# Hierarchical Verification of the BPMN Design Model Using the State Space Analysis

**Article** *in* IEEE Access · January 2019

DOI: 10.1109/ACCESS.2019.2892958

**3 authors:**

Chanon Dechsupa
Chulalongkorn University
**16** PUBLICATIONS **69** CITATIONS
SEE PROFILE

Wiwat Vatanawood
Chulalongkorn University
**75** PUBLICATIONS **359** CITATIONS
SEE PROFILE

Arthit Thongtak
Chulalongkorn University
**27** PUBLICATIONS **93** CITATIONS
SEE PROFILE

Received December 15, 2018, accepted January 7, 2019, date of publication January 14, 2019, date of current version February 14, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2892958*

# Hierarchical Verification for the BPMN Design Model Using State Space Analysis

**C. DECHSUPA, W. VATANAWOOD, AND A. THONGTAK**
Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330, Thailand

Corresponding author: W. Vatanawood (wiwat@chula.ac.th)

The work of C. Dechsupa was supported in part by the 100th Anniversary Chulalongkorn University Fund for Doctoral Scholarship, in part by the 90th Anniversary of Chulalongkorn University Fund (Ratchadaphiseksomphot Endowment Fund), and in part by the Overseas Research Experience Scholarship for Graduate Student.

**ABSTRACT** The BPMN design models are widely used in the software development process. Owing to the lack of BPMN standard semantics, formal verification is used to validate whether the BPMN design model is free of undesirable properties. The primary challenges of BPMN design model verification are the enormous size and complexity of BPMN design models; these challenges may lead to time-consuming processes for model abstraction and overcoming the state space explosion problem. This paper proposes a hierarchical verification technique for the state space analysis based on a colored Petri net (CPN). A BPMN partitioning technique and rules for the transformation of a BPMN into a CPN model are provided. The partitioning approach supports the unstructured BPMN design model, and the obtained CPN model also supports hierarchical verification. To validate and analyze the BPMN design model, the transformation technique and state space generator are implemented as a BPMN verification framework. This method is a viable option for the software process designers and is suitable for the large-scale BPMN design model verification.

**INDEX TERMS** Formal verification, BPMN, colored Petri net, model transformation, hierarchical verification.

## I. INTRODUCTION

The BPMN design models are used to bridge the communication gap between software process designers and developers. Business process model notation (BPMN) [1] enables designers to design various types of software models, such as process diagrams, orchestration diagrams and choreography diagrams. BPMN design models are likely to have flaws or undesirable properties as a result of the use of inappropriate or ambiguous notation. Specifically, the huge design model of a concurrent system that has sophisticated behavior cannot be validated by using ordinary testing approaches or general formal verification techniques.

The main obstacle of BPMN design model verification is the state space explosion problem [2], which is the result of the huge abstract model that produces tremendous state space size exceeding the capacity of the verification tool. The state space size of the abstract model is dependent on the number of activities, the control flow dependency [7], the data dependency, the data type variety and the capacity of the concurrent execution level of the designed model. Hierarchical verification [9] is an alternative solution to resolve or avoid the state space explosion problem. In hierarchical verification, the BPMN design model is sliced into appropriate partitions, and structural-based partitioning is applied to reduce the size of the abstract model. Each partition can be chosen to be verified arbitrarily or determined as a black-box process for the hierarchical verification. However, these techniques are complicated for designers who are not familiar with formal languages and verification tools; thus, various researchers have proposed solutions to perform the verification procedures.

This paper provides a hierarchical verification approach that is an extension of our previous work [5], [6]. The rules for BPMN transformation into a CPN model are extended to handle both structured and unstructured BPMN models [8]. Additionally, the BPMN model partitioning technique is modified by applying the weight-based consideration. We also provide a hierarchical verification approach using state space analysis. The state space is constructed from the CPN sub-nets derived from the automated BPMN model transformation. The sweep-line method [14], [15] is applied

The associate editor coordinating the review of this manuscript and approving it for publication was Mark Ng.

in the state space construction algorithm to decrease the state space size. For practical use, the proposed techniques are implemented as a Java application tool named CP4BPMN to perform BPMN design model verification. This process can reduce time consumption and mistakes and increase the transparency of complicated verification procedures.

This paper focuses on a hierarchical verification technique to verify large-scale BPMN design models. This paper is organized as follows: section 2 presents the background of the BPMN software process model and the model checking procedures. Section 3 discusses the related research. Section 4 describes the proposed approach and its implementation. Finally, the experiments, results, and conclusions are presented in Section 5.

## II. BACKGROUND

BPMN provides graphical standard notation to design software process models, and many software products support the BPMN standard. The BPMN design model of software processes is a collection of BPMN elements composed of control flows, data flows and related activities that produce the specific operation to achieve the business requirements. The modelers can use these elements to describe the BPMN process diagram or collaboration diagram for more sophisticated behavior. The core subset of BPMN elements is divided into six groups: 1) Event elements, 2) Task elements, 3) Gateway elements, 4) Connecting elements, 5) Artifacts, and 6) Swim-lanes and pools. The responsibilities of each element and the formal definition of BPMN models are detailed in [5]. Figure 1 shows the core subset of BPMN elements.

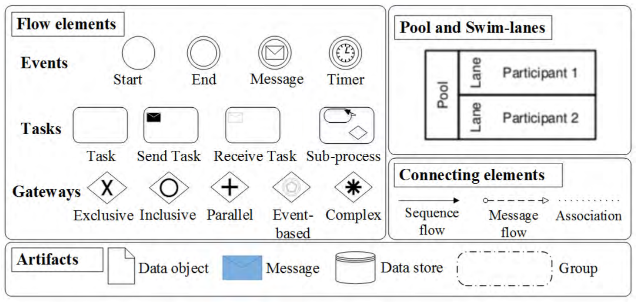


**FIGURE 1.** Core set of BPMN elements [1].

Model checking [10] is the formal verification approach, and the verification procedure requires an abstract model written in formal language. The abstract model is used to generate a state space and explore the model's properties. This step requires a state space generator, including temporal logic [11] interpreted over the state space. The core verification procedure consists of five steps: 1) defining the objective, scope and boundary of verification 2) determining the formal language [12] used for modeling an abstract model, 3) formalizing the desirable properties in temporal logic, 4) validating an abstract model using the model simulation approach or state space analysis approach, and 5) the resulting interpretation.

CPN is an outstanding formal modeling language for designing and verifying a concurrent system. CPN is an enhanced form of the traditional Petri net (PT) [13] obtained by integration of the advantages of the Petri net semantics and programming language. The elements of CPN consist of *place*, *transition*, *guard condition*, *arc, arc inscription* and *colored set*. A directed arc is used as a bridge between place and transition. The place serves as a container for accumulating tokens. A token's value is called the *token color*, whose data type is defined by the *colored set*. Transition firing is a situation where the transition consumes the token at its input place and produces a token at its output place. This situation represents a state change in the system. Figure 2 (a) shows the elements of a CPN, Figures 2 (b) and (c) show the transition firing, and Figure 2 (d) is an example of a CPN model of a simple protocol.

A CPN enables modelers to construct compact and parameterized models. This feature is suitable for complex model verification. Several CPN-based verification tools support the design and verification of concurrent systems and enable designers to validate CPN models via two modes. *Simulation mode* involves checking the model in a particular control action, whereas *verification mode* considers all possible states of the system. Verification mode relies on a state space generator to construct the *reachability graph*. The reachability graph, or *state space graph*, consists of *nodes* and *edges*. A node, or *marking*, represents a system's state, whereas an edge represents a state transition, which is the transition's firing information, called the *binding elements*. The functional commands and temporal formulas interpreted over a state space are used to create the queries for state space exploration.

State space explosion is a problem in the verification mode. The root cause of this problem is the large size and complexity of the model. A CPN model may become large and inconvenient; thus, it should be broken into CPN submodels called CPN sub-nets and refined into a hierarchical structure. To refine the hierarchical structure, black-box behavior, which represents the workings at a higher abstraction level, replaces the sub-net. The hierarchical structure can also reduce the complexity and state space size. A hierarchical CPN model can be combined with reduction methods to reduce the time consumption and state space storage requirements. For instance, the sweep-line method is a state space reduction technique used to avoid representing the entire state space. This method is exploits certain states that store only small fragments.

Generally, functional commands implemented via temporal logic are provided for creating queries to inquire about the state space. The type of temporal logic depends on the verification tool and includes linear temporal logic (LTL), computation tree logic (CTL) and CTL* [11]. The designers must express queries in the appropriate query syntax, and the verification tools may require additional libraries to explore the state space.

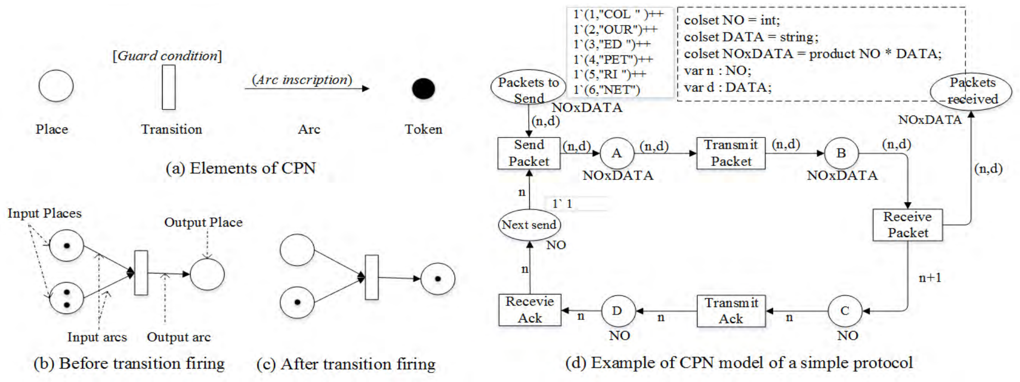


**FIGURE 2.** CPN elements and example of a CPN model [3]. (a) Elements of CPN. (b) Before transition firing. (b) After transition firing. (d) Example of CPN model of a simple protocol.

## III. RELATED WORK

Several BPMN verification are currently being employed, including both BPMN model transformation and behavioral verification using state space analysis. The model abstraction techniques and verification tools used in each paper are different. In the area of BPMN verification using a colored Petri net approach, transformation frameworks and CPN tools are used for flaw detection in BPMN design models. In [16] and [17], the transforming techniques were implemented based on BPMN 2.0 and CPN paradigms. The transformation tools can extract the BPMN elements in XMI format and transform them into CPN XML-based representations. This representation can be supported and verified via CPN tools. The transformation rules focus on the control flows of BPMN model, but they do not address the data flow perspective or the model complexity. Although the technique of [17] focused on data evaluation of the control flow, the rules do not cope with the actual input and output data of a task. Ramadan *et al.* [18], the authors focused the BPMN verification on the loop, sub-process and transaction of the BPMN model. The authors applied the CPN place to represent the task's properties; however, they did not provide a colored set handling and an automated transformation framework.

The classic Petri net approach is similar to the CPN approach. In [19], the traditional Petri net was used to represent the abstract BPMN model by transforming the BPMN elements via their transformation tool. The abstract model was expressed in the Petri Net Markup Language. This work handled the ordinary process design and addressed the complexity of the sub-process, muti-instance and task dependency, but the transformation rules did not follow the data flow perspective. In [20], the formal specification used the Petri net-based language called ECATNets. The meta-model for transforming BPMN elements into a Petri net and framework were provided, and the data flow, control flow, and multiple instances, including exception handling, can be addressed using the transformation language defined as the metadata between BPMN and RECATNet. Next, the Maude model checker was used to verify the soundness of the ECATNets model. Although the task dependencies were conducted, the authors did not consider the actual data values of a task. In [21], an algorithm for reachability checking and flaw detection techniques were provided using time Petri nets (TPN) [22]. The abstract model was described using a formal mapping of the BPMN model into a time Petri nets model, but the authors did not consider the data flow of the BPMN model. Similarly, von Stackelberg *et al.* [23] and AwadGero and Lohmann [24] performed data flow error detection in the BPMN model using the classic Petri net approach. Anti-patterns were applied in the data flow error detection technique of [23]. They also provided the transformation rules and state space analysis of BPMN data flow verification using the model checker LoLA. The techniques of [23] and [24] focused on data anomaly detection in BPMN design models, and the data flow representation was still not the actual data flow in the BPMN model.

Some studies have included experiments outside the context of the CPN modeling language for verifying BPMN design models via timed automata [25], Pi-Calculus [26] and process algebra [27]. Malleka *et al.* [25] provided an interoperability checking technique for the BPMN collaborating diagram. The BPMN model was mapped to a formal model written in UPPAAL language, that is, timed automata. The UPPAAL model checker [28] was used for verifying the model's properties. This work was limited in terms of the enormous state space size, and the interoperability checking depended on the resulting responses of the UPPAAL model checker, including the property formalization. Boussetoua *et al.* [26] provided an intermediate model to transform the BPMN models to Pi-Calculus. The paper illustrated only simple BPMN constructs. Malleka *et al.* [25] used the equivalence checking technique to encode the BPMN choreographed model into process algebra and used the VerChor framework [29] to verify their models.

