Hindawi
Scientific Programming
Volume 2021, Article ID 6610795, 16 pages
https://doi.org/10.1155/2021/6610795

Research Article

A Petri Nets Evolution Method that Supports BPMN Model Changes

Zonghua Li[1] **and Zhengwei Ye**[2]

[1]*School of Computer Science and Technology, Huaiyin Normal University, 223300 Huai'an, Jiangsu, China*
[2]*School of Urban and Environmental Sciences, Huaiyin Normal University, 223300 Huai'an, Jiangsu, China*

Correspondence should be addressed to Zonghua Li; leeleaf@163.com

Received 18 November 2020; Revised 19 March 2021; Accepted 2 June 2021; Published 16 June 2021

Academic Editor: Mu-Chen Chen

The correctness of the business process modelling notation (BPMN) is essential for software success, and the BPMN formalization is the foundation of the correctness verification process. However, dynamically adapting the formalized BPMN model to changes in the BPMN model and protecting tokens from being lost in the remapping formalization are the main limitations of the BPMN formalization under changing business requirements. To overcome these limitations, an approach for evolving a Petri nets model according to the BPMN changes is proposed in this paper. In this approach, a check algorithm is designed to identify the differences between the original BPMN model and the updated BPMN model. Then, the evolution rules of the extended Petri nets (EPN) model are defined according to the results of the checking program. Finally, these evolution rules are described in the query/view/transformation operational mapping (QVTo) language and implemented in the Eclipse platform. We demonstrate the effectiveness of the evolution of the BPMN formalization using a case study of the Web Payment business system. Moreover, the dynamic evolution of the BPMN formalization can maintain the consistency between the original model and the updated model, and this consistency has been successfully verified.

1. Introduction

Business process modelling notation (BPMN) is used as a standard for modelling business processes in the early phase of system development and for instructing the design and development [1]. BPMN2.0 (http://www.omg.org/spec/BPMN/2.0) offers graph notations that can help business analysts and developers represent the process information of a business system. Therefore, the success of software development depends strongly on the correctness of the BPMN model [2, 3]. However, it is difficult to evaluate the correctness and ambiguity of the BPMN model because the notation specification does not include the semantics [4]. There are many initiatives for defining the formal execution semantics [5–9]. These formal definitions are used to evaluate the correctness, precision enforcement, and consistency of the collaboration of the BPMN model. Currently, the formal execution semantics of the BPMN model can be divided into two categories: those that use pure mathematics to describe the behaviours (such as Communicating Sequential Processes (CSP) [10], *Z* language, Formal Concept Analysis (https://www.upriss.org.uk/fca/fca.html), and Backus Naur Form (BNF) syntax [9, 11]) and those that use Petri nets [12] to formalize the behaviours and business flows [13]. Various workflow modelling languages, such as YAWL [14], which are based on Petri nets, can extend Petri nets to address more complex workflows [15–17] and can be directly integrated with JAVA applications [18]. These features enable that the Petri net to be a good candidate for formally defining the semantics of BPMN models [19] since the BPMN model is also flow-oriented [4].

However, the evolution of software artefacts are ubiquitous [20]. These artefacts include programs, data [21], requirements [22], documentation [23, 24], and modelling languages evolution [20]. Due to the change in requirements and advances in technology, software systems are not

immutable. So, the existing systems can be continuously modified, complemented, and improved to be adapted to changes in the external environment by requirements evolution. Thus, one of the main challenges facing BPMN formalization is the automatic adjustment of the formalized model when the BPMN model is changed. To overcome these limitations, a BPMN formalization approach that supports dynamic evolution is proposed in this paper.

This work makes the following main contributions: (a) a check module is designed to capture and record changes to the BPMN model. These changes can cause the target model to carry out in-place transformations; (b) an evolution approach is proposed, which includes a set of in-place transformations; and (c) a blackboxing library is designed for detecting the objects that have changed in the source BPMN model, and the in-place transformations have been implemented in the Eclipse framework.

This paper focuses on EPN model evolution when the BPMN model changes. We not only design a check module to capture the changes between the original BPMN model and the updated BPMN model but also develop EPN model evolution operations to represent the BPMN model changes. The main objective of the EPN model evolution is to avoid model confusion and the loss of the token that is running.

The remainder of this paper is organized as follows: Section 2 analyses the main related work that involves the BPMN formalization. Section 3 describes the architecture of the BPMN formalization evolution. The metamodel definitions of the EPN and the EPN model evolution module are described in Section 4. Section 5 demonstrates the EPN evolution using a case study. Additionally, the consistency between the original BPMN model and the updated BPMN model is examined. Finally, we discuss the primary contributions, conclusions, and future work in Section 6.

2. Related Works

The BPMN 2.0 standard specification [1] describes the complete execution semantics in natural language, which can provide a clear and precise description of the element operations. In the model-driven development (MDD), the BPMN, which is a dominant notation OMG standard, is a popular business process modelling method for the computation-independent model (CIM) level [25–29]. Many researchers have formalized the BPMN model and analysed the semantics using Petri nets theory [4, 7, 8, 30–33], CSP [10, 34, 35], event-B [36], OCL specification [2, 37, 38], and BNF syntax [3, 11, 39], among others. Various formal tools based on the Petri nets are used to specify the BPMN semantics for soundness verification [40, 41], property verification [42], safety verification [43], validity verification [44], and semantics specification [45].

The verification techniques are important for business process modelling. Corradini et al. [3, 11, 39] exploited a textual representation to define the BPMN collaboration structure and formalized the well-structuredness, safeness, and soundness properties to check the correctness of BPMN collaborations. Zatout et al. [44] proposed formal orchestration process modelling with adaptation mechanisms and used the checking technique of the Petri nets to verify the validity of the generated models. For the large-scale business process, Dechsupa [33] provided a partitioning technique to verify the hierarchy and composition of the BPMN model. Mrasek et al. [45] presented an approach for the reduction of a process to enable its verification with model checking based on the colored Petri nets (CPN). Wong and Gibbons [34] used CSP to analyse the behavioural semantics of the BPMN model. By exploiting CSP's refinement orderings, relationships between the timed and the untimed models are clearly defined. To construct behavioural properties of the BPMN model [46], these authors translate behavioural properties into the CSP model for simple refinement verification. Mendoza et al. [10] also used CSP and time constraints to formalize the BPMN model. In particular, the time semantics helps us to understand the consistency of the activity and task information and verify the time information of an exception flow. To enforce model preciseness, Correia and Abreu [2] proposed supplementing the BPMN metamodel with well-formedness rules that are expressed as OCL invariants.

An overview of BPMN formalization studies is presented in Table 1. Although many studies focused on defining the semantics precisely for BPMN [2, 5–8, 10, 34], few methodologies address the dynamic evolution of the BPMN formalization. In terms of the evolution of the Petri nets model, the evolution operations have been discussed in much research [47–52]. Hu et al. [47] and Capra and Camilli [49] emphasized on the structural evolution (such as projection, link, difference, and substitution) of the Petri nets. Meantime, due to the adding, removing, or changing features, the platforms evolution of the rule-based Algebraic High-Level net transformations was proposed by [47], and this evolution can be used to obtain an evolution of scenarios. Zhu et al. [53] proposed an algebraic Petri nets (APN) slicing technique that optimizes the model checking of static or structurally evolving APN models. Krishna et al. [52] used process algebra to define the evolution such as conservative, inclusive and exclusive, selective, context-aware, and so on. However, this research focuses on the evolution of the Petri nets itself and generates a new Petri net via structural transformations again.

When formalizing a BPMN model into a Petri net, it is necessary to consider the change of the BPMN model. In this case, we cannot map the BPMN model to the formalized model each time when the BPMN model undergoes a small change because every mapping consumes time and incurs costs. On the other hand, the remapping formalization model would lose the token that is running. However, existing BPMN formalizations that are model-driven typically lack the support of bidirectional incremental synchronization between the source model and the target model [2]. Therefore, for in-place transformations, Kolahdouz-Rahimi et al. [54] proposed a systematic evaluation framework for comparing model transformation approaches. A relatively complete refactoring transformation approach was proposed by Einarsson [55], in which query/view/transformation operational mapping (QVTo) language (https://www.omg.org/spec/QVT/1.3/PDF) is used to

Table 1: Formal methods, objective, dynamic evolution, and implemented tools of BPMN formal studies.

Source	Formal methods	Objective	Dynamic evolution	Implemented tool
Wong and Gibbons [34]	CSP	Behavioural property verification	No	No
Dijkman and Van Gorp [7]	Petri nets mapping	Semantics specification	No	Yes
Van Gorp and Dijkman [8]	In-place transformation	Semantics specification	No	Yes
Correia and Abreu [2]	OCL invariants	Preciseness enforcement	No	No
Mendoza et al. [10]	CSP + Time	Soundness verification	No	No
Takemura et al. [30]	Petri nets mapping	Semantics specification	No	Yes
Bryans et al. [36]	Event-B	Formal verification	No	Yes
Yu et al. [31]	Object Petri nets mapping	Formal verification	No	Yes
Zhu et al. [53]	Colored Petri nets	Semantics specification	No	Yes
Dechsupa et al. [33]	Colored Petri nets	Hierarchical verification	No	Yes
Krishna [52]	Process algebra	Semantics specification	No	No
This paper	Petri nets mapping evolution	Semantics specification	Yes	Yes

refactor the merge and divide operations. For verifying workflow, service orchestration, and choreography engines, Dijkman and Van Gorp [7] used the GrGen framework to develop the BPMN graph rewrite tools. Domínguez et al. [23] proposed an evolution framework in which the XML schema and documents are incrementally updated according to the changes in the conceptual model (expressed as a UML class model). Therefore, we can use evolution methods to define an evolution transformation, which can dynamically evolve the BPMN formalization model according to changes to the BPMN model.

