (m(p){\geq}F(p,t))$. An enabled transition t in marking m can *fire*, yielding a new marking m', where $m'(p){=}0, \forall p{\in}R(t)$ and $m'(p){=}m(p){-}F(p,t){+}F(t,p), \forall p{\in}P{\setminus}R(t)$.

4 Enhanced Transformation of BPMN Models

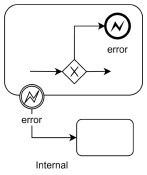
This section presents our proposed transformation of BPMN models to R-nets. We first present additional assumptions on the input BPMN model. Subsequently, we show how we iteratively build a corresponding R-net.

4.1 Modeling Assumptions

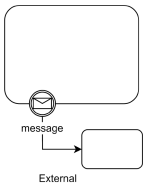
This section presents the core assumptions of our transformation algorithm w.r.t. the input BPMN model. We discuss the different types of events and the assumptions on the modeling of exception handling mechanisms.

Events Considered Contrary to general BPMN, we assume that a given process model has one unique *None End Event* per (sub)-process. For example, the BPMN model depicted in Fig. 1 has one unique end event (e_4) and so does the sub-process $S1$ (e_8). As such, every (sub)-process is assumed to have a clear unique *start* and *end event*, which we refer to as the *regular start/end events*. Furthermore, we assume that the start and end events have one unique outgoing, respectively incoming, arc. All other end events that are part of the BPMN model are referred to as *irregular end events*. For example, the model in Fig. 1 contains three irregular end events, i.e., e_2, e_3 and e_7. Consider Fig. 3, in which we depict an overview of the start, end, and intermediate events considered in this paper. Observe that the events depicted in Fig. 3 are exemplary, i.e., other events (e.g., the intermediate interrupting timer) also apply to the proposed cancellation behavior. However, as such events have a merely semantic meaning (i.e., a time-based interrupt rather than a message-based interrupt), we do not consider these in detail.

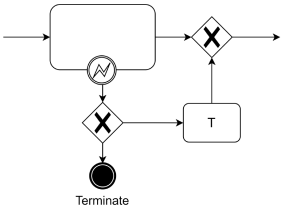
Exception Modeling We assume two primary mechanisms for the modeling of exceptions and cancellation behavior, i.e., visualized in Figures 4a and 4b. The intermediate events and the message, error, and terminate end events (cf. Fig. 3) are used for modeling exceptions and cancellation behavior. The message and error end events are used to signal an exception and are connected to an intermediate event at the boundary of the process in which they are contained.



(a) Internal exceptions occur at a specific state of a sub-process and are of a *throw-catch* nature.



(b) External exceptions can happen at any point in the sub-process.



(c) The process at the higher level either terminates, or the process flow is assumed to continue after the regular termination of the canceled activity.

Fig. 4: The two proposed cancellation mechanisms (Figures 4a and 4b) and subsequent resolution handling (Fig. 4c) adopted in this paper.

Consider Fig. 4a, which schematically represents the use of an end event and an associated "catching" intermediate event. If the end event occurs at hierarchy level H_i for $i \geq 1$, the corresponding intermediate event is located at hierarchy H_{i-1}. An unmapped intermediate event associated with some activity (possibly a sub-process) indicates that the activity it connects to can be canceled at any point in time, i.e., as visualized in Fig. 4b.

In both cases, after activating an intermediate event, a *resolution strategy* applies, i.e., depicted schematically in Fig. 4c. After observing the intermediate event related to cancellation, the process at the higher level either terminates, or the process flow is assumed to continue after the regular termination of the canceled activity. Observe that various other tasks may be modeled between the exception event and corresponding termination (as exemplified in Fig. 1). We assume that these tasks *cannot be canceled* (e.g., t_2 and t_3 in Fig. 1).

4.2 Building Reset Nets

This section presents our proposed transformation. Our approach consists of three steps, i.e., *arc mapping*, *node mapping*, and *cancellation mapping*. The first two steps are relatively straight-forward and adopted from existing BPMN transformations [9,13,15,22]. The *Cancellation Mapping* step can be seen as the main contribution of this work. In the remainder, we explain each step. The final cancellation step is only applied to a model when all sub-models have been completely (recursively) transformed into an R-net.

Arc Mapping In the first step of the transformation, each arc in the BPMN model, i.e., the F component of Definition 1, is converted into a place in the resulting R-net.