Durán *et al.* [33] enhanced the extension BPMN semantics representation supporting conditions and data flow. They provided the symbolic specification of BPMN enriched with

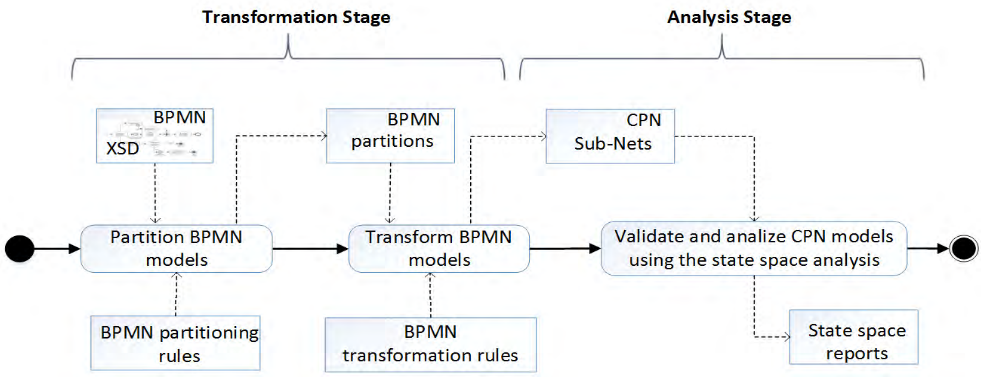


**FIGURE 3.** **Overview of the BPMN verification process.**

data, for verifying deadlock freedom and reachability activities. The BPMN models were represented and verified using Maud's rewriting logic framework. The authors did not provide the automated transformation framework. This paper illustrates only simple BPMN models and their verification approach still faces with the state space explosion problem. Massimiliano *et al.* [34] proposed the fashion to verify the soundness properties of DPN which is Petri net based process model. They illustrated the mapping rules for soundness checking of the decision enriched with data and decision in DPN model, it indicate that the advantages of CPN are useful to verify the soundness properties. This is a reason why we use CPN to be a formal language for representing abstract model of our work.

Almost previous related works emphasized only on the BPMN structural verification which did not consider the actual input and output data of the tasks so that the state space size of those works may possibly be handled by a verification tool used. In contrast, the state spaces size of the behavioral verification concerning the use the data flows tends to increase exponentially. Our work will address the huge state space and improve the CPN model abstraction to achieve the behavioral verification using the partitioning and hierarchical approaches.

## IV. METHODOLOGY

The intention of this section is to describe our verification methodology. The overview of the BPMN verification process is shown in Figure 3. The methodology is divided into the transformation stage and verification stage. The transformation stage describes the related formal definitions, technical terms of the transformation and framework architecture, whereas the verification state describes the state space analysis based on the capability of our framework. On the basis of the formal definitions in [5], we present additional formal definitions involving an extension. The details of each stage are described in subsections A and B.

Due to the increase in the state space size, we coped with this challenge by providing the gateway pairwise analysis to partition the BPMN model into sub-modules called the BPMN partitions. The advantage of the gateway pairwise partitioning is that the pattern of the obtained partition is a block, and it can also be easily represented and investigated using CPN. Whereas, the complexity of BPMN element can be handled by the weight-based consideration. The combination of the gateway pairwise and weight-based approaches yields the BPMN partitions supporting the partial and hierarchical verification. To decrease the state space size, we applied the CPN hierarchical verification technique and sweep-line method to reduce the run-time explosion. Our proposed techniques are the transparent procedures which are proceeded by our framework automatically.

*Definition 1 (A Hierarchical CPN Model):* A hierarchical CPN model is a tuple $HCPN = (CPN, TS, PS, fPS)$ where:

- $CPN$ is a set of non-hierarchical models.
- $TS$ is a set of special substituted transitions used as agents of the sub-nets $cpn \mid cpn \in CPN$.
- $PS$ is a set of special port locations determined as the input place or output place of the substitution transition.
- $fPS$ is a mapping function used to indicate the type of port place direction, $fPS\colon PS \rightarrow \{Input, Output\}$.

Hierarchical CPN model is a CPN model containing multiple non-hierarchical CPN sub-models or sub-nets, and some of them are determined to be the representation in higher abstraction level. For determining the higher abstraction level, the implementations of all sub-nets are hidden, and the substituted transitions are represented instead of such sub-nets. The interfaces of the substituted transition will be addressed by the input/output port places.

*Definition 2 (A Reachability Graph):* A reachability graph is a tuple $DRG = (MN, EG, fFI)$ where:

- $MN$ is a set of nodes that are the *reachable markings*.
- $EG$ is a set of edges such that $(MN \cap EG) = \varnothing$.
- $fFI$ is a mapping function used to indicate the firing information, $fFI\colon (MN, EG) \rightarrow$ The *Binding elements*.

The reachability graph is constructed by a state space generator and will be explored by the queries expressed in term of the temporal logic. The core component of reachability graph consists of nodes and edges. The node is a marking or an state of system while the edge represents the state transitional information that is the data collection of the transition firing and variables snapshot called the binding element. The binding element is used to interpret an identified or identifiable firing transition in the state space exploration process.

*Theorem 1 (Hierarchical Verification):* Let CPN model $M$ consist of sub-nets $\{s_1, \ldots, s_{m-1}, s_{m(1)}, \ldots, s_{m(i)}, s_{m+1}, \ldots, s_n\}$. Model $M$ conforms to model $Mx$ if and only if $\{s_{m(1)}, \ldots, s_{m(i)}\}$ conforms to sub-net $s_m$ and $\{s_1, \ldots, s_{m-1}, s_m, s_{m+1}, \ldots, s_n\}$ conforms to $Mx$.

Hierarchical verification is that the CPN model is partitioned into sub-nets, $\{s_1, s_2 \ldots, s_n\}$ is a set of sub-nets. Each sub-net can also be re-partitioned once again as a set of semi sub-nets. For example, sub-net $s_1$ can be re-partitioned into $\{s_{1-1}, s_{1-2} \ldots, s_{1-n}\}$. The flat CPN model $M$ conforms to the hierarchical CPN model $Mx$ if and only if all sub-nets of the hierarchical CPN model conform to that of flat CPN model even some sub-nets of hierarchical CPN model are determined to be the substituted transitions.

## A. BPMN DESIGN MODEL PARTITIONING

A BPMN model partitioning technique is provided for handling the large-scale BPMN design model. The data of BPMN model in XMI format are retrieved and stored for the model structural analysis. The execution paths of BPMN design models are similar to directed graphs, where each BPMN element acts as a node of a directed graph. We modify the partitioning algorithm of [5] by combining the structural decomposition [30] approach with the weight-based consideration. We first use the node type and gateway pairwise analysis to determine the preliminary boundary of the BPMN partitions. The size of the BPMN partitions derived from the gateway pairwise analysis may be larger or smaller than the configured weight. Next, the weight-based consideration is used for re-partitioning the large partitions or composing the dangling partitions. The modified algorithm supports unstructured BPMN models and yields appropriate BPMN partitions.

Seven flow patterns are derived from the partitioning stage: 1) *sequence*, 2) *If-then*, 3) *If-then-else or Parallel*, 4) *Repeat-until*, 5) *While-do*, 6) *Do-while-do*, and 7) *Indeterminate pattern*. The indeterminate pattern is a result of either the unstructured flow or the re-partitioning of the partitions over the configured weight. The node types of the BPMN directed graph and the flow patterns are shown in Figure 4.

Algorithm 1 illustrates the BPMN partitioning process. Lines 1 and 2 check the model types and whether the BPMN process is well defined. If the input model is a process diagram, the number of pools is set to 1. Lines 4 to 27 are the graph traversal for the gateway pairwise analysis. The gateway matching criteria and the partition number identification are in lines 9 to 16, and the function node is labeled with the partition identifier in lines 17 to 21. After all the nodes in the BPMN design model have been labeled, the BPMN partitions over the configured size are re-partitioned, and the trivial partitions are merged if their total weight

**Algorithm 1** Partitioning BPNM Model in to BPMN Partitions

```
01 Require:BPMN design model is an ordinary, hierarchical
           or collaboration diagram.
02 Ensure:well-defined BPMN design model.
03      If BPMN design model is without pool then set $P = 1$
EndIf
04                If $PL > 0$ then
05                 For $swl \in PL$
06                          $SG := lb(\text{n}) \rightarrow Es$
07                   Do $SG \neq \emptyset$
08                          $n$ := choose the node from
                              $SG$ by FIFO ordering
09                          If $n$ is gateway then
10                              If $n$ does not a pairwise
                                  of previous node then
11                                  set $par\_id$=increase
                                  partition number
12                           $lb\_n = \{n, par\_id\}$
13                            Else
14                                  $lb\_n = \{n, par\_id\}$
15                                  set $par\_id$=increase
                                     partition number
16                            End if
17                          Else if $n$ is a function node
                            then $lb\_n = \{n, par\_id\}$
18                          Else $n$ is a sub process
19                              set $par\_id = create\ new$
                                  $partition\ number$
20                              Partitioning($n$)
21                          End if
22                           $SO$: $SG \cup$ {choose node(s)
                            from $BG$ where source
                            node $= n$}
23                           $SG$:=$SG \setminus \{n\}$
24                           $HBP$:=$HBP \cup lb\_n$
25                      End do
26                    Next
27               End if
28   While $HBP$: $par\_id$/*Check the size of sub-models*/
29   BEGIN
29        $Sub\_modelSize$=CalculateSizeBasedWeight
            ($par\_id$)
30        If $Sub\_modelSize$>$SizeConfig$ then
31             Repartition the sub-model by recursive
               function
32             $HBP$: = ($HBP|par\_id$) $\cup$ re-partition
               ($par\_id$)
33         Else
34             Merge the sub-partitions with neighbor-
                 ing partitions if their total weight
35             does not exceed $SizeConfig$.
36             $HBP$: = ($HBP$ | $par\_id$) $\cup$ mergeParti-
               tion($par\_id$, neighborhood($par\_id$))
37        End If
```

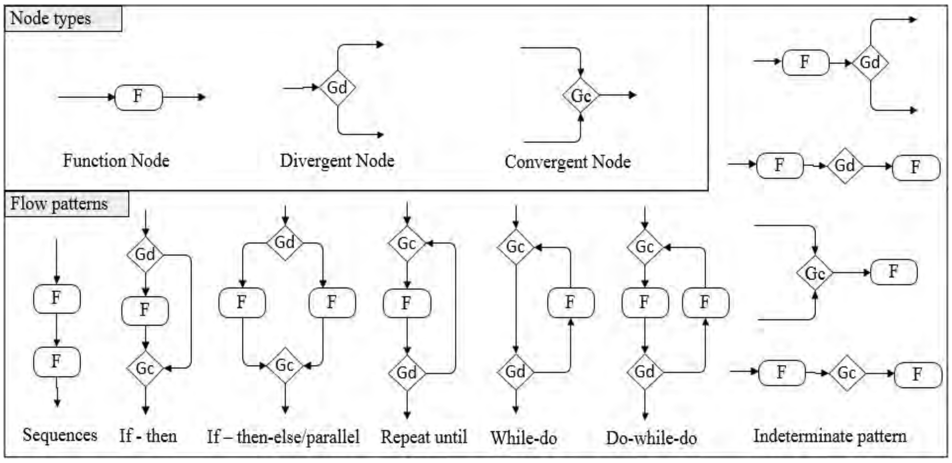


**FIGURE 4.** Node types and flow patterns in the BPMN directed graph.

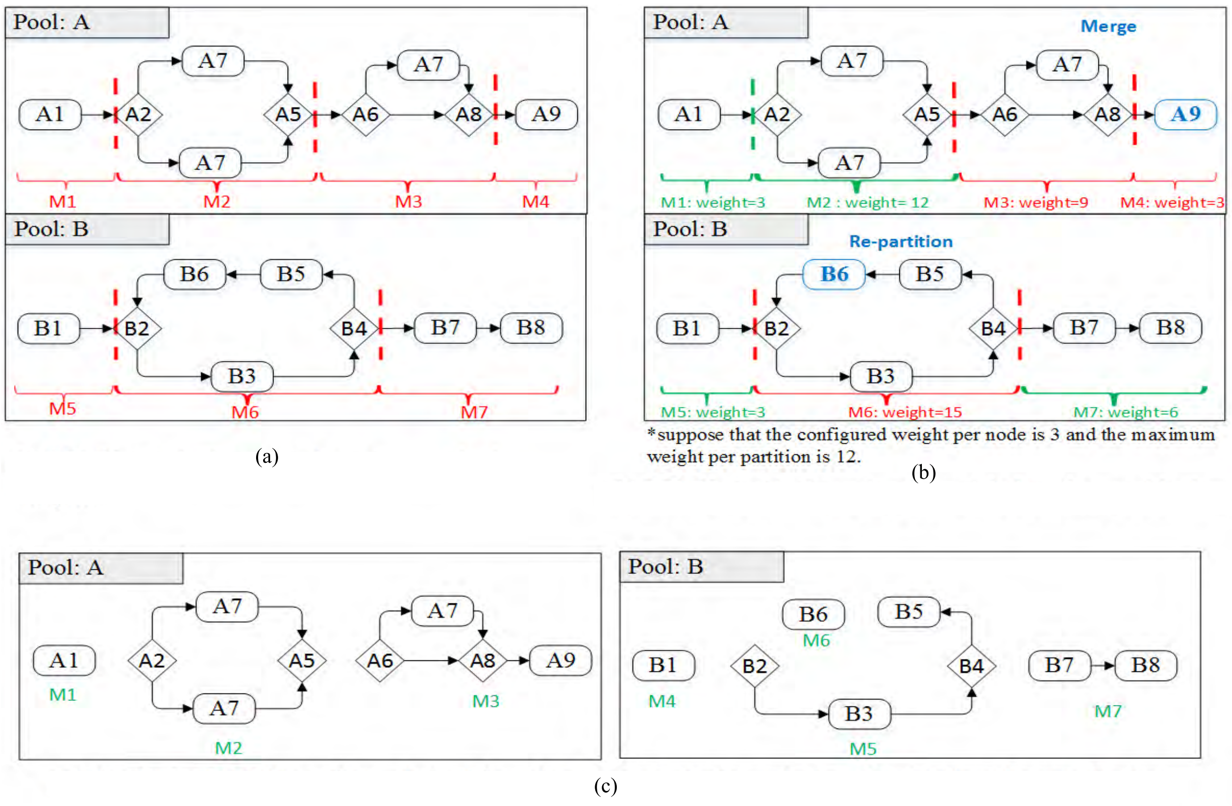


**FIGURE 5.** BPMN model partitioning concept. (a) BPMN partitioning using the gateway pairwise analysis. (b) BPMN partitioning using the weight based re-partitioning. (c) The obtained BPMN partitions derived from the gateway pairwise analysis and weight based re-partitioning in Figure (a) and (b) respectively.