In this paper, we propose a BPMN formalization approach that can support dynamic evolution and use the Eclipse framework to develop its transformation plug-in. Compared with previous studies, our approach focuses on changing requirements and designs a check module that identifies the elements to be modified by comparing the original BPMN model and the updated BPMN model. The main advantage of our approach is that the Petri nets formalization model changes dynamically according to the changes in the BPMN model.

3. BPMN Formalization Evolution Architecture

The formalization evolution function is to maintain the consistency between the BPMN and EPN models when changes occur in a BPMN model. In this way, the EPN model is incrementally updated to avoid repeated mapping of the BPMN model to the EPN model. In addition, this incremental transformation can protect tokens when some place nodes are deleted. The evolution process is described in our transformation architecture (Figure 1), where the modified data of the BPMN model are written into a file and the EPN model is evolved based on this file. The main advantage of this process is that it can support the dynamic evolution of the EPN model. This evolution will involve four models: the BPMN model, the updated BPMN model, the EPN model, and the updated EPN model. The updated BPMN model is based on the BPMN model; the EPN model is mapped by the BPMN model via the BPMN2EPN plug-in; and the updated EPN model is generated by executing the EPN evolution plug-in. Thus, in this approach, once the source model has been changed, the evolution plug-in can refine the target model correspondingly through the QVTo engine based on the model modification data.

In this process, we assume that, at each time, evolution transformations are dispatched from only one designer; hence, we do not consider concurrency issues. This formalization approach differs from that of other researchers because our approach focuses on the dynamic evolution in the formalization transformation process. Therefore, our approach can facilitate the adjustment of the system requirements by the business analyst. As the BPMN2EPN plug-in (Figure 1) has been developed in our previous research [25], this paper will just focus on the EPN evolution.

4. BPMN Formalization Evolution

4.1. EPN Model Specification. A condition-event net (CEN), which is a basic Petri net, is composed of a condition (state), an event (transition), and a link (arrow) and is simple and easy to understand in contrast to other Petri nets that are used for process modelling [25, 56]. However, it is difficult to represent the gateway of the BPMN model. Thus, the CEN has been extended and a mapping between the BPMN model and the CEN model has been designed in our previous research [25]. Here, the EPN model is defined formally as follows.

Definition 1. An extended CEN can be designed as a 7-tuple, namely, EPN = $\langle P, \mathrm{ST}, \mathrm{BT}, \mathrm{IA}, \mathrm{OA}, M_0, \mathrm{SN}\rangle$, where there is the following:

(i) P denotes the control flow and the message flow between the activity nodes and represents the execution order of the activity nodes. P consists of a set of finite conditions and is expressed as $P = \{p_1, p_2, p_3, \ldots, p_n\}$

(ii) $\mathrm{ST} = \{\mathrm{st}_1, \mathrm{st}_2, \mathrm{st}_3, \ldots, \mathrm{st}_n\}$ denotes a set of finite *silent transition* nodes that represent the business process start/end points and specify the direction of the execution path

(iii) $\mathrm{BT} = \{\mathrm{bt}_1, \mathrm{bt}_2, \mathrm{bt}_3, \ldots, \mathrm{bt}_n\}$ denotes a set of finite *behaviour transition* nodes that represent the task or event in the BPMN model

(iv) $\mathrm{IA} \subseteq (P \times \mathrm{ST}) \cup (P \times \mathrm{BT})$ denotes a set of *input arcs*

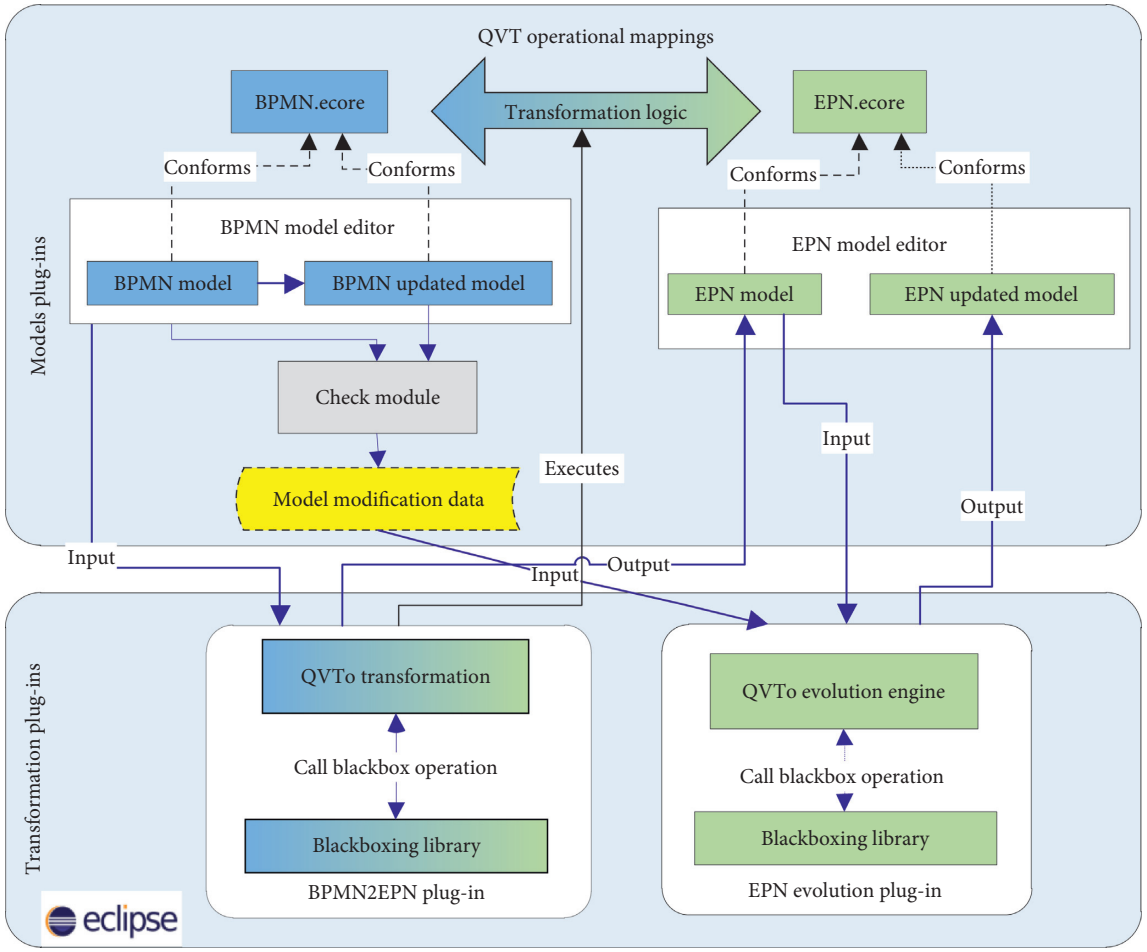

Figure 1: Architecture of the evolution process of the extended Petri nets model.

(v) $OA \subseteq (ST \times P) \cup (BT \times P)$ denotes a set of *output arcs*

(vi) M_0 denotes the initial representation of the EPN

(vii) $SN = \{sn_1, sn_2, sn_3, \ldots, sn_n,\}$ denotes a set of finite sub-EPN net, where $sn_i = \langle P, ST, BT, IA, OA, M_0, SN\rangle_{sub}$ denotes a sub-EPN net that represents a standalone process in the BPMN model

In the EPN model that is specified above, P (condition) is represented as a circuit. ST (labelled by the solid rectangle) and BT (labelled by the hollow rectangle) are extensions of the event element.

Definition 2. For a $EPN = \langle P, ST, BT, IA, OA, M_0, SN\rangle$, $\forall st_1 \in ST$, satisfies the following:

(1) If $\exists p_1 \in P$ and p_1 includes M_0, $p_1 \times st_1 \neq O$, $st_1 \times p_1 = \varnothing$. st_1 is called the start operation node of the Petri nets; it represents the starting point of the Petri nets.

(2) If $\exists p_1 \in P$ and p_1 includes M_0, $st_1 \times p_1 \neq O$, $p_1 \times st_1 = \varnothing$. st_1 is called the end operation node of the Petri nets; it represents the ending point of the Petri nets.

(3) If st_1 is both the start operation and the end operation, st_1 is called the zero operation.

Definition 3. For a $EPN = \langle P, ST, BT, IA, OA, M_0, SN\rangle$, $\exists \forall bt_1, bt_2 \in BT$, let ρ: $bt_1 \longrightarrow bt_2$ denote that bt_1 must be implemented prior to bt_2, where " $\longrightarrow$ " expresses an execution order between bt_1 and bt_2, and $\rho$is a place node that links the two behaviour transition nodes. Thus, we define two functions, namely, source and target, which represent the predecessor node and the successor node, respectively, of the place node, such that $source(\rho) = st_1$ and $target(\rho) = st_2$.

Definition 4. For a $EPN = \langle P, ST, BT, IA, OA, M_0, SN\rangle$, $\forall st_1, st_2, st_3, \ldots, st_n \in ST$, $\forall bt_1, bt_2, bt_3, \ldots, bt_n \in BT$, $\forall p_1 \in P$, should satisfy the following conditions:

(1) According to Definition 2, st_1 is a start operation node and st_n is an end operation node.

(2) Suppose $P(bt_1 \times p_1) = P(p_1 \times bt_2)$, and an execution order has been defined between bt_1 and bt_2 based on Definition 3. A process sequence, which is denoted as $Pi = \{st_1 \longrightarrow bt_1 \longrightarrow bt_2 \longrightarrow bt_3 \longrightarrow \cdots \longrightarrow st_n\}$, consists of the start execution order, the end

execution order, and multiple execution orders ($bt_1 \longrightarrow bt_2, bt_2 \longrightarrow bt_3, bt_3 \longrightarrow bt_4, \ldots$). BT is a set of tasks of the business system.

Based on the above definitions, the main properties of the EPN model transition elements are as follows:

(1) A silent transition node that links a place node: this place node has only one output arc, which represents a Petri net starting point.