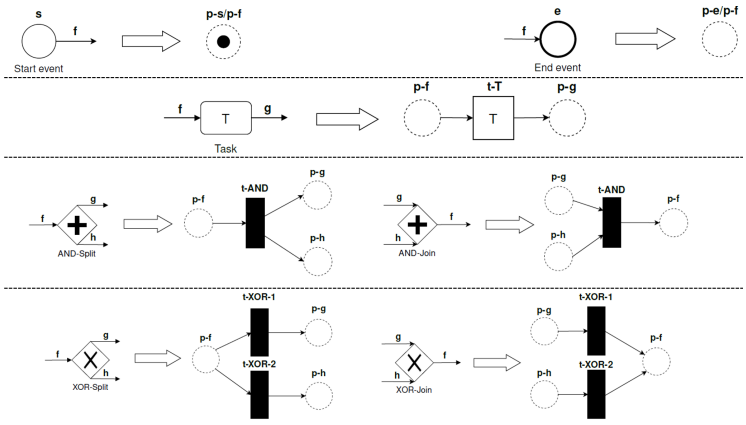


Fig. 5: Overview of node mappings of regular start/end events, tasks, AND-constructs, and XOR-constructs.

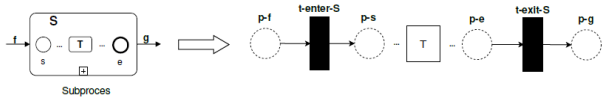


Fig. 6: Mapping procedure for sub-process tasks.

Node Mapping In the second step of the approach, we convert the nodes of the BPMN model into corresponding R-net elements. In Fig. 5, we show a schematic overview of node mappings of regular start/end events, tasks, AND-constructs, and XOR-constructs. Observe that, for the start event `s`, we add a token to the corresponding place `p-f` of its unique outgoing arc. The end event is ignored. Tasks are converted to a transition for which the incoming and outgoing places are equal to the places corresponding to the incoming/outgoing arcs of the tasks in the BPMN model. AND-splits and AND-joins are converted into transitions that generate/consume a token in/from each place corresponding to an outgoing/incoming arc of the split/join. For an XOR join, a transition is created per outgoing/incoming arc of the split/join that consumes/produces a token in the place representing the outgoing/incoming arc of the split/join. The mapping of tasks that describe a sub-process is straightforward, i.e., see Fig. 6. Place `p-f` is connected to a new invisible transition (`t-enter-S`), which is in turn connected to `p-s`. A symmetrical procedure is adopted for `p-e` and `p-g`. Consider Fig. 7, in which we depict the transformation of the running example BPMN model after applying the first two steps (i.e., arc and node mapping).

Cancellation Mapping This section presents the mappings for exception handling. Consider Fig. 8, in which we schematically depict the transformation rule for *internal exceptions*. The choice construct connected to arcs `A`, `B`, and `C` is already part of the partially complete net, i.e., represented by the choice con-

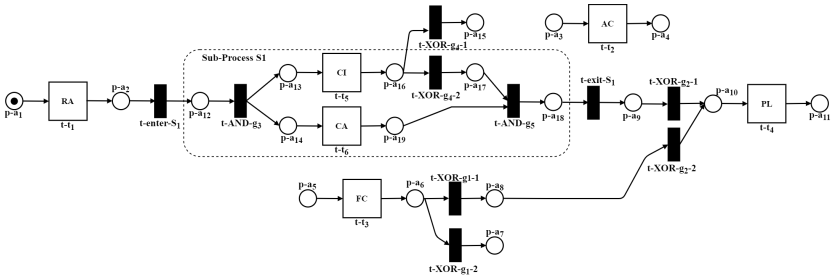


Fig. 7: Transformation of the running example BPMN model (Fig. 1) after the arc/node mapping step.