**Algorithm 1** *(Continued.)* Partitioning BPNM Model in to BPMN Partitions

```
38  End while
39  Return HBP
```

does not exceed the configured size. The re-partitioning and merging via the weight-based consideration are in lines 28-38.

Figure 5 shows the cursory BPMN model partitioning concept. Figure 5(a) is an original BPMN design model partitioned using the gateway pairwise analysis. The model is composed of two pools (A and B), and each pool is analyzed independently. Seven BPMN partitions are derived from the gateway pairwise analysis. Next, the partitions are assessed with the weight-based consideration shown in Figure 5(b). Suppose that the configured weight per node is 3 and that the maximum weight per partition is 12. Partitions M3 and

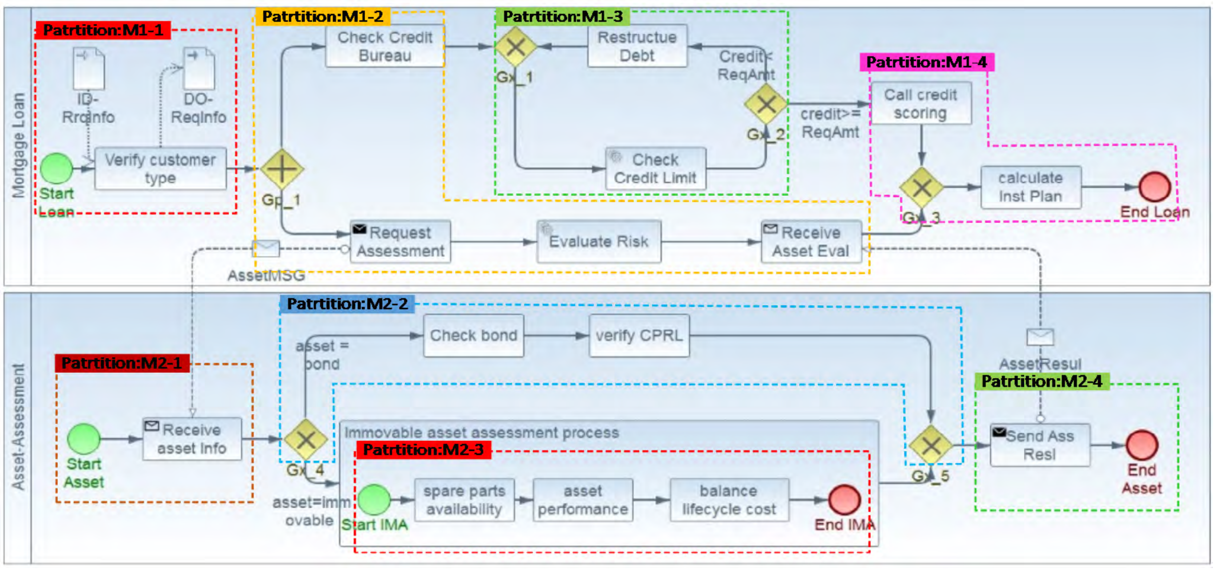


**FIGURE 6.** An example of a partitioned BPMN design model.

**TABLE 1.** Partitioning log of the gateway pairwise analysis of the BPMN model in Figure 6.

| Partition ID | Element names | Patterns | Total Weight |
|---|---|---|---|
| M1-1 | Start Loan, Verify member type | Sequence | 6 |
| M1-2 | Gp_1, Check Credit bureau, Request Assessment, Evaluate risk, Receive Asset Eval | Indeterminate | 15 |
| M1-3 | Ex_1, Check credit Limit, Gx_2, Restructure debt | Do-while-do | 12 |
| M1-4 | Call credit scoring, | Sequence | 3 |
| M1-5 | GX_3, Calculate Inst Plan, End Loan | Indeterminate | 9 |
| M2-1 | Start Asset, Receive asset Info | Sequence | 6 |
| M2-2 | Gx_4, Check bond, Verify CPRL, Start IMA, Spare part availability, Asset performance, Balance lifecycle cost, End IMA, Gx_5 | If-then-else | 27 |
| M2-3 | Send Asst resl, End Asset | Sequence | 6 |

M4 of pool A can be merged because partition M4 is a dangling partition and the combined weight of partitions M3 and M4 does not exceed 12. By contrast, partition M6 of pool B is overweight and is therefore re-partitioned. Thus, seven final BPMN partitions are obtained from the gateway pairwise analysis and weight-based consideration, as shown in Figure 5(c).

Figure 6 shows the implementation of the partitioning algorithm with the real collaboration BPMN process model. The model is part of the process of a mortgage loan system. Table 1 shows the portioning log of the gateway pairwise analysis. The statistical analysis is based on the configured weight. The weight per node is 3, and the maximum weight per partition is 15. Thee preliminary partitioning is the classification of BPMN elements using the pool elements (with the prefixes *M1* and *M2* in column *Partition ID*). Next, the nodes in every pool are classified by gateway matching. Some of the BPMN partitions may become larger or smaller than the maximum configured size.

As shown in Table 1, partition *M2-2* exceeds configured weight, and partitions *M1-1*, *M1-4*, *M1-5*, *M2-1* and *M2-3* are trivial partitions. Thus, partition *M2-2* is repartitioned using the weight-based consideration into *M2-2* and *M2-3*. By contrast, the partitions *M1-4* and *M1-5* are merged. However, partitions *M1-1*, *M2-1* and *M2-3* cannot be merged with the neighboring partitions because the resulting weight would exceed the maximum weight per partition. The partitioning log of the weight-based re-partitioning is shown in Table 2. In the BPMN design model in Figure 6, the dashed-line blocks are the final partitions that are passed to the gateway pairwise analysis and weight-based consideration.

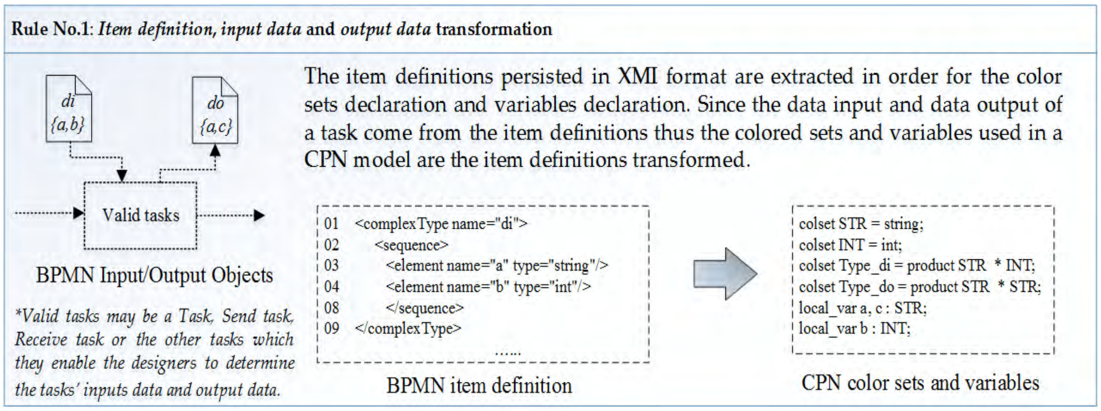
## *B. BPMN DESIGN MODEL TRANSFORMATION*

After model partitioning, each BPMN element is transformed into a CPN construct and concatenated into a a CPN model. The CPN sub-net is the CPN model derived from a set of BPMN elements in the same partition, and the partition identifier of the BPMN model is shared by the CPN model. The BPMN designing tools do not restrict the programming language used for model expression, which may result in a syntactical difference in the inscriptions. The inscriptions of the CPN model consist of the color sets, variable declaration, guard condition expression and arc expression. We use the mapping rules from our previous work [5] and add the model inscriptions expressed in Java programming language. Thus, the inscriptions of the CPN model conform to the syntax and lexicon of Java. The mapping rules are illustrated in Figures 7, 8, 9 and 10.

**TABLE 2. The partitioning log after weight-based re-partitioning of the BPMN model in Figure 6.**

| Partition ID | Element names | Pattern | Total Weight |
|---|---|---|---|
| M1-1 | Start Loan, Verify member type | Sequence | 6 |
| M1-2 | Gp_1, Check Credit bureau, Request Assessment, Evaluate risk, Receive Asset Eval | Indeterminate | 15 |
| M1-3 | Ex_1, Check credit Limit, Gx_2, Restructure debt | Do-while-do | 12 |
| M1-4* | Call credit scoring, Gx_3, Calculate Inst Plan, End Loan | Indeterminate | 12 |
| M2-1 | Start Asset, Receive asset Info | Sequence | 6 |
| M2-2** | Gx_4, Check bond, verify CPRL, Gx_5 | Indeterminate | 12 |
| M2-3** | Start IMA, Spare part availability, Asset performance, Balance lifecycle cost, End IMA | Sequence | 15 |
| M2-4 | Send Asst resl, End Asset | Sequence | 6 |

*The BPMN partition derived from merging partitions *M1-4* and *M1-5* in Table 1; ** The BPMN partitions derived from splitting partition *M2-2* in Table 1.


Rule No.1: *Item definition, input data* and *output data* transformation
di
{a,b}
do
{a,c}
Valid tasks
BPMN Input/Output Objects
*Valid tasks may be a Task, Send task, Receive task or the other tasks which they enable the designers to determine the tasks' inputs data and output data.
BPMN item definition
CPN color sets and variables

The item definitions persisted in XMI format are extracted in order for the color sets declaration and variables declaration. Since the data input and data output of a task come from the item definitions thus the colored sets and variables used in a CPN model are the item definitions transformed.

```
<complexType name="di">
   <sequence>
      <element name="a" type="string"/>
      <element name="b" type="int"/>
   </sequence>
</complexType>
       ......
```

```
colset STR = string;
colset INT = int;
colset Type_di = product STR * INT;
colset Type_do = product STR * STR;
local_var a, c : STR;
local_var b : INT;
```

**FIGURE 7. The mapping rules to transform BPMN elements into CPN constructs.**

Since we create the CPN model using an automated transformation, the completeness of each CPN model passes the basic criteria check. However, the designers have to determine the initial marking, the boundary values of all the variables and the boundary of the places and must refine their CPN models before the state space construction. Figure 11 shows the CPN model derived from the transformation stage of the BPMN model in Figure 6. We have omitted the detailed information about the arc inscriptions and guard conditions of the transitions to simplify the model. The CPN sub-nets still conform to the partitions of the BPMN model. The sub-nets can be selected arbitrarily by the designer either one by one or all at once to generate the state space and validate the properties.

As the mentioned inscription, the data types and variables declaration, including the data manipulation expressions of the BPMN model, are straightforward mappings to the input and output arc inscriptions and the guard conditions of the CPN transition.

The variables used on the arc inscriptions may be the local variables, which are the transformed item definitions of the BPMN model, or the global variables, which are manually declared by the designers. We classify the inscription patterns of the CPN model as follows.

### 1) INPUT ARC INSCRIPTION

For passing token colors (data values) from an input place to the transition, the list of variables on the input arc must be the local variables, and their color sets have to match with the color set of the input place. Figure 12(a) illustrates the input arc inscription for passing token colors from the input to output location. The color set of place *p1* is *RequestInfo*, and the inscription of the arc connecting *p1* and transition *t1* is (*cust_name*, *request_no*). The colored sets of the variables *cust_name* and *request_no* conform to that of *RequestInfo*, whose color sets are *String* and *Integer*, respectively.

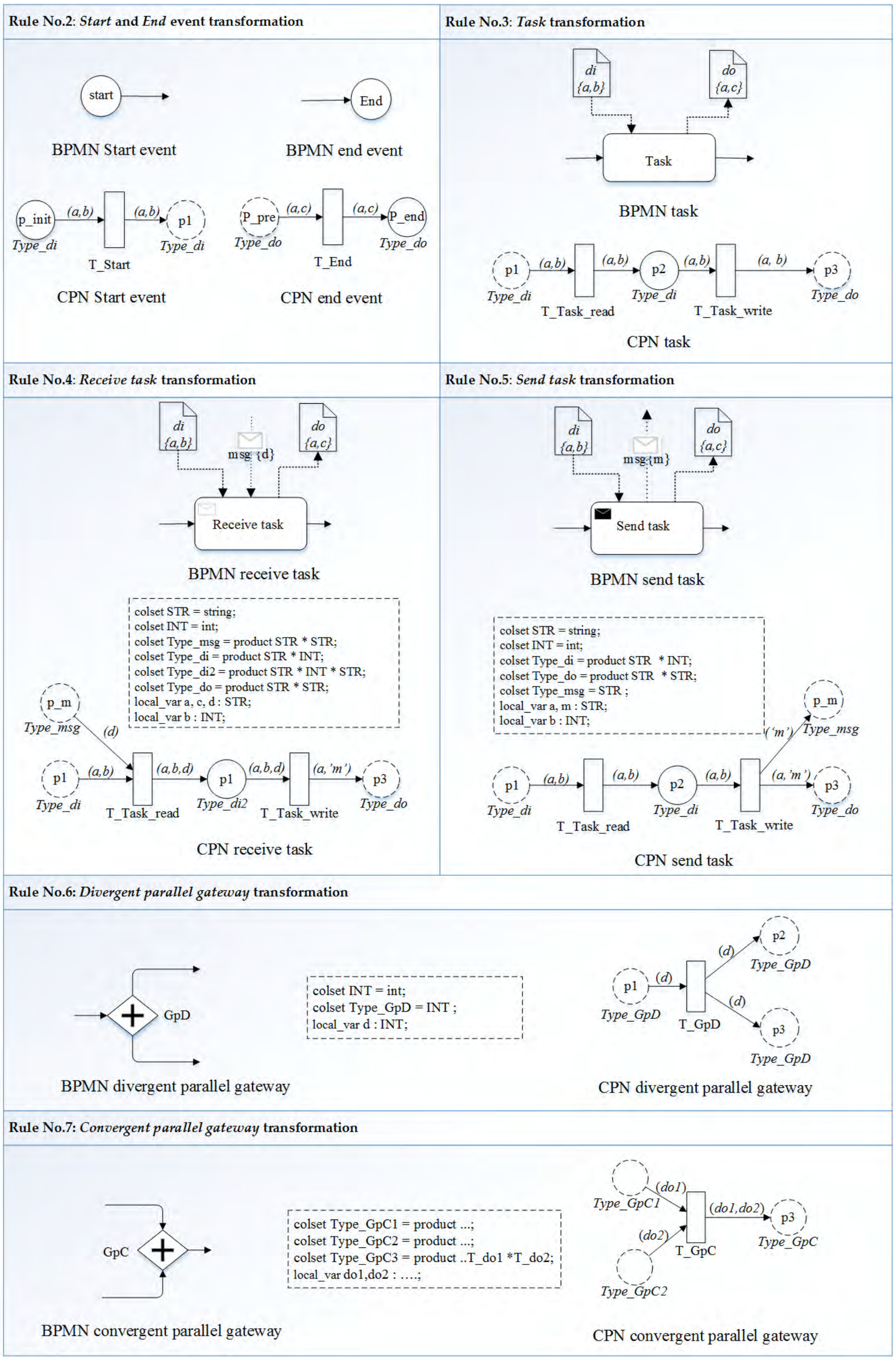
### 2) PREDICATE INSCRIPTION

The predicate inscription is used to express the guard condition of a CPN transition. The variables used in a predicate statement must be the local valuables on the input arcs or the global variables, and the result of the predicate evaluation must be *true* or *false*. Figure 12(b) shows an example of a predicate expression. The guard condition of the transition *t1* is "*request_no!=null & global_var_n>1*", for which the variable *request_no* is the local variable on the input arc and the variable *global_var_n* is the global variable.