(2) A silent transition node that links a place node: this place node has only one input arc, which represents a Petri net ending point.

(3) A silent transition node that links more than one place node represents the path decision in a Petri net.

(4) A behaviour transition node that links more than one place node represents the behaviour operation in a Petri net.

(5) A subpage node that links more than one place node is a sub-Petri net and can be further specified.

4.2. Check Module. Based on the evolution process architecture, the proposed check algorithm is presented in pseudocode in Algorithm 1. The algorithm takes an original BPMN file and a modified BPMN file as input and outputs a list of modification data for the BPMN model. First, the algorithm creates a list, namely, *modification_datas*, of objects that are declared in the BPMN meta-model, which include *M_Task*, *M_SequenceFlow*, *M_MessageFlow*, *M_Subprocess*, and *M_IntermediateThrowEvent.* For example, *M_Task* is a list that stores the modified task elements, which are identified by comparing the *BPMN_old* model with the *BPMN_new* model (line 2). In addition, we define two lists, namely, *O_Task* and *N_Task*, of task elements that have been declared in the *BPMN_old* model and the *BPMN_new* model (lines 4-5). Then, for each task in *O_Task* and *N_Task*, the algorithm executes the comparison operation: if a task exists in the *BPMN_old* model but not in the *BPMN_new* model, then this task is stored in *M_Task* and its operation label is set as "*delete*" (lines 9–17); if a task exists in the *BPMN_new* model but not in the *BPMN_old* model, this task is stored in *M_Task* and its operation label is set as "*insert*" (lines 21–23); and if a task exists in both the *BPMN_old* and *BPMN_new* models but the task names differ, then, the name of this task is stored in *M_Task and* its operation label is set as "*modify*" (lines 24–29).

Next, the remaining elements in *BPMN_old* and *BPMN_new* are compared in the omitted part of the check algorithm. The output file is important because the evolution operation that is executed by the evolution component depends on the results in the output file.

4.3. EPN Model Evolution. According to the result of the check algorithm, the modification of the BPMN model involves the task, gateway, intermediate event, end event, and flow. Therefore, we defined a set of evolution operations of the EPN model as listed in Table 2. For example, if a task is deleted, created, or modified in the BPMN model, then the EPN model should execute an add, delete, or modify operation correspondingly.

According to Table 2, the EPN model evolution has three types of elements: (1) behaviour transitions, (2) silent transitions, and (3) places. In the following, we will explain the main evolution transformation in detail.

4.3.1. Behaviour Transition Node Evolution

(1) Delete Action. The delete behaviour transition action removes a behaviour transition node in the EPN model. For example, if there is a task that validates the user name, then there is another task that validates the user password. The actions of validating the user name and validating the user password may be merged into a single task (validating the user information) of the client, as illustrated in Figure 2. In this scenario, based on the check algorithm, two *behaviour transition* nodes that are linked by a *place* will be merged into one, namely, a behaviour transition node must be deleted.

Two delete scenarios must be distinguished: (1) if the token of a deleted behaviour transition node has just arrived at the input place, which should be linked to the next behaviour transition node, then the token should be transferred to the output place to the next behaviour transition to ensure that the token exists; (2) if the token of a deleted behaviour transition node has not yet reached the input place or has already reached the output place, then this behaviour transition node should be deleted directly.

Therefore, the steps in the evolution are as follows:

(1) Design two queries that obtain the id numbers of the behaviour transition node and place node to be deleted from the blackboxing library.

(2) Design a query that obtains the id number of the behaviour transition node to be modified from the blackboxing library.

(3) Design a query that identifies the token's position. If the input place of the deleted node has a token, proceed to step 4. Otherwise, proceed directly to step 5.

(4) Modify the connection of the output arc linked input place to protect the token.

(5) Delete the behaviour transition node.

(6) Delete the output place node.

(7) Delete the output arc that connects the behaviour transition node to be deleted.

(8) Modify the behaviour transition node.

(2) Insert Action. The insert behaviour transition action adds a new behaviour transition node into the EPN model. As illustrated in Figure 3, a task (add to cart) is added in the source BPMN model; thus, in this scenario, the add behaviour transition action should be executed in the EPN model.

Therefore, the steps in the evolution are as follows:

```
Input: BPMN_old: original BPMN XML file BPMN_new: modified BPMN XML file
Output: modification_datas: modification data file
(1) Collections:
(2)   M_Task; //object list of the modification of the task object elements
(3)   M_SequenceFlow; //object list of the modification of the sequence flow object elements
(4)   O_Task; //object list of the task declared in BPMN_old
(5)   N_Task; //object list of the task declared in BPMN_new
(6)   O_SeqenceFlow; //object list of the sequence flow declared in BPMN_old
(7)   N_SeqenceFlow; //object list of the sequence flow declared in BPMN_new
(8)   . . . . . .
(9) for task t_o in O_Task do
(10)   for task t_n in N_Task do
(11)     if t_o.id! = t_n.id then
(12)       t_o.mark = "delete";
(13)       M_Task.add(t_o);
(14)       break;
(15)     end
(16)   end
(17) end
(18) for task t_n in N_Task do
(19)   for task t_o in O_Task do
(20)     if t_o.id! = t_n.id then
(21)       t_o.mark = "insert";
(22)       M_Task.add(t_n);
(23)       break;
(24)     else
(25)       if t_n.name! = t_o.name then
(26)         t_o.mark = "modify";
(27)         M_Task.add(t_n);
(28)         break;
(29)       end
(30)     end
(31)   end
(32) end
(33) . . . . . .
```

Algorithm 1: BPMN model check algorithm.

Table 2: Evolution operations of the EPN model according to the BPMN model.

BPMN model element (source)	EPN model element (target)	Evolution operation
Task	BehaviourTransition	Insert, delete, and modify
Gateway	SilentTransition	Insert, delete, and modify
IntermediateThrowEvent	BehaviourTransition	Insert, delete, and modify
IntermediateCatchEvent	BehaviourTransition	Insert, delete, and modify
BoundaryEvent	BehaviourTransition	Insert, delete, and modify
MessageFlow	Place	Insert, delete, and modify
SequenceFlow	Place	Insert, delete, and modify
EndEvent	SilentTransition	Insert and delete

(1) Design a query that obtains the id number and name of the behaviour transition node to be added from the blackboxing library.

(2) Add the behaviour transition node.

(3) Add a place node and corresponding input and output arcs.

(4) Set the sources and targets of the new input and output arcs.

(5) Modify the output arc that connects the behaviour transition node to be added.

(6) Modify the input arc that connects the behaviour transition node to be added.

4.3.2. Silent Transition Node Evolution

(1) Delete Action. The delete silent transition action removes a silent transition node in the EPN model; hence, an end event or a gateway is deleted from the BPMN model. As illustrated in Figure 4, the customer requests to cancel her order after she has placed it. The web shopping process must

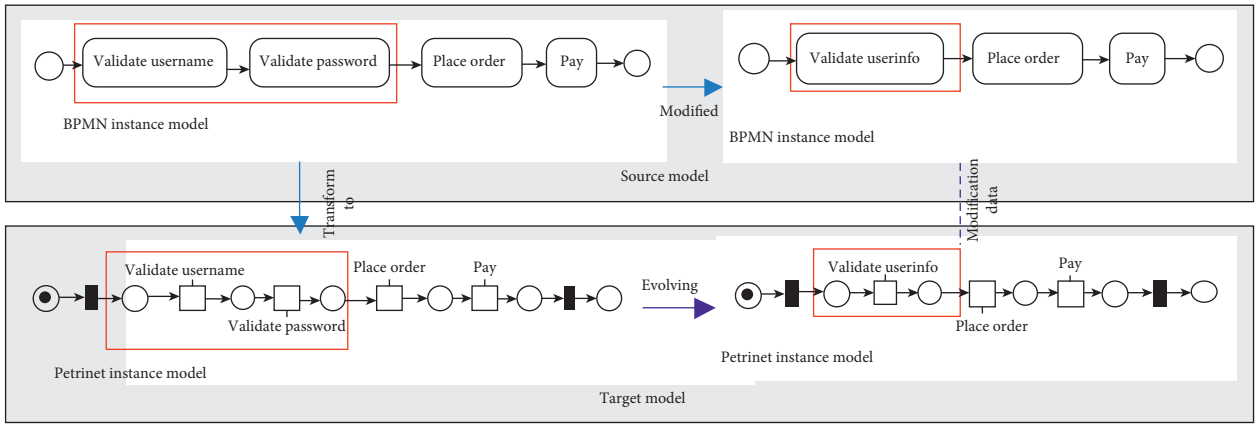

Figure 2: Merge action of the behaviour transition node evolution.

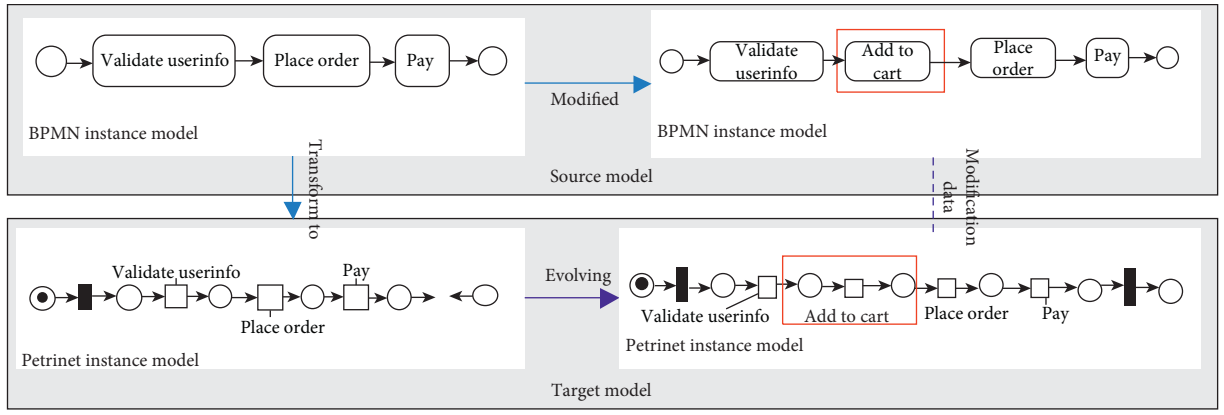

Figure 3: Insert action of the behaviour transition node evolution.