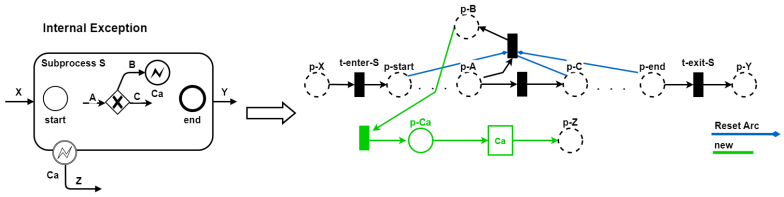


Fig. 8: Schematic visualization of internal exception handling.

struct formed by places `p-A`, `p-B`, and `p-C`. Firstly, we create a new net fragment, representing the intermediate error event `Ca` (represented by the place `p-Ca` and transition `Ca`), yielding a token in `p-Z`. Secondly, we attach reset nets from all places to the transition marking `p-B`. Finally, we connect `p-B` by means of a fresh invisible transition to place `p-Ca`. If multiple end events with label `Ca` exist, the previously described second and third steps can be replicated for the error event. Observe that, if `p-B` is marked, all places related to sub-process `S` are unmarked. The mapping procedure for external exception handling is depicted in Fig. 9. A place `p-Ex` is created that has an incoming arc from the `t-enter-S` transition and is connected to the exception transition `Ex`. Place `p-Ex` is connected by means of an outgoing arc to `t-exit-S`. If the transition labelled `Ex` is connected to very place in `S` by means of a reset arc.

Observe that the R-net in Fig. 2 is the result of applying the transformation algorithm on the running example BPMN model.

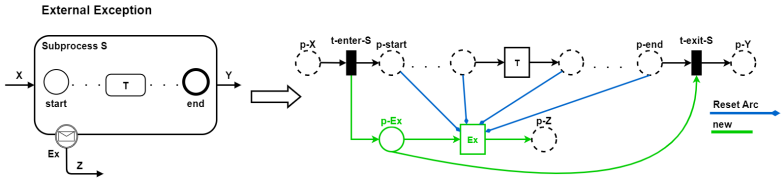


Fig. 9: Schematic visualization of external exception handling.

Table 2: Overview of the characteristics of the models used in the experiments.

Model No.	Tasks	Events	XOR Gates	AND Gates	Sub-Processes	Exceptions
I	14	10	2	6	1	2
II	11	10	4	4	1	2
III	14	30	8	6	8	5
IV	9	20	10	4	5	3

5 Evaluation

In this section, we present the evaluation of our approach. We compare the performance of *conformance checking*, using different transformations of BPMN models with cancellation behavior. In Section 5.1, we present the experimental setup.[5] Section 5.2 presents the results.

5.1 Experimental Setup

In this section, we present the experimental setup of our experiments. We describe the process adopted to generate input data, which is divided into two steps, i.e., *model construction* and corresponding *event data generation*. Additionally, we describe an alternative *transformation* of the BPMN models created, which we use to compare our approach against. Finally, we briefly provide details on the conformance checking techniques studied.

Model Construction None of the existing work on generating process models, e.g., Burattin et al. [7], propose a BPMN generator that includes support for the (generic) addition of cancellation behavior. Hence, we design four BPMN models of varying sizes with different control-flow behavior and cancellation behavior. An overview of the characteristics of these models is presented in Tab. 2.

Event Data Generation To generate data, as a first step, a noise-free language is generated exhaustively for each BPMN model. For models with looping behavior, the language is trimmed to have a maximum trace length of 25. Within the language generation, the assumed behavior of (potentially ambiguous) cancellation features is as follows. (i) An external exception event may fire even after the last enabled task has been executed inside a sub-process, and (ii) Tasks parallel to error or terminate end events may fire whenever enabled, even after any preceding task of the end event was executed. We add artificial noise to the generated languages. For each language, we randomly select 200 distinct *sub-logs* containing 15 traces, yielding a total of 3000 traces. Given a *noise probability*

[5] All models (i.e., both designed and obtained by means of transformation), event data generated and computational results, are available via https://drive.google.com/drive/folders/10Q11FfRu_Lf9kA1moR2gikQc9HAwnsNv?usp=sharing. The code used in the experiments is available via https://github.com/require-gio/pm4py-resetnet.

$(0\%, 10\%, 30\%, 50\%)$, we iterate over each event of each trace and apply a *noise operation* with the corresponding probability. The noise operation is either an addition, substitution, or removal operation. The concrete choice of which operation to apply is made with $\frac{1}{3}$ for each operation type.