### 3) OUTPUT ARC INSCRIPTION

The output arc inscription plays an important role in determining the token colors for the output place; the designers can express the output arc inscription in two fashions.

- **Direct passing**: The variables on the output arc may come from the variables on the input arc or may be global variables or constant values. Figure 13(a) shows the direct passing of an output arc inscription.



```
colset STR = string;
colset INT = int;
colset Type_msg = product STR * STR;
colset Type_di = product STR * INT;
colset Type_di2 = product STR * INT * STR;
colset Type_do = product STR * STR;
local_var a, c, d : STR;
local_var b : INT;
```

```
colset STR = string;
colset INT = int;
colset Type_di = product STR  * INT;
colset Type_do = product STR  * STR;
colset Type_msg = STR ;
local_var a, m : STR;
local_var b : INT;
```

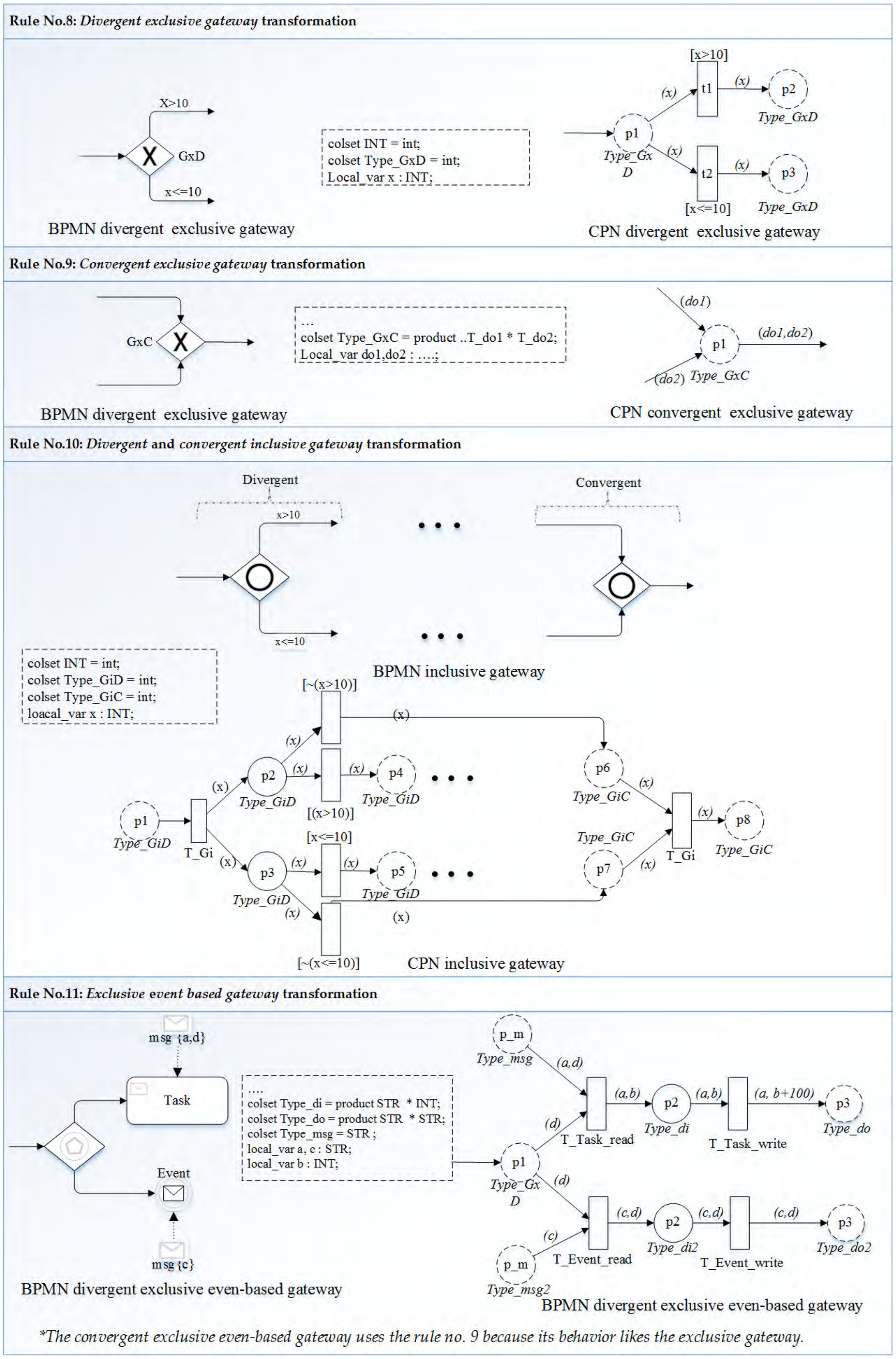
```
colset INT = int;
colset Type_GpD = INT ;
local_var d : INT;
```

```
colset Type_GpC1 = product ...;
colset Type_GpC2 = product ...;
colset Type_GpC3 = product ..T_do1 *T_do2;
local_var do1,do2 : ....;
```

**FIGURE 8.** The mapping rules to transform BPMN elements into CPN constructs (Continued).

The output arc inscription of transition *t1* is (*cust_name*, *request_no*), which means that after transition *t1* is enabled, the next transition *t1* produces new token colors by passing the token colors of variables *cust_name* and *request_amount* to the output place *p3*.

- **Functional passing**: The advantage of functional passing is that the transition can produce the output token


Rule No.8: *Divergent exclusive gateway* transformation
X>10
GxD
x<=10
BPMN divergent exclusive gateway
[x>10]
(x)
t1
p2
Type_GxD
p1
Type_Gx D
t2
p3
[x<=10]
CPN divergent exclusive gateway
Rule No.9: *Convergent exclusive gateway* transformation
GxC
BPMN divergent exclusive gateway
(do1)
(do1,do2)
(do2)
Type_GxC
CPN convergent exclusive gateway
Rule No.10: *Divergent* and *convergent inclusive gateway* transformation
Divergent
Convergent
x>10
x<=10
BPMN inclusive gateway
[~(x>10)]
[(x>10)]
[x<=10]
[~(x<=10)]
p1
Type_GiD
T_Gi
p2
p3
p4
p5
p6
Type_GiC
p7
p8
CPN inclusive gateway
Rule No.11: *Exclusive event based gateway* transformation
msg {a,d}
Task
Event
msg {c}
BPMN divergent exclusive even-based gateway
p_m
Type_msg
(a,d)
(a,b)
(a, b+100)
(d)
T_Task_read
T_Task_write
Type_di
Type_do
(c,d)
(c)
T_Event_read
T_Event_write
Type_di2
Type_do2
Type_msg2
BPMN divergent exclusive even-based gateway
*The convergent exclusive even-based gateway uses the rule no. 9 because its behavior likes the exclusive gateway.

```
colset INT = int;
colset Type_GxD = int;
Local_var x : INT;
```

```
…
colset Type_GxC = product ..T_do1 * T_do2;
Local_var do1,do2 : ….;
```

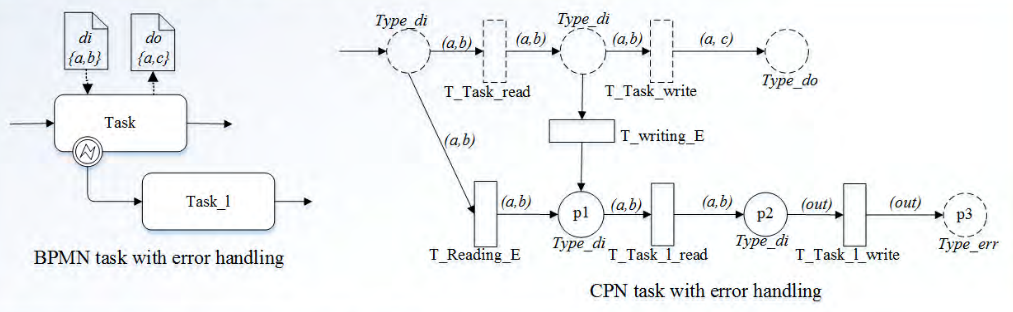
```
colset INT = int;
colset Type_GiD = int;
colset Type_GiC = int;
loacal_var x : INT;
```

```
….
colset Type_di = product STR  * INT;
colset Type_do = product STR  * STR;
colset Type_msg = STR ;
local_var a, c : STR;
local_var b : INT;
```

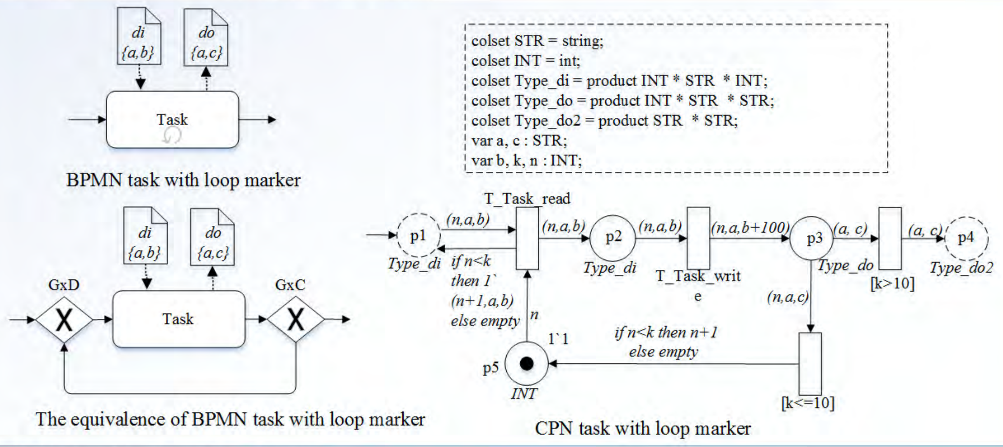
**FIGURE 9. The mapping rules to transform BPMN elements into CPN constructs (Continued).**

colors using the capabilities of functional programming. The operators and operands are used in mathematical form to manipulate the global variables and produce the token colors. Figure 13(b) illustrates the functional passing of the output arc inscription. The transition *t1* produces the token colors to place *p3* using function

**Rule No.12:** ***Exception handling*** **transformation**



**Rule No.13: Transformation rules of the** ***task with loop marker.***


```
colset STR = string;
colset INT = int;
colset Type_di = product INT * STR  * INT;
colset Type_do = product INT * STR  * STR;
colset Type_do2 = product STR  * STR;
var a, c : STR;
var b, k, n : INT;
```
di {a,b}
do {a,c}
Task
BPMN task with loop marker
GxD
GxC
The equivalence of BPMN task with loop marker
T_Task_read
p1
Type_di
(n,a,b)
if n<k then 1` (n+1,a,b) else empty
p2
Type_di
T_Task_write
(n,a,b+100)
p3
Type_do
(a, c)
[k>10]
p4
Type_do2
(n,a,c)
n
1`1
p5
INT
if n<k then n+1 else empty
[k<=10]
CPN task with loop marker

**Rule No.14:** ***Connecting elements*** **transformation**

The *Sequence flow* outgoing from the inclusive or exclusive gateway is mapped into a transition while the sequence flows connected between elements are absorbed by the concatenation step.

The *Message flow* is mapped into a transition.

**Rule No.15:** ***Sub-process*** **transformation**

The *Sub-Process* is partitioned into BPMN partitions by the model partitioning step already. All BPMN elements in a partition are transformed using the transformation rules no. 1 to 13.

**Rule No.16: CPN constructs concatenation**

The CPN constructs will be concatenated on the dashed line places.

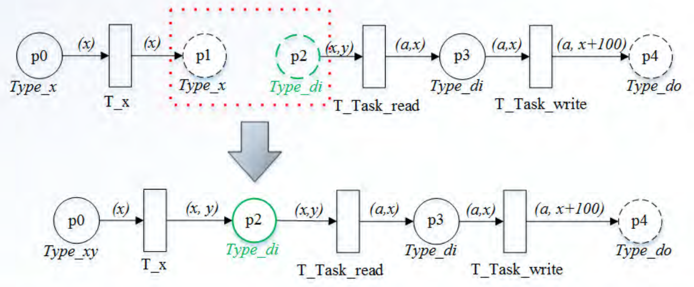


**FIGURE 10.** The mapping rules to transform BPMN elements into CPN constructs (Continued).

*fn_produce*, whose input parameters are the list of variables on the input arcs of the transition *t1*. The operations of function *fn_produce* are the global value manipulation and token color construction in the statement "*return*".