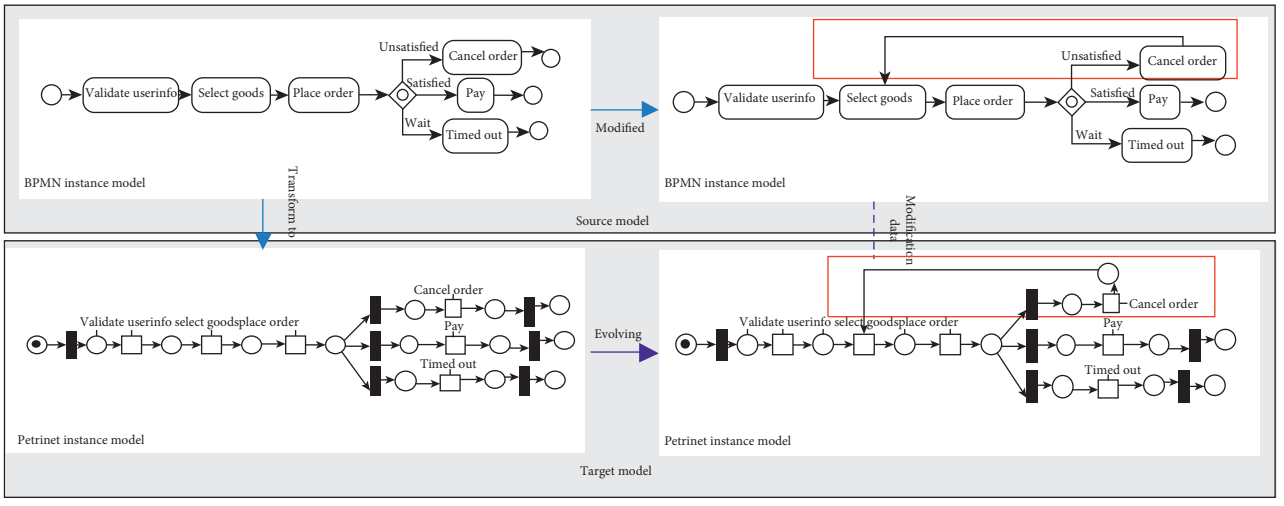

Figure 4: Delete action of the silent transition node evolution.

be terminated in an old process. However, in the new business process, the customer can reselect goods after the order has been cancelled.

In this case, an end event node has been deleted and the target node of a sequence flow is modified in the source BPMN model. Therefore, a silent transition and the

postcondition place node that is connected to this silent transition should be deleted in the EPN model.

Thus, the steps in the evolution are as follows:

(1) Design a query that obtains the id number of the silent transition node to be deleted from the blackboxing library.

(2) Design a query that detects the token's position. If the input place of the deleted node has a token and the deleted node is not an end node, proceed to step 3. Otherwise, proceed directly to step 4.

(3) Modify the connection of the output arc linked input place to protect the token.

(4) Delete the silent transition node.

(5) Find the place node to be deleted when the source of an arc is the silent transition node to be deleted.

(6) Delete the place node.

(7) Delete the output arc that connects the silent transition node to be deleted.

(8) Modify the input arc that connects the silent transition node to be deleted.

(2) Insert Action. The insert silent transition action adds a new silent transition node into the EPN model; hence, a new gateway or a new end event is added in the BPMN model. An example is presented in Figure 5. The typical process of web shopping online is to select goods, place an order, and provide payment. However, if the user decides not to buy the selected goods, she can cancel the order and the process should be terminated. Alternatively, if she is indecisive regarding this order and does not perform the next operation, the system will take an action (time out) to address this situation, and the process should be terminated.

In this case, an inclusive gateway and the corresponding actions are added in the source BPMN model. Therefore, in the evolution of the EPN model, a silent transition element and its related behaviour transition element will be added based on the check algorithm.

Thus, the steps in the evolution are as follows:

(1) Design a query that obtains the number of gateway paths and clones the original corresponding silent transition.

(2) Specify the name of the new silent transition.

(3) Clone the original place.

(4) Clone the original arc.

(5) Set the source of the arc to the clone.

(6) Set the target of the arc to the clone.

4.3.3. Place Node Evolution. The place node in the EPN is related to the sequence flow or message flow element in the BPMN, and the evolution action includes "*add*," "*delete*," and "*modify*" actions. For the add action, the steps in the evolution are as follows:

(1) Design a query that obtains the id of the place node from the blackboxing library, and add the corresponding place node

(2) Add an input arc for the place node, and set the source node and target node of this input arc

(3) Add an output arc for the place node, and set the source node and target node of this output arc

For the delete action, the steps in the evolution are as follows:

(1) Design a query that obtains the id of the place node to be deleted from the blackboxing library

(2) Delete the place node

(3) Delete the input arc and output arc that connect the place node to be deleted

For the modified action, the steps in the evolution are as follows:

(1) Design a query that obtains the id of the place node to be modified from the blackboxing library

(2) Modify the output arc that connects the place node to be modified

(3) Set the target of the arc to be modified

The model evolution proposed in this section mainly considers model element modifications (such as the addition of a transition node, deletion of a transition node, and addition of a place node), in contrast to Hu's [47] proposed evolution, which focuses on using one or more Petri nets models to generate code via structural transformations (such as projection, link, difference, and substitution).

4.4. Evolution Implementation. The QVT language, which is an open model transformation language that is based on the OMG standard, includes the QVT relations, the QVT core, and the QVT operational mappings (QVTo). The QVT language offers a blackbox implementation mechanism that is similar to the JAVA virtual machine, which enables programmers to perform more complex model transformations. Thus, based on the QVT language, the EPN model evolution in the Eclipse framework (https://projects.eclipse.org/projects/modeling.emf) is illustrated in Figure 6. The evolution component includes three subcomponents: the *petrinet refactoring.core.qvt.transformation* component contains a QVTo transformation; the *petrinet refactoring.core.qvt.libraries* contains a JAVA blackboxing library, which can be made available by other plug-ins; and the check module executes the check algorithm that was proposed in Section 4.2 and can identify mismatch data between the original BPMN model and the modified BPMN model. These mismatch data are analysed by the JAVA blackboxing library and the blackboxing operation delivers these mismatch data to the QVTo transformation. The QVTo transformation obtains the parameters and the EPN model for executing the evolution transformations (e.g., *AddBehaviourTransition, DeleteBehaviourTransition, and AddSilentTransition*) and overwriting the original model.

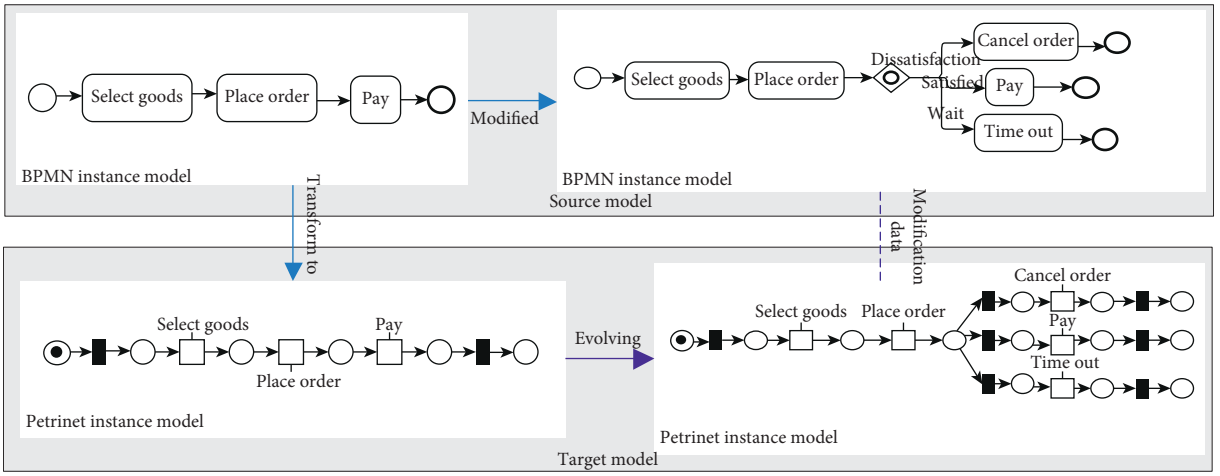

Figure 5: Add action of the silent transition node evolution.