Transformations Considered Aside from the proposed transformation in this paper, another transformation, i.e., based on Dijkman et al. [12] (referred to as *Dijkman transformation*), is considered. The *Dijkman transformation* maps BPMN models with cancellation behavior to regular Petri nets.[6] However, the approach requires slight changes to ensure that it can transform the BPMN models in a language-equivalent manner. *Dijkman* considers intermediate events inside sub-processes as triggers for attached internal exception events. Our mapping, however, assumes that error end events are responsible for triggering internal exceptions. The input models are, therefore, manually adjusted for the *Dijkman transformation* such that error end events are exchanged with intermediate events. Terminate end events behave similarly to internal exceptions with the difference that, instead of triggering an exception event, they provide a direct path to the final state of the sub-process they are inside. Hence, the terminate events are manually replaced by intermediate events in the input models and then mapped the same way as internal exceptions, however, without an exception flow mapping.

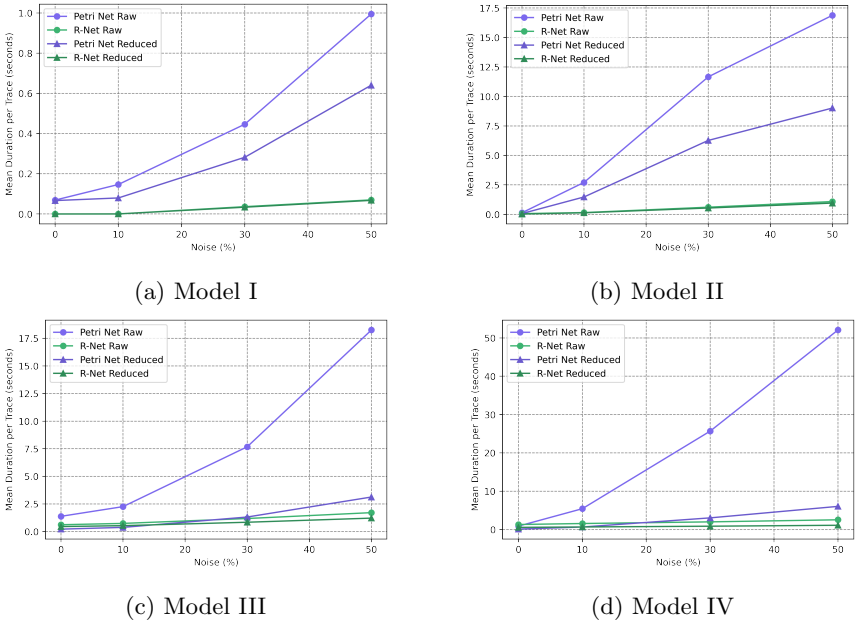
Conformance Checking In our experiments, we consider how well the transformed models perform when applying *conformance checking* on the models. To this end, we propose to compute *optimal alignments* as conformance checking artifacts. Computing alignments requires a Petri/R-net and an event log. We assess the A^* (marking equation) implementation of alignment computation [2].[7] During the experiments, several performance statistics are recorded: mean computation times, queued sates, visited states, traversed arcs, and the number of linear programs solved. The machine used for the experiments comprises an `AMD Ryzen 7 2700X 8-core 4.00 GHz` processor, `32 GB RAM` and a `Windows 10 operating system`.

5.2 Results

This section presents the results of our experiments, i.e., as described in Section 5.1. We discuss general time efficiency and search efficiency of computing alignments using the R-nets obtained by using our proposed approach and compare it with the *Dijkman transformation*. As both algorithms render many invisible transitions, we apply reduction rules on the obtained R/Petri-nets to reduce

[6] As there is no executable implementation available of the *Dijkman transformation*, we re-implemented the approach.

[7] Note that the A^* variant for reset/inhibitor nets has been implemented in `python`, extending the `pm4py` framework [5].