## C. STATE SPACE GENERATION AND STATE SPACE EXPLORATION

To validate the CPN models using the state space analysis, the CPN models obtained from the transformation stage are

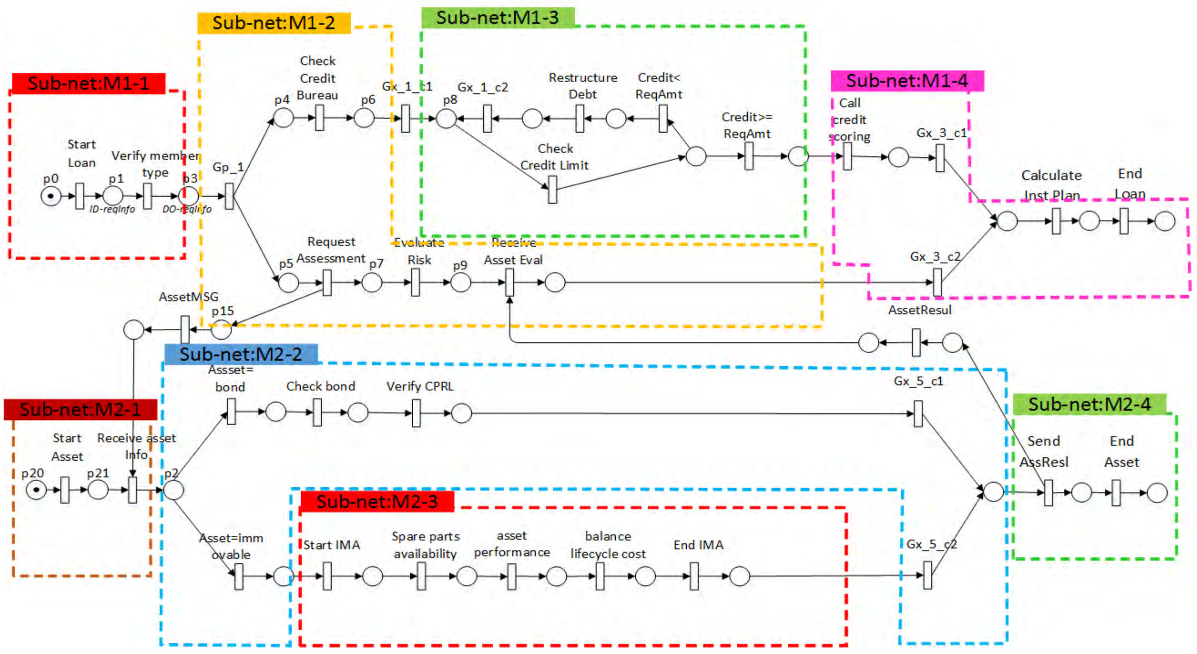


FIGURE 11. CPN model of the BPMN model in Figure 6 transformed using the BPMN transformation rules.

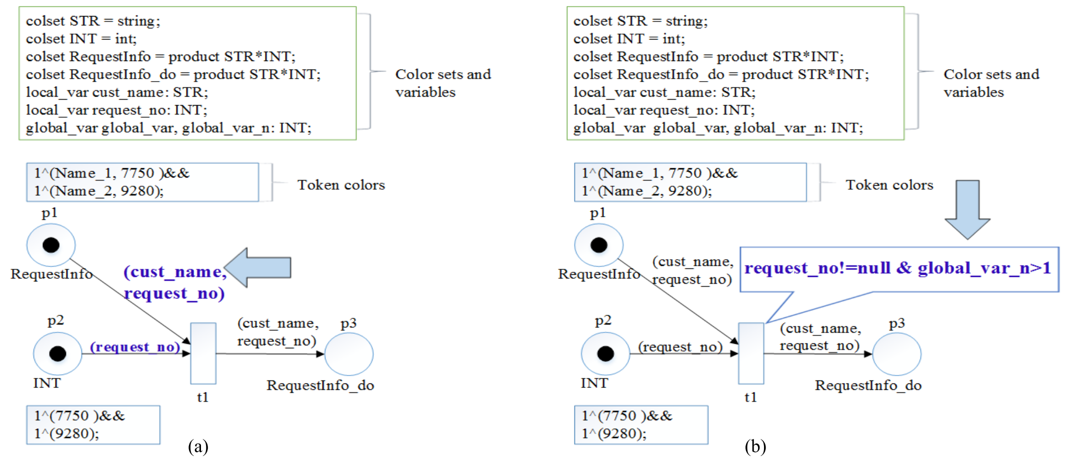
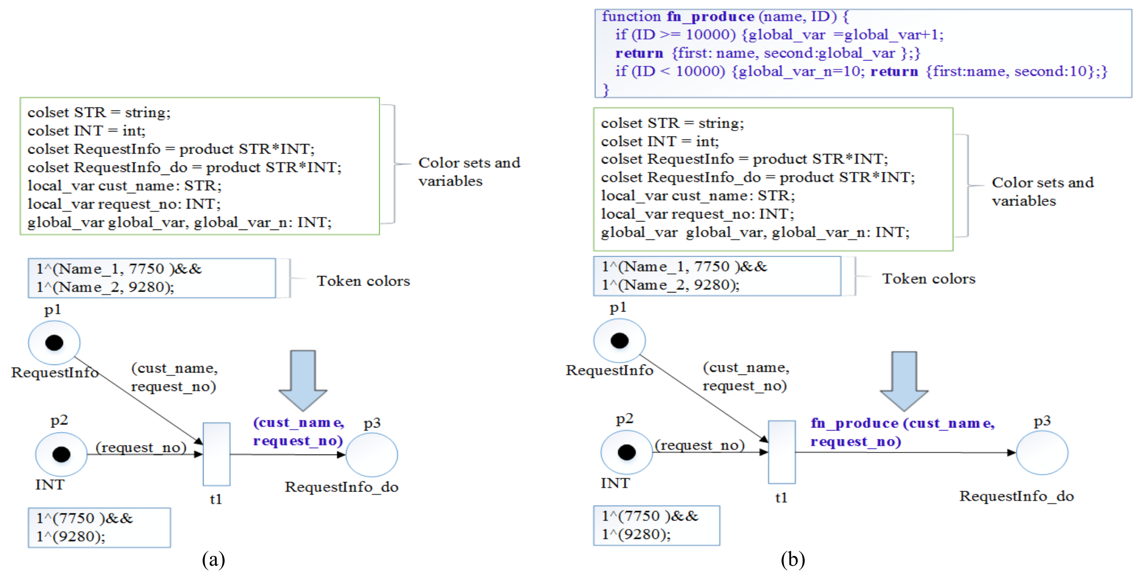


FIGURE 12. Examples of input arc inscriptions and predicate inscription of the CPN transition. (a) Input arc inscription. (b) Predicted inscription of CPN transition.

used to construct the reachable graph [31] called the state space. The reachable graph represents the possible states of a CPN model based on a given initial marking. The reachable graph is generated from the CPN model via the state space generator tools. The existing state space generators have limitations in terms of the size of the state space and programming language used for the inscription expression. We propose a state space generator that can store the state space data in secondary storage and support model inscriptions that are written in Java programming language.

According to Definition 2, each node in the reachable graph represents a reachable marking. The edges connecting nodes represent the transition firing information, i.e., binding elements. Nodes and edges are retrieved and interpreted to account for the state transition. We apply the reachability graph construction algorithm of [4] and extend part of the multi-binding element construction for our state space generator. Because of the elaborated algorithm, this paper demonstrates merely the core algorithm of state space construction. The algorithm for computing the reachable graph is illustrated in Algorithm 2.

In Algorithm 2, line 3 is the initial marking, which acts as the initial state and is determined to be the first marking of a set of *work*. Work is the controller of the marking queue for producing the successor markings and binding elements. The process continues as long as the work is not empty. M1 is the



```
colset STR = string;
colset INT = int;
colset RequestInfo = product STR*INT;
colset RequestInfo_do = product STR*INT;
local_var cust_name: STR;
local_var request_no: INT;
global_var global_var, global_var_n: INT;
```

```
1^(Name_1, 7750 )&&
1^(Name_2, 9280);
```

```
1^(7750 )&&
1^(9280);
```

```
function fn_produce (name, ID) {
  if (ID >= 10000) {global_var =global_var+1;
  return {first: name, second:global_var };}
  if (ID < 10000) {global_var_n=10; return {first:name, second:10};}
}
```

**FIGURE 13.** Examples of output arc inscriptions. (a) Output arc inscription using direct passing. (b) Output arc inscription using functional passing.

**Algorithm 2** The State Space Generation

```
01 Require: A CPN model $CPN = (PP, TT, AA, \Sigma\ VV, fCC, fGG, fEA, fII)$
02 Ensure: $CPN$ is the finite reachable marking
03 $\{M_0\} = fII([26])$
04 $work := \{M_0\}$
05 While $work \neq \emptyset$
06     $M_1$ := select a marking from $work$ by FIFO ordering
07     $work := work \setminus \{M_1\}$
08     $singleBinding.=$empty
09         For enabled($M_1$): $trans$
10           If fire($M_1$, $trans$) then
11               $M_2 :=$ fire($M_1$, $trans$)
12               b := binding elements of transition
                 firing $\xrightarrow{b}$
13             if $M_2$ dose not exists in nodes then
14               $nodes := nodes \cup \{M_2\}$
15               $work = work \cup \{M_2\}$
16             end if
17             $edges := edge \cup \{M_1 \xrightarrow{b} M_2\}$
18               $singleBinding := singleBinding \cup \{b\}$
19         End if
20         Next
21     $MultiBinding(singleBindjng)$:
22 End while
```

current marking chosen from the marking queue by first-in-first-out ordering. Lines 9 to 20 are loops that depend on the number of enabled transitions for the current markings. The enabled transitions produce successor marking M2, aned the binding element can be expressed by the symbol $\longrightarrow$, which represents the relationship between markings M1 and M2. Line 13 checks whether marking M2 exists in a set of *nodes*.

If the marking does not exist in the nodes, the new marking M2 is added to the set of *nodes* and *work*. For every transition firing, the transition firing information on the edge that represents a single binding element is recorded in the set of *edges*. Next, the multi-binding elements are calculated from the set of single binding elements, and the results of the multi-binding elements are added to the set of *nodes*, *work* and *edges*.

As the result of successor marking construction, the binding element occurs when the transition is enabled, and the transition fires the tokens to the input places. The successor marking construction consist of three key components.

**1) Binding the token colors.**

The token colors are assigned to the variables on the input arc inscriptions if the variable name is matched. The color assignments are recorded as the colored binding information.

**2) Computing the enabled transition.**

The enabled transition relies on the CPN constructs and the guard condition. In the case that the transition results in the guard condition, the variables on the guard conditions are replaced by the values in the colored binding information if the variable name matches. If the guard condition evaluation is *true*, the transition is enabled. In other cases, the enabled transition depends on the CPN constructs.

**3) Producing the new token for the output places.**

When the transition is enabled, the transition fires the new token to its output places by means of the output arc inscriptions. The colored binding information and transition name are collected as the binding elements.

For instance, we begin the processing set with the enabled transition (t) and then assign the token colors residing in the input place to the variables (V) on an input arc inscription of transition *t*. The variables related to the transition can be denoted by V(t). The colored binding information is recorded as the pairwise item (variable: value). After binding all the

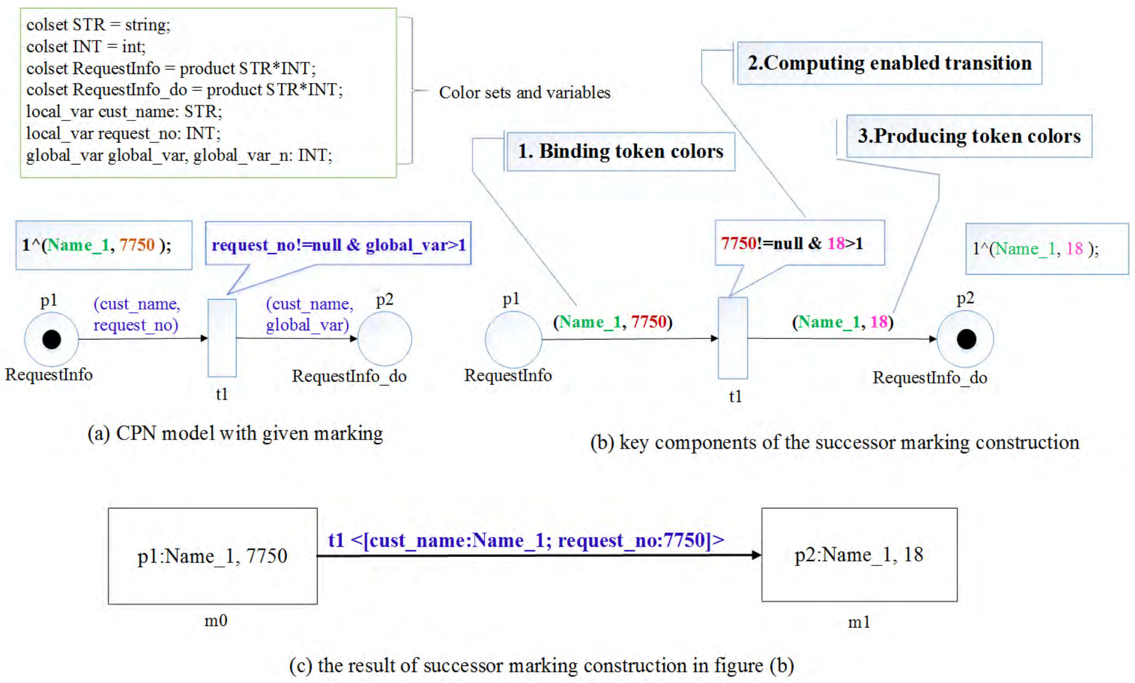

colset STR = string;
colset INT = int;
colset RequestInfo = product STR*INT;
colset RequestInfo_do = product STR*INT;
local_var cust_name: STR;
local_var request_no: INT;
global_var global_var, global_var_n: INT;
Color sets and variables
1^(Name_1, 7750 );
request_no!=null & global_var>1
p1
(cust_name, request_no)
(cust_name, global_var)
p2
RequestInfo
RequestInfo_do
t1
(a) CPN model with given marking
1. Binding token colors
2.Computing enabled transition
3.Producing token colors
7750!=null & 18>1
1^(Name_1, 18 );
(Name_1, 7750)
(Name_1, 18)
(b) key components of the successor marking construction
p1:Name_1, 7750
t1 <[cust_name:Name_1; request_no:7750]>
p2:Name_1, 18
m0
m1
(c) the result of successor marking construction in figure (b)

**FIGURE 14.** **Example of the successor marking computation. (a) CPN model given marking. (b) Key components of the successor construction. (c) The result of successor marking construction in figure (b).**

token colors, we obtain a set of binding information as $< v_1 = tc_1, \ldots, v_n = tc_n >$, where $v_i \in$ V(t), $tc_i$ is the token color, $i = 1, 2, 3 \ldots$. If the transition does not have a guard condition and its construct satisfies the enabling rule, the transition is enabled. By contrast, if the transition has a guard condition, the variable on the guard condition is replaced by the token color if the variable matches variable V(t) or a global variable with the same name; then, the guard condition is evaluated. If the result of the guard evaluation is true, the transition is enabled. The next step is the processing of the successor markings. The token colors passed through the variables V(t), the global variables and constant values can be used to produce new token colors for the output place of the transition. When the new token is added to the output place, the new marking is recorded as the successor marking. Figure 14 shows the successor marking process.