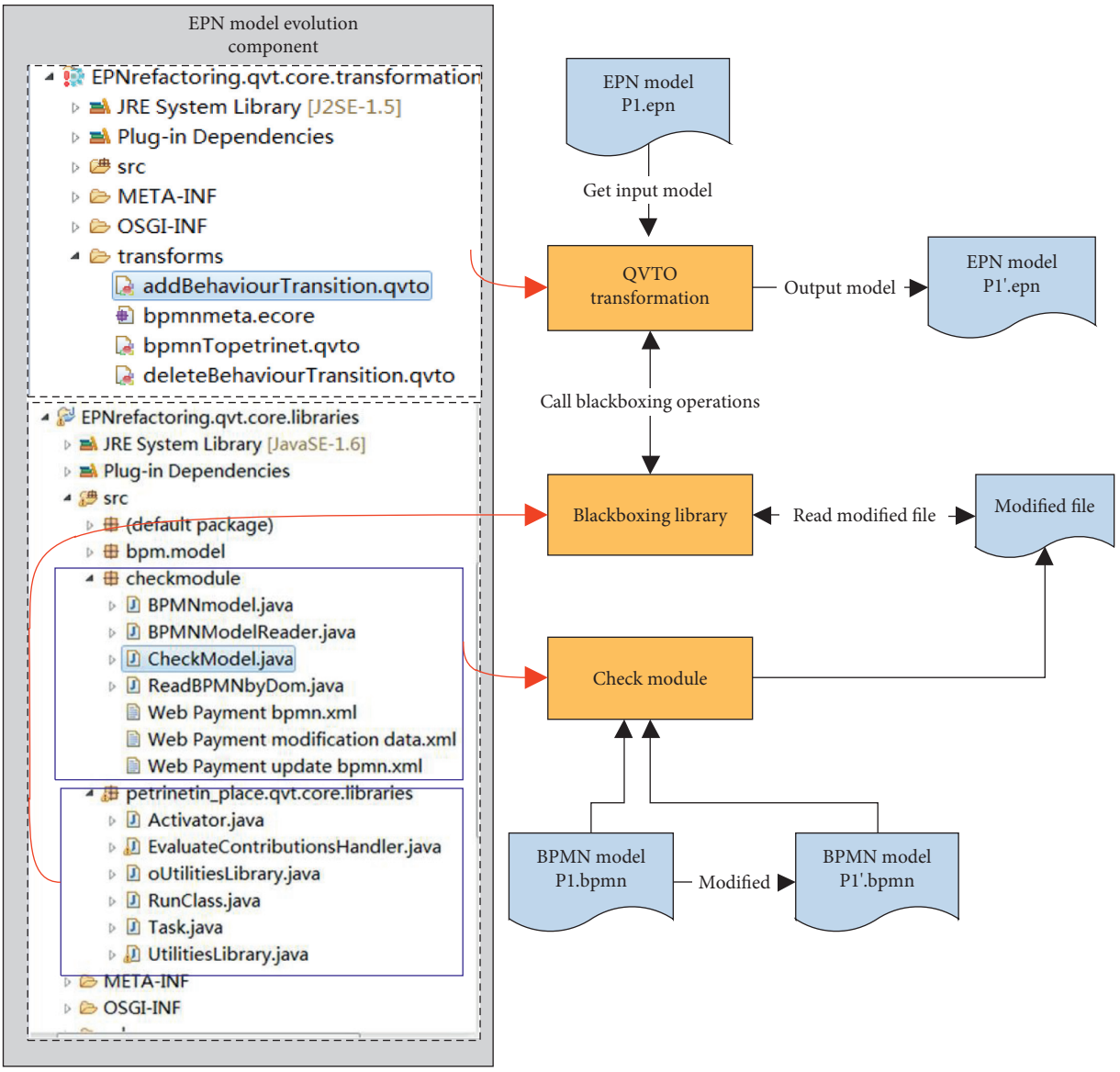

Figure 6: Workflow of the evolution transformation (overview of the evolution component prototype).

Based on the step in which the evolution rule is implemented, a set of actions (Table 3) is designed for implementing the EPN model evolution. These evolution actions can be activated when the execution result of the check is not null; for instance, the merge action of the behaviour transition is activated when the outgoing flow of a task is removed and the target node of the incoming flow is modified in the modification data file. To perform the merge action, a set of operations (e.g., *query getObjectToDelete(), mapping inout deleteBehaviourTransition(),* and *mapping inout deletePlace()*) should be executed. The insert action of the behaviour transition is activated when a new task is added and the target of a sequence flow is modified in the modification data file. Thus, a set of operations (including *query getObjectToAdd()*, *mapping inout addBehaviourTransition()*, *mapping inout addPlace()*, *map setTarget()*, and *mapping inout modifyArc()*) should be executed for this insert action. The add actions of the silent transition are activated when there are new gateway keywords or end events in the modification data file.

To illustrate this evolution process, a partial view of the EPN model in-place transformation is shown in Figure 7. *DeleteBehaviourTransition.qvto* conducts the delete evolution of the behaviour transition node when a task node has been deleted in the BPMN model. Thus, the *inout* mapping initially defines a guard that determines whether the transition node must be deleted by using the query *getObjectToDelete()* (line 91). Query *getObjectToDelete()* uses the blackboxing method *isDeleteObject()* to read the modification data file and obtain the delete node. Then, the delete transition node operation is executed (line 96). The output arc and the place that is linked by this transition node are also deleted (lines 98–103). Finally, the input arc of the transition node is modified (lines 105–108).

5. Case Study

5.1. Scenario. The following scenario (http://www.w3.org/TR/web-payments-use-cases/) is provided by the W3C organization. We consider a web payment scenario, which is organized into four phases: negotiation of payment terms, negotiation of payment instruments, payment processing, and delivery of product/receipt and refunds.

Negotiation of payment terms phase: a customer browses the items on the *online shopping site*. She selects the goods and puts them into the shopping cart prior to logging into her account. We define this phase as a ***Search Service.***

Negotiation of payment instruments phase: once the customer has decided to buy these goods, the shopping site accepts a payment, which can be in the form of a credit card, debit card, secured money order, Google wallet, or ApplePay. Next, the customer selects a card that is highlighted by default because she used it for a previous purchase and the shopping site asks her to input her zip code and the verification code before the purchase is completed. We define this phase as a *Payment Instruments Service***.**

Payment processing phase: the website receives a message from the customer's device that authorizes the payment. The *shopping site* submits a message to the financial company that requests a proof of hold for the funds. Then, the customer and the shopping site receive proof of payment from the financial company, and the funds are immediately deducted from the customer's line of credit and transferred to the website's bank account. We define this phase as a *Payment Service*.

Finally, the *shopping site* sends the customer a digital receipt, the website's shipping department packs and delivers the purchased items to the shipper, and the shipper sends them to the customer. We define this phase as a *Delivery Service*.

5.2. Models. Figure 8 illustrates the business process model of the web payment case, which consists of a *Search Service*, *Payment Instruments Service*, *Payment Service*, and *Delivery Service*. These services are subprocesses in the BPMN model. Here, we use our previous research to map the BPMN model onto the EPN model as illustrated in Figure 9. The BPMN model can be verified by the EPN model using the verification technique that is proposed in [20].

To promote the shopping service, the online shopping site decides to offer a cash-on-delivery service. So, the cash-on-delivery service should be added in the original BPMN model, as shown in Figure 10. Three new tasks (*Select Payment type*, *Select Cash on Delivery*, and *Select Online Pay*), one exclusive gateway, and one control flows should be added and message flows should be modified or deleted. The consistency of the new business process model is important for the online site. Thus, the check module is executed to record the modification data between the original BPMN model and the updated BPMN model. Then, the EPN model evolution plugin is executed to avoid repeated mapping. Figure 11 shows the result of the EPN evolution action. Using the Petri nets standard verification techniques such as deadlock and livelock analysis [25, 26], the updated BPMN model is correct.

In contrast to other BPMN formalization approaches, our proposed approach can ensure the consistency of the business process when its requirements change. More importantly, our proposal can effectively protect the token from being lost during the evolution process. We can use the formalized model to verify the consistency between the original BPMN model and the updated BPMN model (Section 5.3). This model is essential for protecting the business process model changes that do not affect the entire business system.

5.3. Evaluation. Firstly, we can use the ePNK tool to verify the correctness properties of the evolved EPN model (Figure 11), such as no deadlocks, no closed loops, and no isolated nodes in the evolved EPN model; then, we use the formalized model to verify the consistency between the original BPMN mode and the updated BPMN model and to analyse the behavioural semantic compatibility of these two Petri nets models. Thus, the original BPMN model and the updated BPMN model are defined as semantically compatible if they perform the same business behaviour. Let the original BPMN model, which is denoted as BP_{old}, be

Table 3: Partial list of the EPN model evolution action.

	Evolution action	DeleteBehaviourTransition	InsertBehaviourTransition	DeleteSilentTransition	InsertSilentTransition	InsertPlaceTransition
Query operation	query getObjectToDelete()	√		√		
	query getFlowToDelete()	√		√		
	query getObjectToModify()	√				
	query getToken()	√		√		
	query getObjectToAdd()		√		√	√
	query getFlowToAdd()		√		√	√
Mapping inout operation	mapping inout deleteBehaviourTransition()	√				
	mapping inout deleteArc()	√		√		
	mapping inout deletePlace()	√		√		
	mapping inout modifyArc()	√	√	√		
	mapping inout modifyBehaviourTransition()	√				
	mapping inout addBehaviourTransition()		√			
	mapping inout addPlace()		√		√	√
	mapping inout addArc()		√		√	√
	mapping inout addSilentTransition()				√	
	mapping inout deleteSilentTransition()			√		
Map operation	map setSource()	√	√		√	√
	map setTarget()	√	√	√	√	√