(a) Model I

(b) Model II

(c) Model III

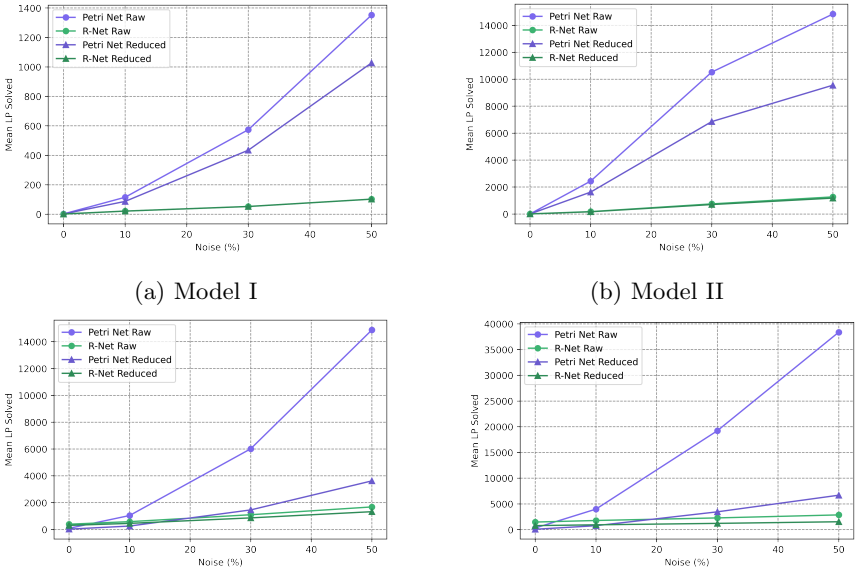
(d) Model IV

Fig. 10: Average alignment computation time per trace of our transformation ("R-Net") significantly outperforms the *Dijkman transformation* ("Petri net").

the number of invisible transitions. We use the reduction rules described in [25], yet, we only apply the non-exponential rules.

The mean computation times (in seconds) for the alignment calculation are visualized for each noise value and applied algorithm in Fig. 10. We observe that our transformation (R-Net in the figure) significantly outperforms the *Dijkman transformation* (Petri net in the figure). The model reduction has little influence on our transformation; however, it does have a significant positive effect on the time performance of the *Dijkman transformation*.

Since the *Dijkman transformation* generates more *invisible transitions* in the resulting Petri net (compared to the R-nets obtained by our transformation), we expect the underlying search efficiency to be better for the models based on our transformation. Hence, to further investigate the underlying effect of the observed results, we consider different experimental statistics related to search efficiency. Since the A^* approach uses Linear Programming (LP) internally (i.e., as a heuristic for the search), we depict the average number of LPs solved for each model/noise combination in Fig. 11. All four plots follow the same shape as the computation time results presented in Fig. 10. This similarity indeed hints strongly at a more efficient state-space traversal of the A^* algorithm. Other metrics investigated, i.e., the number of queued states visited states, and traversed arcs per search, yield the same insights. As such, we conclude that the models



(a) Model I

(b) Model II

(c) Model III

(d) Model IV

Fig. 11: Average number of LPs solved per trace of our transformation ("R-Net") significantly outperforms the *Dijkman transformation* ("Petri net").

obtained by our proposed transformation allow for significant efficiency in the state-space traversal of A^*-based alignment calculation.

5.3 Threats to Validity

We acknowledge that the number of models assessed in the evaluation is limited. At the same time, for all four models, we observe a similar pattern for increasing levels of noise. As such, we assume that the observed trends are also to be observed when conducting the experiments on a larger scale. We acknowledge that the transformation performed on the models used with the *Dijkman transformation* may slightly impact the performance. However, since the core problem of the *Dijkman transformation* is the excessive amount of invisible transitions in the model, we assume this effect to be negligible.

6 Conclusion

The possibility of canceling a process's main behavior is a natural phenomenon. Whereas BPMN provides rich support for cancellation features, there has been a lack of transformation algorithms that can convert said models to an alternative formal representation for further analysis. In this work, a structured mapping from BPMN to reset nets is proposed, which serves as the underlying execution

model for applying advanced analytical algorithms. The mapping of cancellation features has been defined in detail and is the first to include *terminate end events* and *internal exceptions triggered by end events*. The evaluation shows that the proposed approach produces models that allow more efficient computation of conformance checking artifacts.

Future Work We plan to extend the work proposed as follows. We aim to extend the approach to support a more extensive set of BPMN objects, i.e., there are many additional event types as well as gateways whose control flow behavior does not significantly differ from the ones considered in the presented work, e.g., interrupting escalation events, event gateways, timeouts, etc. We further aim to formally characterize classes of BPMN models that are supported by the proposed approach. Additionally, we aim to perform experiments on a larger scale. As a corresponding prerequisite, an additional interesting avenue for future work is the automated generation of BPMN models with cancellation features.