The marking $m_0$ is the current state of the system, and the token color of place *p1* is the token (Name_1, 7750). To bind the token colors, the matching of token color (Name_1, 7750) to the input arc inscription (*cust_name, request_no*) proceeds via the variable *cust_name* = "Name_1" and *request_no* = 7750. The guard condition on transition t1 is expressed as "request_no!=null & global_var>1", and at this moment, the variable *request_no* is 7750 and global variable *global_var* is 18. Thus, variables *request_no* and *global_var* on the guard condition are replaced by "7750" and 18, respectively. When the result of the guard condition evaluation is true, transition *t1* is enabled, which may produce token colors for place *p2*. Next, the variables related to transition *t1* are placed into the output arc inscription as the new added token (Name_1, 18) on place *p2*.

The state space construction of the hierarchical CPN model is similar to the processing of a non-hierarchical CPN model. The difference is that some sub-nets of the CPN model may be determined as black-box processes in which the sub-net is reduced to the substitution transition. However, we must execute all parts of the sub-net because the neighboring sub-nets always require the output data from the substitution transition to construct their states. In the state space reduction process, the dead marking states of the substitution transition are ignored.

Figure 15(a) shows an excerpt of the non-hierarchical CPN model in Figure 11. *sub-net:M1-1*, *sub-net:M1-2* and *sub-net:M1-3* are selected for the CPN model for the state space generation. Figure 15(b) shows a hierarchical CPN model where *sub-net M1-2* is the substitution transition. The dual line place *IP1* and *IP2* are the input ports of substitution transition *sub-net:M1-2*, whereas places *OP1*, *OP2* and *OP3* are the output ports.

Figure 16(a) illustrates the intuition behind the state space generation of the CPN model in Figure 15(a). To simplify the reachable graph, we have omitted the marking information and binding elements. The tree shows the implicit states produced from *sub-net:M1-1* and *sub-net:M1-2*. Node $m_0$ denotes the initial state, which is the given marking on place *p0* of *sub-net:M1-1*. The successor states are $m_1$ and $m_2$. The marking $m_2$ represents the explicit state with one token in place *p3* with token color 1^(Name_1, 7750, memOK). Because place *p3* is shared between *sub-net M1-1* and

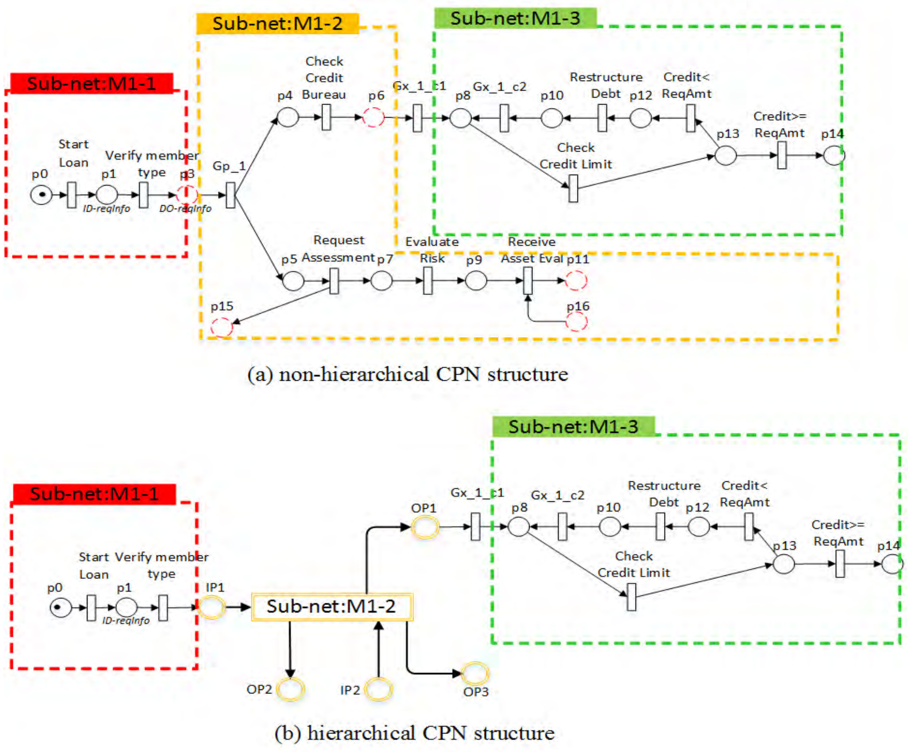


**FIGURE 15.** Examples of CPN model structures for state space generation. (a) Non-hierarchical CPN structure. (b) Hierarchical CPN structure.

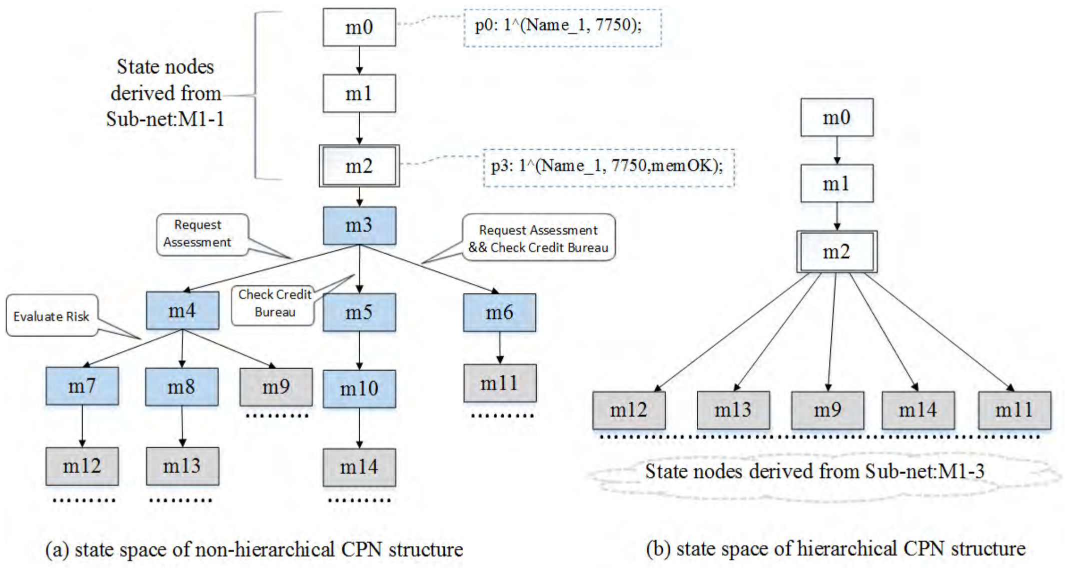

State nodes derived from Sub-net:M1-1
p0: 1^(Name_1, 7750);
p3: 1^(Name_1, 7750,memOK);
Request Assessment
Request Assessment && Check Credit Bureau
Check Credit Bureau
Evaluate Risk
State nodes derived from Sub-net:M1-3
(a) state space of non-hierarchical CPN structure
(b) state space of hierarchical CPN structure

**FIGURE 16.** The excerpt reachable graph of the CPN model in Figure 15. (a) State space of non-hierarchical CPN structure. (b) State space of hierarchical CPN structure.

*sub-net M1-2*, state $m_2$ is the latest successor marking of *sub-net M1-1* and is also the initial marking of *sub-net:M1-2*. The markings $m_3$ to $m_{14}$ are the successor markings derived from *Sub-net:M1-2*. The gray nodes are the *dead markings* of *sub-net:M1-2*, which represent the markings that cannot enable the transitions of *sub-net:M1-2*. The dead markings are an important factor in the state space reduction for hierarchical verification. They represent the possible outputs of *sub-net:M1-2* and are used to compute the successor markings of the neighboring *sub-net:M1-3*. Figure 17 illustrates the binding elements of marking $m_{11}$, which is composed of three binding elements: *e1*, *e2* and *e3* (with execution

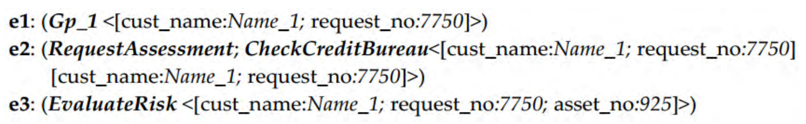

**e1**: (***Gp_1*** <[cust_name:*Name_1*; request_no:*7750*]>)
**e2**: (***RequestAssessment***; ***CheckCreditBureau***<[cust_name:*Name_1*; request_no:*7750*]
[cust_name:*Name_1*; request_no:*7750*]>)
**e3**: (***EvaluateRisk*** <[cust_name:*Name_1*; request_no:*7750*; asset_no:*925*]>)

**FIGURE 17. Binding elements of marking $m_{11}$ in Figure 16(a).**

**TABLE 3. Commands for state space exploration.**

| No | Objectives or Behaviors | Pattern Names | Equivalent CTL |
|---|---|---|---|
| 1 | Find unreachable transitions. | Basic commands | - |
| 2 | Find dead markings of a model. | Basic commands | - |
| 3 | Print marking information. | Basic commands | - |
| 4 | Print counter examples. | Basic commands | - |
| 5 | It is possible for state $\rho$ to occur. | Occurrence pattern | EF($\rho$ ) |
| 6 | It is not possible for state $\rho$ to occur. | Exclusion pattern | -EF($\rho$ ) |
| 7 | It is possible for state $\rho$ to occur in next. | Occurrence pattern | EX($\rho$ ) |
| 8 | It is always for state $\rho$ to occur in next. | Occurrence pattern | AX($\rho$) |
| 9 | If state $\rho$ occurs, then it is possibly followed by state $\varphi$. | Consequence pattern | AG ($\rho \longrightarrow$EF ( $\varphi$)) |
| 10 | If state $\rho$ occurs, then it is necessarily followed by state $\varphi$ | Consequence pattern | AG ($\rho \longrightarrow$AF ( $\varphi$)) |
| 11 | State $\rho$ is reached and is possibly preceded at some time by state $\varphi$ | Sequence pattern | EF($\rho$ and EF($\varphi$ )) |
| 12 | State $\rho$ can persist indefinitely. | Invariance pattern | AG($\rho$) |
| 13 | State $\rho$ must persist indefinitely. | Invariance pattern | EG($\rho$) |

An invariant pattern is a common query and represents useful safety properties while an occurrence pattern represents the liveness properties.

path $m_2 \longrightarrow m_3 \longrightarrow m_6 \longrightarrow m_{11}$). The binding element *e1* is the firing information of transition *Gp_1*. Next, the transitions *Request Assessment* and *Check Credit Bureau* fire tokens simultaneously; these tokens are binding element *e2*. The explicit state of *e2* is the tokens on places *p6*, *p7* and *p15* (represented by marking $m_6$); however, marking $m_6$ is not a dead marking because it can enable the transition *Evaluate risk*. Therefore, marking $m_6$ can reach marking $m_{11}$, which is the implicit state firing of transition *EvaluateRisk*.

Figure 16(b) illustrates the state space of the hierarchical CPN model in Figure 15(b). The state space size of the hierarchical CPN model is smaller than that of the hierarchical CPN model in Figure 16(a). This is an application of the sweep-line method [15]. The state space construction of a substitution transition is focused on the initial and end states of a sub-net, which results in a smaller state space. Considering the dead marking *sub-net:M1-2*, we keep only the dead markings. The transitive states (blue nodes) are ignored in the state space storing process. Next, the dead markings of *sub-net:M1-2* are taken as the initial markings of *sub-net:M1-3*. When processing the successor markings of *sub-net:M1-2*, the markings are computed based on the assumption that *sub-net:M1-2* works like an atomic process. Therefore, the transitions of the neighboring sub-nets (*sub-net:M1-1* and *sub-net:M1-3*) cannot be enabled at this moment (in the scenario where there are initial tokens distributed at more than one place and in different sub-nets).

In the preceding paragraphs, we have conducted the state space generation. After the state space has been generated, an important part of BPMN design model verification is the temporal logic formulas interpreted over the state spaces. We design the functional commands for exploring the state information and transition information. For practical use, we provide the basic commands and the equivalent temporal logic commands. These commands are detailed in Table 3.