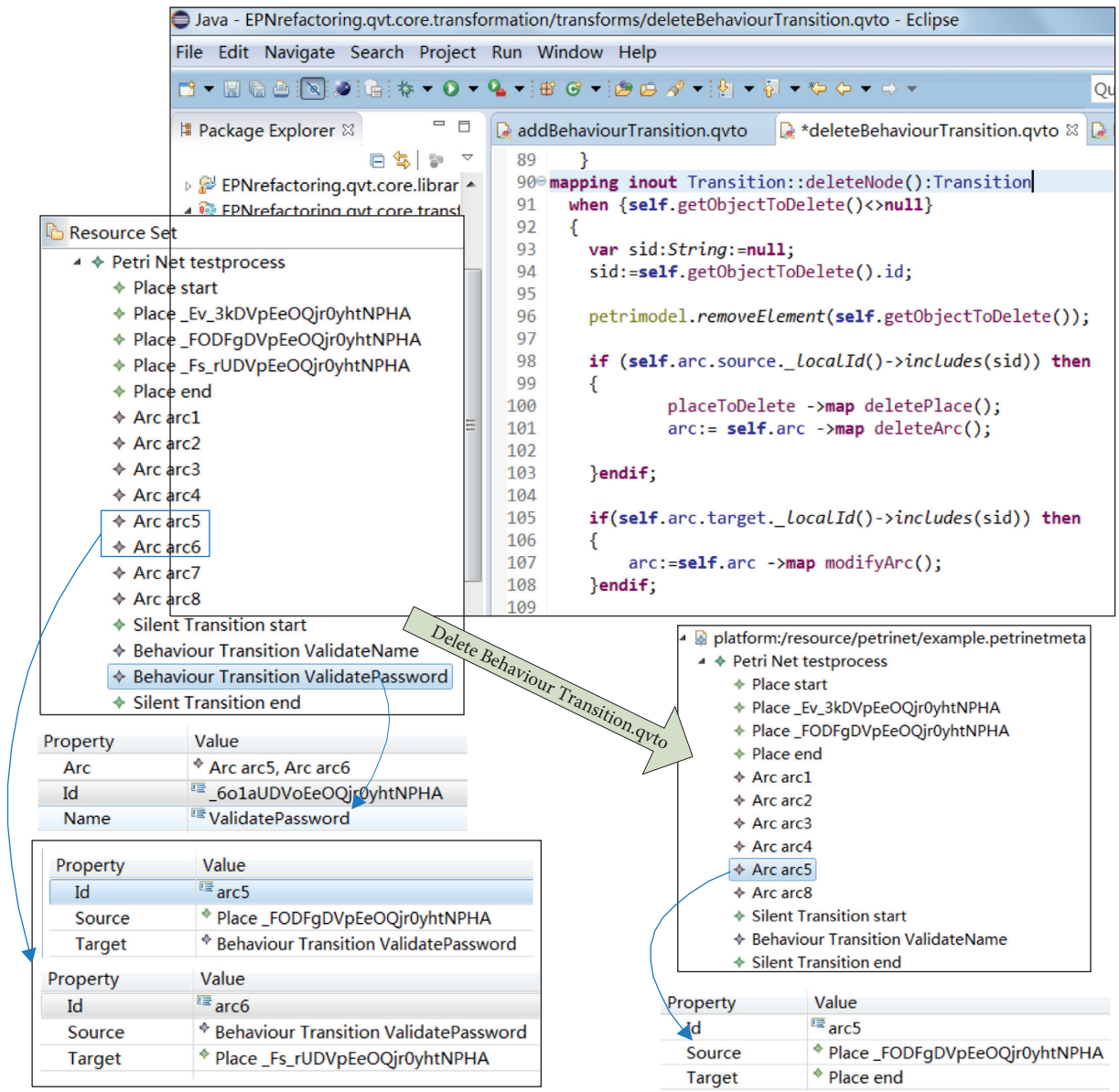

Figure 7: Partial view of the EPN model evolution transformation and DeleteBehaviourTransition QVT code excerpt.

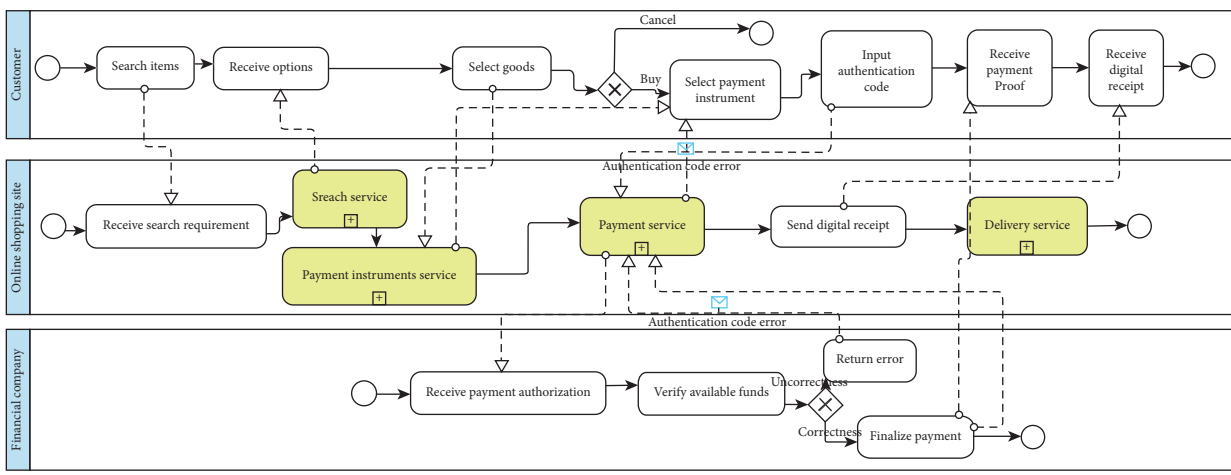

Figure 8: BPMN model of the web payment system.

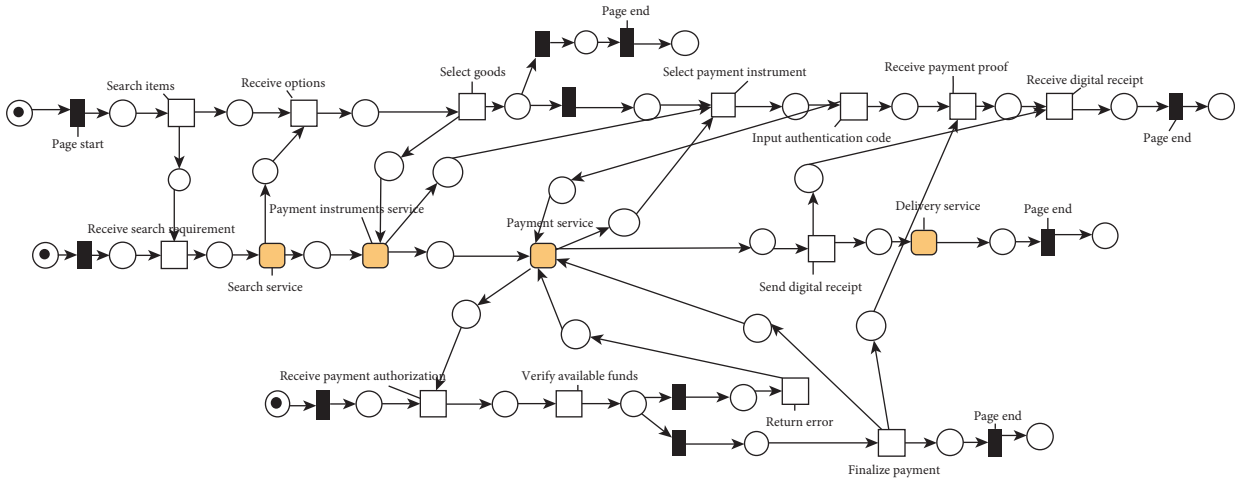

Figure 9: EPN model that is based on the BPMN model of the web payment.

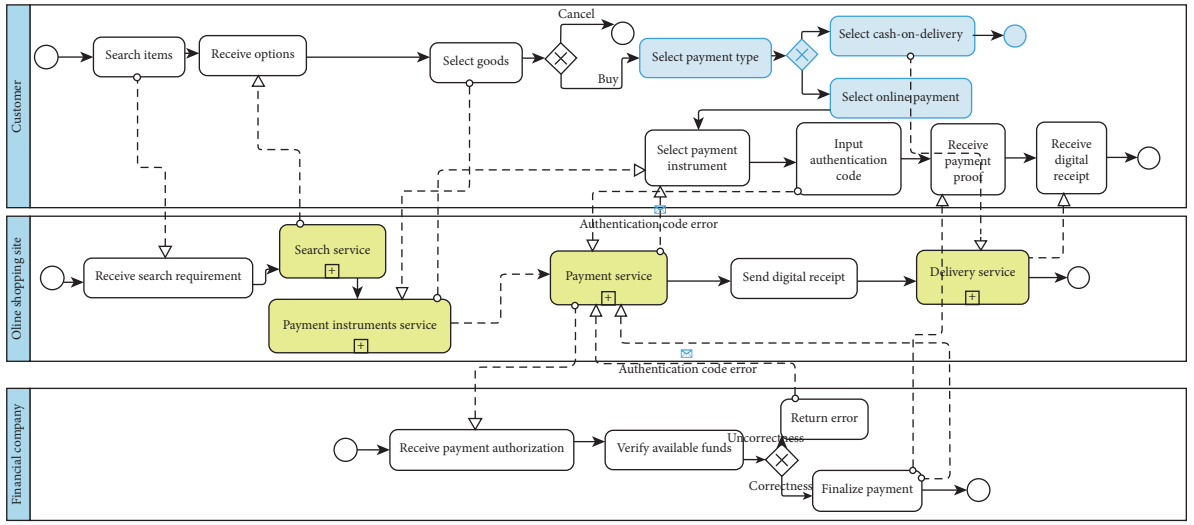

Figure 10: Updated BPMN model of the web payment system.