References

1. van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.: An alternative way to analyze workflow graphs. In: CAiSE 2002, Toronto, Canada, May 27-31, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2348. Springer (2002)
2. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.: Memory-efficient alignment of observed and modeled behavior. BPM Center Report **3** (2013)
3. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comput. Sci. **14**(3) (2004)
4. Arbab, F., Kokash, N., Meng, S.: Towards using reo for compliance-aware business process modeling. In: ISoLA 2008, Porto Sani, Greece, October 13-15, 2008. Proceedings. Communications in Computer and Information Science, vol. 17. Springer (2008)
5. Berti, A., van Zelst, S.J., van der Aalst, W.M.P.: Process mining for python (PM4Py): Bridging the gap between process-and data science. In: ICPM Demo Track 2019, Aachen, Germany, June 24-26, 2019. (2019)
6. Boonyawat, S., Vatanawood, W.: Transforming YAWL workflows with time constraints to generalized stochastic Petri nets. In: In 3rd International Conference on Software and e-Business (2019)
7. Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. In: BPM Demo Track 2016, Rio de Janeiro, Brazil, September 21, 2016. CEUR Workshop Proceedings, vol. 1789. CEUR-WS.org (2016)
8. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
9. Dechsupa, C., Vatanawood, W., Thongtak, A.: Hierarchical verification for the BPMN design model using state space analysis. IEEE Access **7** (2019)
10. Decker, G., Dijkman, R.M., Dumas, M., García-Bañuelos, L.: Transforming BPMN diagrams into YAWL nets. In: BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5240. Springer (2008)
11. Decker, G., Dijkman, R.M., Dumas, M., García-Bañuelos, L.: The business process modeling notation. In: Modern Business Process Automation - YAWL and its Support Environment. Springer (2010)

12. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and analysis of BPMN process models using Petri nets. Queensland University of Technology, Tech. Rep (2007)
13. Dijkman, R.M., Van Gorp, P.: BPMN 2.0 execution semantics formalized as graph rewrite rules. In: International Workshop on Business Process Modeling Notation. Springer (2010)
14. Dufourd, C., Finkel, A., Schnoebelen, P.: Reset nets between decidability and undecidability. In: ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1443. Springer (1998)
15. Kalenkova, A.A., van der Aalst, W.M.P., Lomazova, I.A., Rubin, V.A.: Process mining using BPMN: relating event logs and process models. Softw. Syst. Model. **16**(4) (2017). https://doi.org/10.1007/s10270-015-0502-0
16. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Specification and verification of complex business processes - A high-level petri net-based approach. In: BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9253. Springer (2015)
17. Kherbouche, O.M., Ahmad, A., Basson, H.: Using model checking to control the structural errors in BPMN models. In: RCIS 2013, Paris, France, May 29-31, 2013. IEEE (2013)
18. Lohmann, N., Verbeek, E., Dijkman, R.M.: Petri net transformations for business processes - A survey. Trans. Petri Nets Other Model. Concurr. **2** (2009)
19. Mili, H., Tremblay, G., Jaoude, G.B., Lefebvre, E., Elabed, L., El-Boussaidi, G.: Business process modeling languages: Sorting through the alphabet soup. ACM Comput. Surv. **43**(1), 4:1–4:56 (2010). https://doi.org/10.1145/1824795.1824799, https://doi.org/10.1145/1824795.1824799
20. OMG: url: https://www.omg.org/spec/BPMN/2.0/About-BPMN/ (2021), accessed at 30.11.2021
21. Ou-Yang, C., Lin, Y.: BPMN-based business process model feasibility analysis: a Petri net approach. International Journal of Production Research **46**(14) (2008)
22. Raedts, I., Petkovic, M., Usenko, Y.S., van der Werf, J.M.E.M., Groote, J.F., Somers, L.J.: Transformation of BPMN models for behaviour analysis. In: MSVVEIS-2007, Funchal, Madeira, Portugal, June 2007. INSTICC PRESS (2007)
23. Reisig, W.: Petri Nets: An Introduction, EATCS Monographs on Theoretical Computer Science, vol. 4. Springer (1985)
24. Terayawan, S., Vatanawood, W.: Transforming control-flow patterns of YAWL to Petri nets. In: In International Communication Engineering and Cloud Computing Conference (2019)
25. Verbeek, H.M.W., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Reduction rules for reset/inhibitor nets. J. Comput. Syst. Sci. **76**(2) (2010)
26. Ye, J., Song, W.: Transformation of BPMN diagrams to YAWL nets. J. Softw. **5**(4) (2010)
27. Ye, J., Sun, S., Song, W., Wen, L.: Formal semantics of BPMN process models using YAWL. In: 2008 Second International Symposium on Intelligent Information Technology Application. vol. 2. IEEE (2008)