### D. IMPLEMENTATION

To illustrate the proposed techniques, we have implemented the BPMN design model verification framework as a Java application called the CP4BPMN tool. Screenshots of the tool are shown in Figures 18, 19 and 20. The application comprises five main modules.

#### 1) THE MODEL VIEW CONTROLLER

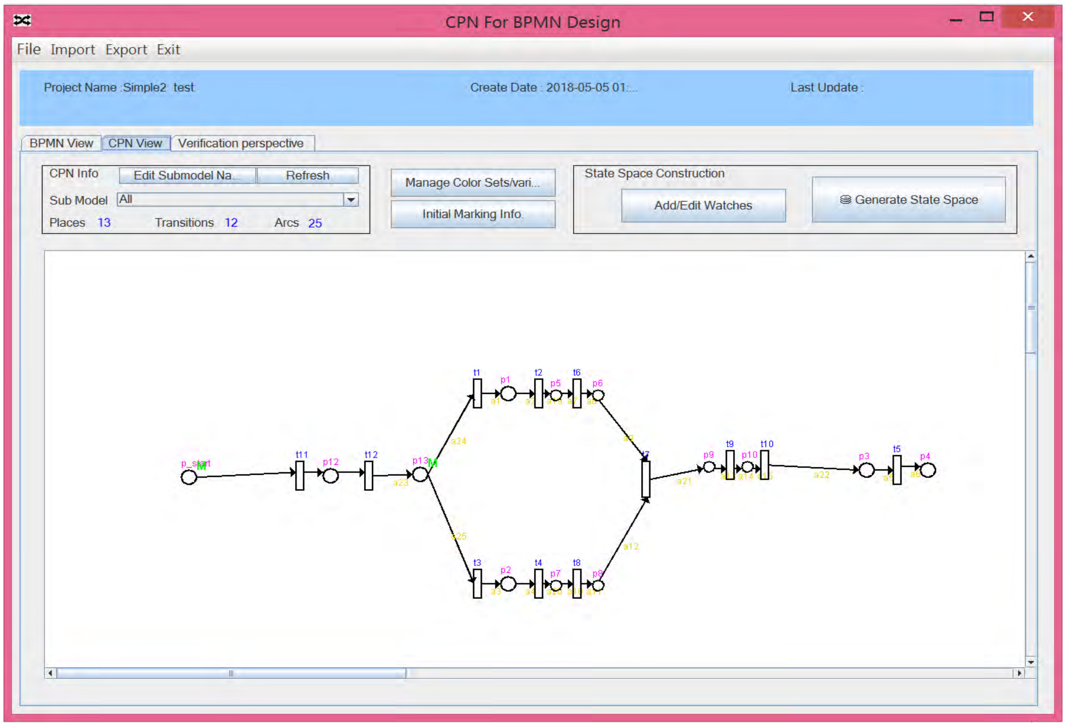
This functionality is used for visualizing the input BPMN model and the obtained CPN models, parameterizing the CPN model and refining the CPN model structure.

#### 2) THE BPMN MODEL PARTITIONER

When the BPMN design model and item definition data files are imported into the system, the BPMN elements are extracted using the XML parser. Next, the attributes of the BPMN elements are stored in the model repository for transforming and debugging purposes. Then, the BPMN model partitioner classifies and labels the BPMN elements with the partition number. The gateway pairwise analysis and weight-based consideration are back-end process of BPMN partitioning.

#### 3) THE BPMN TRANSFORMER

The responsibility of the BPMN transformer is to transform the BPMN elements into CPN constructs. The CPN constructs are composed of the structural elements and the CPN inscriptions. The structural transformation of a BPMN



Project Name :Simple2 test

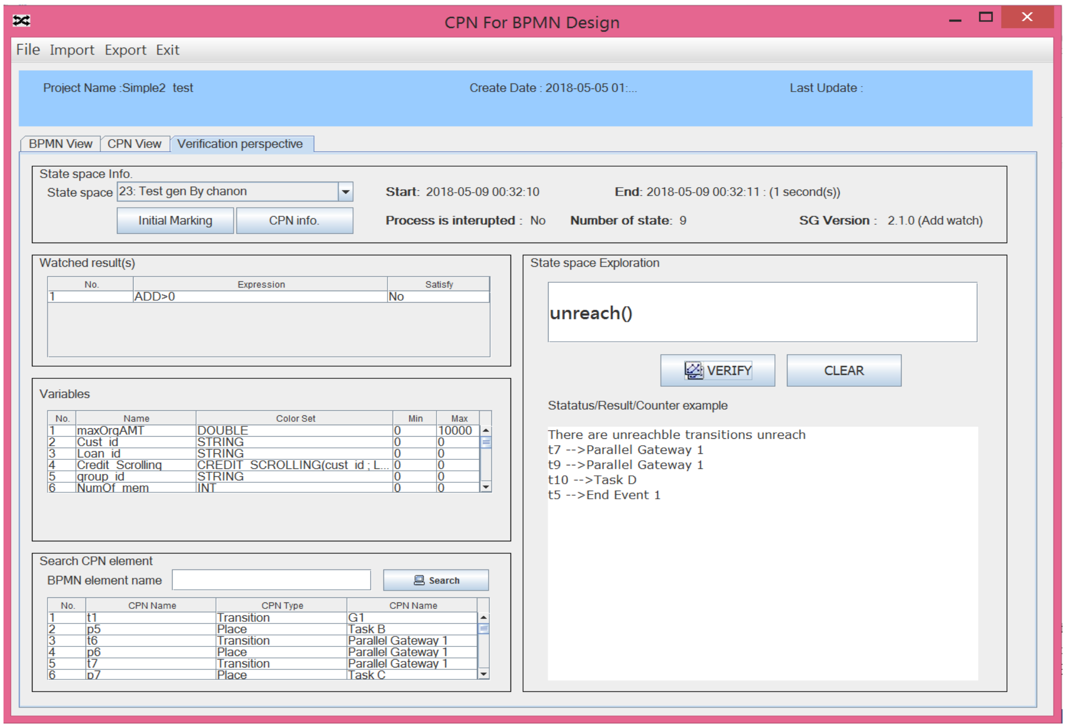
Create Date : 2018-05-05 01:...

Last Update :

Sub Model All

Places 13 Transitions 12 Arcs 25

FIGURE 18. Screenshot of the CPN model refinement of the CP4BPMN tool.



Project Name :Simple2 test

Create Date : 2018-05-05 01:...

Last Update :

State space Info.

State space 23: Test gen By chanon

**Start**: 2018-05-09 00:32:10

**End**: 2018-05-09 00:32:11 : (1 second(s))

**Process is interupted** : No

**Number of state**: 9

**SG Version** : 2.1.0 (Add watch)

Watched result(s)

| No. | Expression | Satisfy |
|---|---|---|
| 1 | ADD>0 | No |

Variables

| No. | Name | Color Set | Min | Max |
|---|---|---|---|---|
| 1 | maxOrgAMT | DOUBLE | 0 | 10000 |
| 2 | Cust_id | STRING | 0 | 0 |
| 3 | Loan_id | STRING | 0 | 0 |
| 4 | Credit_Scrolling | CREDIT_SCROLLING(cust_id ; L... | 0 | 0 |
| 5 | group_id | STRING | 0 | 0 |
| 6 | NumOf_mem | INT | 0 | 0 |

Search CPN element

BPMN element name

| No. | CPN Name | CPN Type | CPN Name |
|---|---|---|---|
| 1 | t1 | Transition | G1 |
| 2 | p5 | Place | Task B |
| 3 | t6 | Transition | Parallel Gateway 1 |
| 4 | p6 | Place | Parallel Gateway 1 |
| 5 | t7 | Transition | Parallel Gateway 1 |
| 6 | p7 | Place | Task C |

State space Exploration

```
unreach()
```

Statatus/Result/Counter example

```
There are unreachble transitions unreach
t7 -->Parallel Gateway 1
t9 -->Parallel Gateway 1
t10 -->Task D
t5 -->End Event 1
```

FIGURE 19. Screenshot of the state space exploration of the CP4BPMN tool.

element into a CPN construct is a one-to-one mapping, and the CPN inscriptions are directly mimicked from the inscriptions of the BPMN model. To check the completeness of the transformation, the system includes basic criteria to check all the CPN constructs, and a Java syntax checker verify whether the inscriptions conform to Java syntax.

**TABLE 4. Transformation result of applying the CPN4BPMN to the existing models.**

| Mortgage loan system of a commercial bank. | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Cases | BPMN elements | | | | | | | CPN elements | | |
| | Pl. | Tk&Ev. | Gw. | Sub-p. | Msg. | Var. | Col. | Sub-n. | Plas. | Trans. |
| 1 | 0 | 15 | 4 | 0 | 0 | 18 | 8 | 3 | 17 | 14 |
| 2 | 2 | 22 | 6 | 1 | 2 | 21 | 16 | 8 | 37 | 35 |
| 3 | 3 | 37 | 13 | 2 | 5 | 29 | 31 | 12 | 195 | 184 |
| Inventory management system of a retail department store. | | | | | | | | | | |
| Cases | BPMN elements | | | | | | | CPN elements | | |
| | Pl. | Tk&Ev. | Gw. | Sub-p. | Msg. | Var. | Col. | Sub-n. | Plas. | Trans. |
| 1 | 0 | 11 | 4 | 1 | 0 | 15 | 19 | 4 | 24 | 23 |
| 2 | 2 | 26 | 8 | 1 | 3 | 21 | 23 | 7 | 144 | 109 |
| 3 | 3 | 34 | 12 | 3 | 5 | 28 | 31 | 9 | 163 | 152 |

Pl.= Pool, 0 means that the input BPMN design model is the process diagram; otherwise, it is the collaboration diagram; Tk&Ev. = Task and event; Gw. =Gateways; Sub-p. = Sub-processes; Msg. =Message flows; Var. =Variables; Col. =Color sets; Sub-n. = CPN sub-nets; Trans. =Transition.

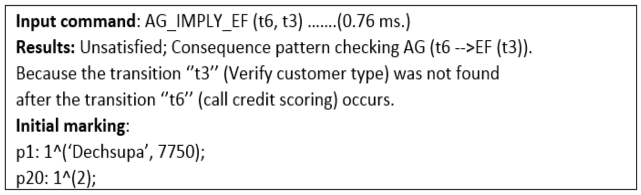

```
Input command: AG_IMPLY_EF (t6, t3) .......(0.76 ms.)
Results: Unsatisfied; Consequence pattern checking AG (t6 -->EF (t3)).
Because the transition "t3" (Verify customer type) was not found
after the transition "t6" (call credit scoring) occurs.
Initial marking:
p1: 1^('Dechsupa', 7750);
p20: 1^(2);
```

**FIGURE 20. Example report of the state space exploration.**

### 4) THE STATE SPACE GENERATOR

The verification using the state space may consider specific situations. The CPN model can have an enormous or even infinite number of reachable markings. To address the size of the state space, prior to the state space generation, the designers must refine the CPN model and determine the initial marking. The state space generator will check the correctness of the CPN model. If the CPN model is successful, the state space produces the reachable graph based on the given initial marking. During the state space generation, if the CPN model violates the invariant properties, a counterexample is shown, and the state space construction is terminated. The reachable graph, including the state space report, is stored in the state space repository designed using a combination of XML standard and MS SQL Server.

### 5) THE STATE SPACE EXPLORER

The basic commands and temporal logic for searching the model properties are implemented as functional queries. For example, the command UNREACH($\rho$) is used to find the unreachable transitions or unreachable places. The command EXIST_NEXT($\rho$) is used for finding whether it is possible for state $\rho$ to occur next, where $\rho$ is the name of a transition or place.

## V. EXPERIMENT

We demonstrate our techniques and validate the CP4BPMN tool by applying to the existing models. The BPMN models of the Mortgage Loan System (MLS) and Inventory Management System (IMS) are used as our test sets. The BPMN design models of both systems were designed using the *Eclipse BPMN2 Modeler* [32]. The models have been tested with different sizes, types, and hierarchical structures. The elements of the input BPMN model and the result of transforming the BPMN model into a CPN model are shown in Table 4.

As the statistical report shows in Table 4, BPMN partitioning is dependent on the weight configuration. The weight per element is 3, and the maximum total weight per partition is 15. In our experiments, we separate the testing into two phases.

In the first phase, the verification of the transformation results is focused on the completeness of the CPN models. We define the basic criteria for checking the obtained CPN models as follows.

1) Each CPN element can be traced back to the sourced BPMN element.
2) All CPN elements in the same sub-net must have a path to each other.
3) The inscription in the CPN model must conform to that of the BPMN model.
4) The CPN sub-net and the BPMN partition must be consistent.

In the experiments, the CP4BPMN tool can transform the BPMN models into CPN models correctly. The partitioning technique and framework not only reduce the mistakes in the CPN model but also reduce the time consumption.