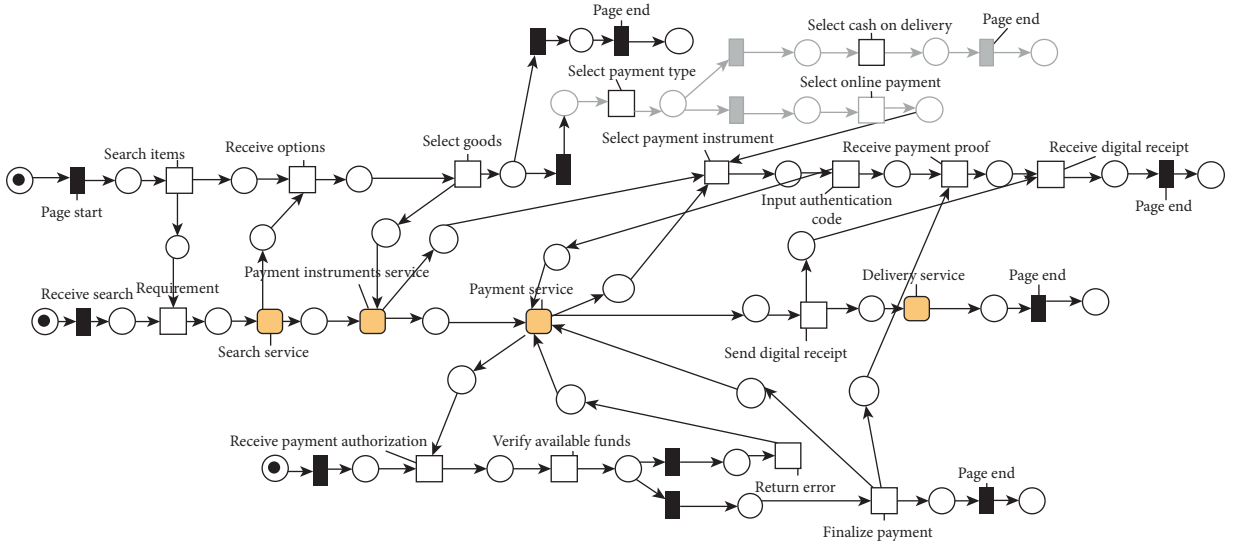

Figure 11: EPN model that has been refactored by the updated BPMN model of the web payment system.

formalized by the Petri nets model, which is denoted as PN_{old}, and let the formalized model of the updated BPMN model, which is denoted as BP_{new}, be formalized by the Petri nets model, which is denoted as PN_{new} and evolved from PN_{old}. We define a Petri nets-type language, namely, $C(PN_\alpha|\sum c)$, such that the business process sequence α is generated over $\sum c$, where $\sum c$ denotes the common alphabet in the business process Petri nets model; $\sum c = \sum c_{subpage} \cup \sum c_{b\_transition}$, $\sum c_{subpage}$ denotes the common alphabet of the subpages in the PN_{old} and PN_{new} Petri nets models; and $\sum c_{subpage} = \sum PN_{old\cdot subpage} \cap \sum PN_{new\cdot subpage}$, and $\sum c_{b\_transition}$ denotes the common alphabet of the behaviour transitions in the PN_{old} and the PN_{new} Petri nets models and $\sum c_{b\_transition} = \sum PN_{old\cdot b\_transition} \cap \sum PN_{new\cdot b\_transition}$.

To verify the consistency between BP_{old} and PN_{new}, three scenarios are defined:

(1) The original BPMN model and the updated BPMN model are completely semantically compatible, which is denoted as $BP_{old} \lhd\rhd BP_{new}$, iff: $C(PN_{old}|\sum c) = C(PN_{new}|\sum c)$. This scenario demonstrates that a few atomic activity names are modified in the original BPMN model. The number of atomic activities, the source and target nodes of the control flow, and the source and target nodes of the message flow in the updated BPMN model are the same as in the original BPMN model.

(2) The original BPMN model and the updated BPMN model are partially semantically compatible, which is denoted as $BP_{old} \rhd BP_{new}$, iff: $C(PN_{old}|\sum c) \cap C(PN_{new}|\sum c) \neq \varnothing$. This scenario demonstrates that atomic activities, control flows, and message flows are added or deleted in the original BPMN model, but at least one business process sequence exists in both the BP_{old} model and the BP_{new} model.

(3) The original BPMN model and the updated BPMN model are weakly semantically compatible, which is denoted as $BP_{old} \rhd\lhd BP_{new}$, iff: $\sum c \neq \varnothing$. This scenario demonstrates that many atomic activities, control flows, and message flows are modified, added, or deleted in the original BPMN model. No business process sequence exists in both the original BPMN model and the updated BPMN model. However, various atomic activities, control flows, or message flows exist in both the BP_{old} model and the BP_{new} model.

To assess the definition of semantic compatibility between the BP_{old} model and the BP_{new} model, we must consider the same behaviour in each.

$\sum c_{subpage}$ = {Search Service, Payment Instruments Service, Payment Service, Delivery Service}.

$\sum c_{b\_transition}$ = *{Search Items, Receive Search Requirement, Receive Options, Select Goods, Select Payment Instrument, Input Authentication Code, Receive Payment Authorization, Verify Available Funds, Return Error, Finalize Payment, Send Digital Receipt, Receive Payment Proof, Receive Digital Receipt}*.

$\sum c$ = {*Search Service, Payment Instruments Service, Payment Service, Delivery Service, Search Items, Receive Search Requirement, Receive Options, Select Goods, Select Payment Instrument, Input Authentication Code, Receive Payment Authorization, Verify Available Funds, Return Error, Finalize Payment, Send Digital Receipt, Receive Payment Proof, Receive Digital Receipt*}.

Next, each of the business processes is projected on this common alphabet:

$C(PN_{old}|\sum c)$ = {*Receive Search Requirement* ⟶ *Search Service* ⟶ *Payment Instruments Service* ⟶ *Payment Service* ⟶ *Send Digital Receipt* ⟶ *Delivery Goods, Receive Payment Authorization* ⟶ *Verify Available Funds* ⟶ *Finalize Payment, Search Items* ⟶ *Receive Search Requirement* ⟶ *Search Service* ⟶ *Receive Options* ⟶ *Select Goods* ⟶ *Payment Instruments Service* ⟶ *Select Payment Instrument*· · · · ·}.

$C(PN_{new}|\sum c)$ = {*Receive Search Requirement* ⟶ *Search Service* ⟶ *Payment Instruments Service* ⟶ *Payment Service* ⟶ *Send Digital Receipt* ⟶ *Delivery Goods, Receive Payment Authorization* ⟶ *Verify Available Funds* ⟶ *Finalize Payment, Search Items* ⟶ *Receive Search Requirement* ⟶ *Search Service* ⟶ *Payment Instruments Service* ⟶ *Select Payment Instrument*· · · · · ·}.

Finally, the projected process sequences are compared:

$$C\left(PN_{old}|\sum c\right) \cap C\left(PN_{new}|\sum c\right) \neq \varnothing,\ PN_{old} \rhd PN \Rightarrow BP_{old} \rhd BP_{new}. \tag{1}$$

According to this semantic compatibility analysis, the original BPMN model and the updated BPMN model are partially semantically compatible; namely, tasks, gateways, and control flows have been added in the updated BPMN model (Figure 10, rounded rectangle filled with blue), and the corresponding silent transitions, behaviour transitions, places, and arcs must be added into the EPN evolution model (Figure 11, grey icon). The internal business process of the online shopping site and the financial company in the original BPMN model are the same as those in the updated BPMN model. Thus, if the BPMN model is modified, we can verify the consistency between the original BPMN model and the updated BPMN model using the refactoring formalization model.

6. Conclusions and Ongoing Work

Currently, the development of methodologies that enable the dynamic evolution of the BPMN formalization is one of the main challenges in BPMN model formalization. In the formalization approach that is proposed in this paper, we have designed an evolution module that supports the automatic update of the formalized model and protects the token from being lost when the original BPMN model has been changed.

In this study, we propose a model transformation process that focuses on a BPMN model formalization that supports dynamic evolution. The evolution operation is the core element of the formal model transformation approach that we have proposed. We have demonstrated how we address dynamic evolution using in-place transformations in a case study in which we apply the Eclipse tool. Overall, our proposed approach can completely formalize the BPMN model elements while supporting in-place transformations, which can reduce the transformation costs and maintain the consistency between the BPMN model and the EPN model (the BPMN formalization model).

However, the effectiveness of the check module affects the evolution cost directly: if the BPMN model becomes more complex, the effectiveness of the check module might decrease. Thus, increasing the execution speed of the check algorithm is necessary, which can help to reduce the evolution costs. In addition, our proposed approach constrains the evolution operation without further refinement. Thus, the evolution rules, involving artifacts, conversations, resources, and exception event, need to be formulated in our future work.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (61902141); the MOE (Ministry of Education in China) Youth Foundation Project of Humanities and Social Sciences (19YJCZH095); and the Natural Science Foundation of Jiangsu Education Department of China (18KJB520006).

References

[1] M. Chinosi and A. Trombetta, "BPMN: an introduction to the standard," *Computer Standards & Interfaces*, vol. 34, no. 1, pp. 124–134, 2012.

[2] A. Correia and F. B. e. Abreu, "Adding preciseness to BPMN models," *Procedia Technology*, vol. 5, pp. 407–417, 2012.

[3] F. Corradini, A. Morichetta, A. Polini, B. Re, L. Rossi, and F. Tiezzi, "Correctness checking for BPMN collaborations with sub-processes," *Journal of Systems and Software*, vol. 166, no. 8, Article ID 110594, 2020.

[4] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[5] A. Armando and S. E. Ponta, "Model checking authorization requirements in business processes," *Computers & Security*, vol. 40, no. 2, pp. 1–22, 2014.

[6] D. Borrego, R. M. Gasca, and M. T. Gómez-López, "Automating correctness verification of artifact-centric business process models," *Information and Software Technology*, vol. 62, no. 6, pp. 187–197, 2015.

[7] R. Dijkman and P. Van Gorp, "BPMN 2.0 execution semantics formalized as graph rewrite rules," *Business Process Modeling Notation 67 of the Series Lecture Notes in Business Information Processing*, Springer, Berlin, Germany, pp. 16–30, 2010.

[8] P. Van Gorp and R. M. Dijkman, "A visual token-based formalization of BPMN 2.0 based on in-place transformations," *Information and Software Technology*, vol. 55, no. 2, pp. 365–394, 2013.

[9] F. Ul Muram, H. Tran, and U. Zdun, "Supporting automated containment checking of software behavioural models using model transformations and model checking," *Science of Computer Programming*, vol. 174, pp. 38–71, 2019.

[10] L. E. Mendoza, M. I. Capel, and M. A. Pérez, "Conceptual framework for business processes compositional verification," *Information and Software Technology*, vol. 54, no. 2, pp. 149–161, 2012.