In the second phase, we validate the CPN models using the state space analysis method. Our test plan is that the composition sub-nets are necessarily re-arranged to be a hierarchical model. The substituted transition is used to hide the internal implementation of the sub-nets. While the flat verification of that all sub-nets intends to show the state space without hiding their internal implementations. Next, we compare the state space construction, exploration and the results of the flat structure to those of hierarchical structure. Intuitively, the time consuming and number of state space of the flat structure is much more than those of the hierarchical structure. However, the queries used for the state spaces exploration should be working at the state space of both the flat model and hierarchical model. We will explore and analyze the state space using the functional queries of CP4BPMN tool

**TABLE 5. State space statistics of the CPN model in Figure 11.**

| Case No. | Loan app. | Sub-nets Used | Substitution transitions | Nodes | Edges | CPU Min (s) |
|---|---|---|---|---|---|---|
| 1 | 1 | all | - | 8,514 | 8,942 | 2.57 |
| 2 | 1 | all | M1-2, M2-2 | 5,823 | 5,158 | 2.12 |
| 3 | 1 | M1-1, M1-2, M1-3, M1-4 | M1-2,M1-3 | 1,311 | 1,731 | 0.54 |
| 4 | 1 | M2-1, M2-2, M2-3,M2-4 | M2-1, M2-2, | 514 | 557 | 0.12 |
| 5 | 1 | M1-1, M1-2, M1-3, M2-1, M2-2, M2-3 | M1-2, M1-3, M2-2 | 4,737 | 5,270 | 1.09 |
| 6 | 2 | all | - | 24,512 | 29,980 | 4.11 |
| 7 | 2 | all | M1-2, M2-2 | 14,578 | 17,345 | 3.19 |
| 8 | 3 | all | - | 79,089 | 84,349 | 6.56 |
| 9 | 3 | M1-1, M1-2, M1-3, M2-1, M2-2, M2-3 | M1-2, M1-3, M2-2 | 51,267 | 5,370 | 5.79 |

The column Loan app. is the number of initial tokens for the concurrent execution verification, which is determined for place p0. Sub-nets are the names of the CPN sub-nets chosen for the state space construction. The substitution transitions are the CPN sub-net names determined to be substitution transitions, where "-" indicates that the state space is generated from a non-hierarchical CPN model.

**TABLE 6. Excerpt results of the CPN model verification with specific requirements.**

| No. | Requirements | Exploration queries | Results |
|---|---|---|---|
| 1 | The system must verify the type of customer; then, the processes of the credit bureau check and request asset assessment can occur simultaneously. | EF_AND('t2', AND('t3', 't4')) | Satisfied |
| 2 | All loan applications have to be verified by three processes: the checking of the credit bureau, credit limit and risk evaluation. | EF_IMPLY_EF('t2',AND('t8', 't6')) | Satisfied |
| 3 | In the case that the customer provides immovable assets as the mortgaged property, such assets must pass the balance lifecycle cost evaluation. | EF_IMPLY_EF('t22', 't27') | Satisfied |
| 4 | In the case that the customer provides bond assets as the mortgaged property, they must be zero coupon bonds and their bond ratings are calculated by the web service of the balance lifecycle cost. | EF_IMPLY_EF('t21', AND('t29', 't31')) | Satisfied |
| 5 | The assets ratio and credit limit are mandatory data for the credit scoring evaluation. | EG_AND('t15', 't17') | Unsatisfied |

The 't*' in the query is the CPN transition name representing the BPMN task.

to check deadlock, unreachable task, invariant, soundness and possibly other specific requirements.

The example of test plan: The sub-nets of a CPN model are selected to generate a state space, and the sub-nets are arranged in different hierarchical structures. For example, we choose *sub-net:M1-1, sub-net:M1-2* and *sub-net:M1-3* from the eight sub-nets in Figure 11. Next, we set one or all of them to be substituted transitions for the hierarchical verification. Since the abstraction level between the BPMN model and CPN model may be different (most CPN models have lower abstraction levels), we have to refine the CPN model by adding the global variables and the predicate for checking the invariant properties, including revising some of the inscriptions for completeness. Then, the initial markings are determined, and the state spaces are constructed using our state space generator. Table 5 shows the state space statistics of the CPN model in Figure 11.

The standard report of state space generation is the information regarding the standard properties, size of the state space, time consumed, and dead markings. We investigate the state space analysis of the refined CPN model in both non-hierarchical and hierarchical CPN structures. The CPN models are verified using the bottom-up approach. The sub-net identified as the substitution transition is verified before repeating the large-scale process simultaneously.

Generally, the termination behavior of a CPN model must conform to the BPMN design with *end events*. In short, the end events of the BPMN model must be reached from the *start events*. Failure of the state space to satisfy this condition indicates that the BPMN design model may have unreachable tasks or deadlocks.

We consider the size of the state space and the CPN model refinement. The size of a state space depends on the number of sub-nets chosen, the substitution transition determination and the number of initial tokens. In Table 5, case number 1 is tests the large-scale model based on a single loan application, where all the sub-nets derived from the partitioning step are combined into an abstract model with a flat structure for state space construction.

We verify the CPN model by gradual refinement in the difference of sub-nets and hierarchical structures. Hierarchical structure refinement usually leads to an easier representation and smaller state space. Sub-nets can be selected from different participants (pools) or from the same participant when checking the communication between organizations. Finally, the initial marking of place *p0* is determined by a single token, and the number of tokens is increased to validate the concurrent execution. The results shows that *sub-net:M1-2* and *sub-net:M1-3* are complicated processes. Whenever they are identified as the substituted transitions, the size of the state space and time consumed are smaller than those of a flat model.

We also investigate whether the CPN model has specific properties. The specific properties come from the user requirements and describe the business needs. For the properties verification, the queries are determined to traverse the state space. The goal states may possibly be in different sub-nets. For example, the query of requirement No.1 in Table 6 is *EF_AND('t2', AND('t3', 't4'))*, which state t2 "verify the type of customer" is in *sub-net:M1-2* while state t3 "check credit bureau" and state t4 "request asset assessment" are in *sub-net:M1-2*. The excerpt specific properties for the state space exploration of the mortgage loan are shown in Table 6. The mortgage loan model does not satisfy requirement number 5. We resolve this issue by redesigning the BPMN design model by changing gateway *Gx_3* to a parallel gateway and moving the task *Call credit scoring* to the position after the parallel gateway.

## VI. CONCLUSIONS AND FUTURE WORK

The verification of BPMN design models using state space analysis is related to model abstraction and state space exploration. The CPN model can have an enormous or even infinite number of reachable markings. Specifically, in large-scale BPMN model verification, the huge size and complexity of the BPMN design model may result in the state space explosion problem. To address this problem, prior to state space generation, the designers should refine the model by partitioning and rearranging the abstract model in the hierarchical structure to reduce the complexity and size of the abstract model, including the state space size. We provide CPN model abstraction and partitioning techniques, transformation rules, and an automated framework called CP4BPMN to perform BPMN design model verification. The obtained CPN models support the verification in the specific partitions, as well as in the hierarchical structures. In the verification stage, we provide a state space generator that supports the input CPN models with inscriptions written in Java programming language. The sweep-line technique is applied to reduce the state space storage. The state space explorer interprets the functional commands for state space exploration. The experiment shows that the partitioning techniques and hierarchical structure refinement reduce the size of the state space and time consumption. This technique is a viable option for large-scale BPMN design model verification for designers who are not familiar with CPN language.

Because the behavior of the complex gateway is sophisticated, the modelers should simplify the gateways to simple gateways before model transformation. Another limitation is parallel execution since synchronization may require input data from different tasks that might provide data with the same variables. Thus, the designers must refine the CPN synchronization logic to choose the correct data version. Our ongoing work is directed toward improving the performance of state space construction and implementing nested commands to explore the state space for the practical use. We plan to extend the transformation rules supporting the BPMN *transaction*, *cancellation* and *compensation handling* and to apply probability component analysis in the BPMN model partitioning process.

## REFERENCES

[1] *OMG Unified Modeling Language TM (OMG UML) Version 2.5*, document Formal/15-03-01, The Object Management Group, Needham, MA, USA, 2015.

[2] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, *Model Checking and the State Explosion Problem*. Berlin, Germany: Springer, 2012, pp. 1–30.

[3] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modelling and validation of concurrent systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, nos. 3–4, pp. 213–254, 2007.

[4] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol. 37, no. 588. Berlin, Germany: DAIMI Report Series, 2008.

[5] 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.

[6] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Formal verification of web service orchestration using colored Petri net," in *Proc. Int. Multiconf. Eng. Comput. Scientists (IAENG)*, vol. 1, 2016, pp. 398–403.

[7] N. Trčka, W. van der Aalst, and N. Sidorova, "Analyzing control-flow and data-flow in workflow processes in a unified way," Technische Univ. Eindhoven, Eindhoven, Tech. Rep., 2008, vol. 0831.

[8] E. Cardoso, K. Labunets, F. Dalpiaz, J. Mylopoulos, and P. Giorgini, "Modeling structured and unstructured processes: An empirical evaluation," in *Conceptual Modeling*. Cham, Switzerland: Springer, 2016, pp. 347–361.

[9] D. Pradubsuwun, "Hierarchical verification of WS-BPEL," in *Proc. Int. Comput. Sci. Eng. Conf. (ICSEC)*, Jul./Aug. 2014, pp. 378–382.

[10] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[11] M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic*. Hoboken, NJ, USA: Wiley, 2011.

[12] H. A. Gabbar, *Modern Formal Methods and Applications*. Berlin, Germany: Springer, 2006.

[13] C. A. Petri and W. Reisig, "Petri net," *Scholarpedia*, vol. 3, no. 4, p. 6477, 2008.

[14] S. Christensen, L. M. Kristensen, and T. Mailund, "A sweep-line method for state space exploration," in *Tools and Algorithms for the Construction and Analysis of Systems* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2001, pp. 450–464.

[15] K. Jensen, L. M. Kristensen, and T. Mailund, "The sweep-line state space exploration method," *Theor. Comput. Sci.*, vol. 429, pp. 169–179, Apr. 2012.

[16] C. Ou-Yang and Y. D. Lin, "BPMN-based business process model feasibility analysis: A Petri net approach," *Int. J. Prod. Res.*, vol. 46, no. 14, pp. 3763–3781, 2008.

[17] M. M. Ibrahim, "Formal semantics of BPMN process models using CPN," *IREIT*, vol. 5, no. 3, pp. 1–7, 2017.

[18] M. Ramadan, H. G. Elmongui, and R. Hassan, "BPMN Formalisation using coloured Petri nets," in *Proc. 2nd GSTF Annu. Int. Conf. Softw. Eng. Appl.*, 2011, pp. 1–6, doi: 10.5176/2251-2217_SEA39.

[19] R. M. Dijkman, M. Dumas, and C. Ouyang, "Formal semantics and analysis of BPMN process models using Petri nets," Queensland Univ. Tech., Brisbane, QLD, Australia, Tech. Rep., 2007.

[20] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Formal verification of complex business processes based on high-level Petri nets," *Inf. Sci.*, vols. 385–386, pp. 39–54, Apr. 2017.

[21] A. Rachdi, A. En-Nouaary, and M. Dahchour, "Liveness and reachability analysis of BPMN process models," *J. Comput. Inf. Technol.*, vol. 24, no. 2, pp. 195–207, 2016.

[22] H. Boucheneb and R. Hadjidj, *Model Checking of Time Petri Nets*. Accessed: Aug. 12, 2018. [Online]. Available: https://www.bcs.org/upload/pdf/

[23] S. von Stackelberg, S. Putze, J. Mülle, and K. Böhm, "Detecting data-flow errors in BPMN 2.0," *Open J. Inf. Syst.*, vol. 1, no. 2, pp. 1–19, 2014.

[24] A. Awad, G. Decker, and N. Lohmann, "Diagnosing and repairing data anomalies in process models," in *Lecture Notes in Business Information Processing*, vol. 43. Berlin, Germany: Springer, 2010, pp. 5–16.

[25] S. Mallek, N. Daclin, V. Chapurlat, and B. Vallespir, "Enabling model checking for collaborative process analysis: From BPMN to 'Network of Timed Automata,'" *Enterprise Inf. Syst.*, vol. 9, no. 3, pp. 279–299, 2015.

[26] R. Boussetoua, H. Bennoui, A. Chaoui, K. Khalfaoui, and E. Kerkouche, "An automatic approach to transform BPMN models to Pi-Calculus," in *Proc. IEEE/ACS 12th Int. Conf. Comput. Syst. Appl.*, Nov. 2015, pp. 1–8.

[27] M. Güdemann, P. Poizat, G. Salaün, and L. Ye, "VerChor: A framework for the design and verification of choreographies," *IEEE Trans. Services Comput.*, vol. 9, no. 4, pp. 647–660, Jul./Aug. 2016.
[28] Uppsala University. (2015). *UPPAAL*. [Online]. Available: http://www.uppaal.org/
[29] The VerChor Team. (2013). *VerChor Framework*. [Online]. Available: https://pascalpoizat.github.io/verchor-web/
[30] W. Jaradat, A. Dearle, and A. Barker, "Workflow partitioning and deployment on the cloud using orchestra," in *Proc. IEEE ACM 7th UCC*, Dec. 2014, pp. 251–260.
[31] A. Finkel, "The minimal coverability graph for Petri nets," in *Advances in Petri Nets*. Berlin, Germany: Springer, 1993.
[32] *Bpmn2 Modeler*. Accessed: Aug. 12, 2018. [Online]. Available: https://www.eclipse.org
[33] F. Durán, C. Rocha, and G. Salaün, "Symbolic specification and verification of data-aware BPMN processes using rewriting modulo SMT," in *WRLA Rewriting Logic and Its Applications*, 2018, pp. 76–97.
[34] M. de Leoni, P. Felli, and M. Montali, "A holistic approach for soundness verification of decision-aware process models (extended version)," in *Proc. ER*, 2018, pp. 219–235.

**C. DECHSUPA** received the B.S. degree in computer information system from Burapha University, Thailand, in 2008, the M.S. degrees in software engineering from Chulalongkorn University, in 2012, and the Ph.D. degree in computer engineering from the Faculty of engineering, Chulalongkorn University, in 2018. From 2008 to 2015, he was a Database Programmer, and also a Senior System Analyst with many private sectors. His field of interest includes the formal method in software engineering and workflow design.

**W. VATANAWOOD** received the Ph.D. degree in computer engineering from Chulalongkorn University, Thailand, where he is currently an Associate Professor of computer engineering with the Faculty of Engineering. His research interests include formal specification methods and software architecture.

**A. THONGTAK** received the Dr.Eng. degree in electrical and electronic engineering from the Tokyo Institute of Technology, Japan. He is currently an Assistant Professor with the Department of Computer Engineering, Chulalongkorn University, Thailand. His research interests include asynchronous logic design and verification, dependable computing, and computer architecture.