[11] F. Corradini, A. Morichetta, C. Muzi, B. Re, and F. Tiezzi, "Well-structuredness, safeness and soundness: a formal classification of BPMN collaborations," *Journal of Logical and Algebraic Methods in Programming*, vol. 119, no. 2, Article ID 100630, 2021.

[12] J. Billington, S. Christensen, K. van Hee et al., "The petri net markup language: concepts, technology, and tools," *Applications and Theory of Petri Nets 2003*, vol. 2679, pp. 483–505, 2003.

[13] J. Fabra, P. Álvarez, J. Banares et al., "DENEB: a platform for the development and execution of interoperable dynamic web processes," *Concurrency and Computation: Practice & Experience*, vol. 23, no. 8, pp. 2421–2451, 2011.

[14] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, 2005.

[15] M. Jüngel, E. Kindler, and E. Weber, "The petri net markup language," in *Workshop Algorithmen und Werkzeuge für Petrinetze*, S. Philippi, Ed., pp. 47–52, Universität Koblenz-Landau, Mainz, Germany, 2000.

[16] J. Boubeta-Puig, G. Díaz, H. Macià, V. Valero, and G. Ortiz, "MEdit4CEP-CPN: an approach for complex event processing modeling by prioritized colored petri nets," *Information Systems*, vol. 18, no. 3, pp. 267–289, 2019.

[17] E. Tello-Leal, O. Chiotti, and P. D. Villarreal, "Software agent architecture for managing inter-organizational collaborations," *Journal of Applied Research and Technology*, vol. 12, no. 3, pp. 514–526, 2014.

[18] S. Philippi, "Automatic code generation from high-level petri-nets for model driven systems engineering," *Journal of Systems and Software*, vol. 79, no. 10, pp. 1444–1455, 2006.

[19] R. Kim, J. Gangolly, and P. Elsas, "A framework for analytics and simulation of accounting information systems: a petri net modeling primer," *International Journal of Accounting Information Systems*, vol. 27, no. 11, pp. 30–54, 2017.

[20] B. Meyers and H. Vangheluwe, "A framework for evolution of modelling languages," *Science of Computer Programming*, vol. 76, no. 2, pp. 1223–1246, 2011.

[21] E. Rahm and P. A. Bernstein, "An online bibliography on schema evolution," *ACM SIGMOD Record*, vol. 35, no. 4, pp. 30-31, 2006.

[22] L. Montalvillo and O. Díaz, "Requirement-driven evolution in software product lines: a systematic mapping study," *Journal of Systems and Software*, vol. 122, no. 12, pp. 110–143, 2016.

[23] E. Domínguez, J. Lloret, B. Pérez, Á. Rodríguez, Á. L. Rubio, and M. a. A. Zapata, "Evolution of XML schemas and

documents from stereotyped UML class models: a traceable approach," *Information and Software Technology*, vol. 53, no. 1, pp. 34–50, 2011.

[24] M. Camargo, M. Dumas, and O. González-Rojas, "Automated discovery of business process simulation models from event logs," *Decision Support Systems*, vol. 134, no. 7, Article ID 118384, 2020.

[25] Z. Li, X. Zhou, A. Gu, and Q. Li, "A complete approach for CIM modelling and model formalising," *Information and Software Technology*, vol. 65, no. 9, pp. 39–55, 2015.

[26] B. Bousetta, O. Beggar, and T. Gadi, "A methodology for CIM modelling and its transformation to PIM," *Journal of Thermal Science and Engineering Applications*, vol. 3, no. 2, pp. 1–21, 2013.

[27] V. De Castro, E. Marcos, and J. M. Vara, "Applying CIM-to-PIM model transformations for the service-oriented development of information systems," *Information and Software Technology*, vol. 53, no. 1, pp. 87–105, 2011.

[28] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "Towards CIM to PIM transformation: from secure business processes defined in BPMN to use-cases," in *Proceeding of 5th International Conference on BPM*, pp. 408–415, Springer Verlag, Brisbane, Australia, 2007.

[29] J. Touzi, F. Benaben, H. Pingaud, and J. P. Lorré, "A model-driven approach for collaborative service-oriented architecture design," *International Journal of Production Economics*, vol. 121, no. 1, pp. 5–20, 2009.

[30] T. Takemura, "Formal semantics and verification of BPMN transaction and compensation," in *Proceedings of the 2008 IEEE Asia-Pacific Conference on Services Computing*, pp. 284–290, IEEE Computer Society, Los Alamitos, CA, USA, 2008.

[31] R. Yu, Z. Huang, L. Wang, and H. Zhang, "Analyzing BPMN with extended object petri net," *Journal of Software Engineering*, vol. 8, no. 2, pp. 58–74, 2014.

[32] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Hierarchical verification for the BPMN design model using state space analysis," *IEEE Access*, vol. 7, pp. 16795–16815, 2019.

[33] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Transformation of the BPMN design model into a colored petri net using the partitioning approach," *IEEE Access*, vol. 6, pp. 38421–38436, 2018.

[34] P. Y. H. Wong and J. Gibbons, "Property specifications for workflow modelling," *Science of Computer Programming*, vol. 76, no. 10, pp. 942–967, 2011.

[35] G. N. Rai, G. R. Gangadharan, and V. Padmanabhan, "Algebraic modeling and verification of web service composition," *Procedia Computer Science*, vol. 52, pp. 675–679, 2015.

[36] J. W. Bryans and W. Wei, "Formal analysis of BPMN models using event-B," *Formal Methods for Industrial Critical Systems*, vol. 6371, pp. 33–49, 2010.

[37] C. Arevalo, M. J. Escalona, I. Ramos, and M. Domínguez-Muñoz, "A metamodel to integrate business processes time perspective in BPMN 2.0," *Information and Software Technology*, vol. 77, no. 9, pp. 17–33, 2016.

[38] M. Estañol, M.-R. Sancho, and E. Teniente, "Ensuring the semantic correctness of a BAUML artifact-centric BPM," *Information and Software Technology*, vol. 93, no. 1, pp. 147–162, 2018.

[39] F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi, "Formalising and animating multiple instances in BPMN collaborations," *Information Systems*, vol. 85, Article ID 101459, 2019.

[40] G. Kang, L. Yang, and L. Zhang, "Verification of behavioral soundness for artifact-centric business process model with synchronizations," *Future Generation Computer Systems*, vol. 98, no. 9, pp. 503–511, 2019.

[41] P. Felli, M. de Leoni, and M. Montali, "Soundness verification of decision-aware process models with variable-to-variable conditions," in *Proceedings of the 19th International Conference on Application of Concurrency to System Design (ACSD)*, pp. 82–91, Aachen, Germany, 2019.

[42] M. Foughali and P.-E. Hladik, "Bridging the gap between formal verification and schedulability analysis: the case of robotics," *Journal of Systems Architecture*, vol. 111, no. 12, Article ID 101817, 2020.

[43] B. Aristyo, K. Pradityo, T. A. Tamba et al., "Model checking-based safety verification of a petri net representation of train interlocking systems," in *Proceedings of the 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pp. 392–397, Nara, Japan, 2018.

[44] S. Zatout, M. Souilah Benabdelhafid, and M. Boufaida, "Formal transaction modeling and verification for an adaptable web service orchestration," in *Proceedings of the 2018 IEEE International Conference on Software Quality, Reliability and Security Companion*, pp. 531–536, Lisbon, Portugal, 2018.

[45] R. Mrasek, J. Mülle, and K. Böhm, "A new verification technique for large processes based on identification of relevant tasks," *Information Systems*, vol. 47, pp. 82–97, 2015.

[46] B. Cao, F. Hong, J. Wang, J. Fan, and M. Lv, "Workflow difference detection based on basis paths," *Engineering Applications of Artificial Intelligence*, vol. 81, no. 3, pp. 420–427, 2019.

[47] Q. Hu, Y. Du, and S. Yu, "Service net algebra based on logic petri nets," *Science China-Information Sciences*, vol. 268, no. 6, pp. 271–289, 2014.

[48] C. Chen, Y. Yang, and X. Zhang, "Knowledge characterization and evolution methodology of emergency scenario based on hybrid petri net," in *Proceedings of the 2017 International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD 2017)*, Guilin, China, 2017.

[49] L. Capra and M. Camilli, "Towards evolving petri nets: a symmetric nets-based framework," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 480–485, 2018.

[50] K. Gabriel and H. Ehrig, "Modelling evolution of communication platforms and scenarios based on transformations of high-level nets and processes," *Theoretical Computer Science*, vol. 429, no. 4, pp. 87–97, 2012.

[51] Y. I. Khan, "Optimizing verification of structurally evolving algebraic petri nets," in *SERENE 2013: Software Engineering for Resilient Systems, Lecture Notes in Computer Science*, vol. 8166, pp. 64–78, Springer, Berlin, Germany, 2013.

[52] A. Krishna, P. Poizat, and G. Salaün, "Checking business process evolution," *Science of Computer Programming*, vol. 170, pp. 1–26, 2019.

[53] X. Zhu, S. vanden Broucke, G. Zhu, J. Vanthienen, and B. Baesens, "Enabling flexible location-aware business process modeling and execution," *Decision Support Systems*, vol. 83, no. 3, pp. 1–9, 2016.

[54] S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, and P. Van Gorp, "Evaluation of model transformation approaches for model refactoring," *Science of Computer Programming*, vol. 85, no. 7, pp. 5–40, 2014.

[55] H. P. Einarsson, "Refactoring UML diagrams and models with model-to-model transformations," Master thesis, University of Iceland, Reykjavik, Iceland, 2011.

[56] Y. Ye, Z. Jiang, X. Diao, and G. Du, "Extended event-condition-action rules and fuzzy Petri nets based exception handling for workflow management," *Expert Systems with Applications*, vol. 38, no. 9, pp. 10847–10861, 2011.