IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 9, 2022, accepted April 8, 2022, date of publication April 18, 2022, date of current version April 27, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3168000

Stepwise Verification for the BPMN With Timed and Stochastic Process Using a Colored Generalized Stochastic Petri Net

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 by the Postdoctoral Fellowship (Ratchadaphiseksomphot Endowment Fund) of Chulalongkorn University.

ABSTRACT Internet of Things (IoT) technologies have been increasingly developed for real-time application in manufacturing processes to address heterogeneous devices and software effectively. Although almost all activities in a manufacturing process can perform an action when data objects arrive at the activity, physical devices or activities have process involving the operation of chance over time and probabilistic function for proceeding with their operations. Therefore, the formal verification of an IoT process design model have to consider the timed constraints, probabilistic tasks and dependencies between activities. This paper proposes a quantitative verification approach for analyzing and optimizing IoT manufacturing design models that are designed in business process model and notation (BPMN) representation. The transformation rules of BPMN element into the colored generalized stochastic Petri net (CGSPN) are proposed, and the stepwise approaches for refining and verifying the components of the CGSPN models are illustrated. Our framework helps the designers to automate the CGSPN model and to localize the operational gaps, time and flaws of the process manufacturing models.

INDEX TERMS Formal verification, colored generalized stochastic Petri net, timed and stochastic process, process manufacturing model, BPMN.

I. INTRODUCTION

The existing process manufacturing may be composed of many devices and individual machine systems. The load balancing control and some manufacturing operations are performed by the human (system control operators), which some process works on time constraints [1], probabilistic rules [2] and events. These manners result in the operational gaps, time and flaws of the process manufacturing control. To redesign the existing system, the stakeholders have to learn the existing system firstly, and to analyse and verify the designed model in order to make sure that the system to-be with an applying can reduce the operational gaps, time and flaws.

Communications and web service technologies [3] in the IoT provide opportunities for enhancing an existing process manufacturing system with integrated subsystems, physical devices or things and with real-time monitoring and control. IoT process manufacturing models include three clusters: the controller, IoT embedded software, and physical devices. Business process modeling notation or BPMN [4] can be used to design the middleware along with logical data, control flows and time constraints. The BPMN middleware describes the part of controller behaviors like the IoT service orchestration. It also acts as the IoT controller managing the interactions between the physical devices or between the physical devices and the software applications. The physical devices have an embedded software that is responsible for controlling the machines.

To analyze the performance of the IoT process manufacturing models, we must consider all three clusters of an IoT manufacturing model. A state transition of an activity in each cluster not only depends on data but also relies on delay, latency transition or random variables indexed by the time parameters. Each part of the BPMN IoT process manufacturing models quite involve various control flows and time constraints. These dependencies seem to be a complicated design that cannot be verified uprightly by using ordinary

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

software testing techniques or cannot be verified straightforwardly by using model checking techniques. Quantitative verification [5] using model checking techniques are a cumbersome procedure for the modelers. They have to create the formal model in the specific formal language and determine the corresponding time function for each state transition in the IOT process manufacturing models. Moreover, the modelers are required to understand a model checking tool, and model verification technique.

This paper proposes a methodology to help the IoT system designers design and verify the time and stochastic BPMN manufacturing models. The proposed methodology addresses the model abstraction process by providing the BPMN transformation rules and transformation framework to automate the CGSPN model [6], [7] which the CGSPN elements come from the extended BPMN time properties. The proposed framework is validated by analyzing and localizing the process behaviors of the obtained CGSPN models in Snoopy [33].

The organization of this paper is as follows. Section II reviews the related researches. Section III describes the background of IoT process manufacturing model. section IV discusses methodology and the proposed stepwise verification framework, and section V illustrates the implementation and validation with case studies and Section VI is the study's conclusion.

II. LITERATURE REVIEW

Due to the increased capabilities of software verification tools, formal verification using model checking [8] has increasingly been applied in computer and software engineering in the last decade. Qualitative verification is a valuable strategy for establishing reliability and validity for software design and development, and quantitative verification is no less important from an optimization perspective. The nature of manufacturing systems includes heterogeneous resources. The production process is complicated and may face verification problems. However, the shortcoming can be addressed by explaining and analyzing the manufacturing system in a high-level Petri net.

El Mehdi *et al.* [9] provided an analysis of the reliability of a repairable system using a deterministic and stochastic Petri net (DSPN). The use of the DSPN is unpractical for modeling the production system because it is not allowed to consider the steady state with the marking enabling multiple transitions. Peter Denno *et al.* [10] presented dynamic production identification using a colored Petri net and genetic programming (GA) [11] to offer the model of a smart manufacturing system. The objectives of this paper were to analyze and optimize the manufacturing system by developing a method and solution for the recommendation of an accurate model from the log operation content. This technique is appropriate for analyzing a dynamic production of the process manufacturing that relates to the production line balancing and queueing management. These authors checked the correctness of the obtained solution by using a SPN and a probabilistic Petri net [12]. In place of Petri net model automation by using the system logs, we automated a CGSPN model from the existing BPMN process models and represented the part of individual machine systems and their controllers separately as well. Zhang *et al.* [13] proposed an approach to enhance the performance of the shop-floor planning, execution and control of IoT real-time production. The authors used a hierarchical timed-colored Petri net (HTCPN) [14] to present an analysis module and analyze the interactions and the sensor data objects and used a decision tree (DT) [15] to identify the exceptions of production performance. The exception information derived from the DT-based exception extraction was important for system recovering and maintaining the production efficiency. The outcome of this work was a dynamic decision support system model that can be integrated with the real-time scheduling system. Although the proposed HTCPN models advocate multi-level event analysis, the behaviors of physical system and controller were not represented separately. We apply the multi-level model technique and capability of color relations for the topology arrangement as the multi individual representations, and represent the formal model by using CGSPN instead of HTCPN. Zhang *et al.* [16] presented a CPN-based prototype for monitoring and controlling the real-time sensor of a shop flow. The proposed technique is appropriate to manage and configure the multiple sensors. The application services of prototype are readable, and they run based on the heterogeneous sensors sent from the actual devices.

Zhou *et al.* [17] provided the stochastic timed Petri nets-based framework to analyze the emergency healthcare systems. The advantage of this work is that their implementation on STPN tool supports non-Markovian transition firing times and non-deterministic transition firing, supporting all timed transition firing policies and server types. The framework comes along with the simulation engine, STPN editor and components of configurator and resources repository. The authors should extend the STPN editor, and implement a function that allows for transforming or importing the STPN model generated from other modeling tools, in order to describe a complex system and reducing the time consumption of the STPN design process. Kheldoun *et al.* [18] proposedthe formal semantics of BPMN using Petri net based language, considering a subprocess, multiple instance, exception handling and data flow. Due to the limitation of high-level Petri nets, the data manipulation on a data object cannot be expressed and the actual data values cannot be considered.

Gharbi *et al.* [7] demonstrated the modeling and analysis of the performance of a multiclass retrial system using a colored stochastic Petri net (CSPN). The CSPN model represents several multi classes of customers and servers. The authors experimented with random server discipline and fastest free server discipline. This work shows a simple model, and the colored functions of the model are not realistic. For simulating multipurpose plants, this article [19] provided an approach to model the multipurpose plants by applying a CSPN. The approach was used to show the shared resources

and idle-time of the system. The objective of this work may be quite viable for flexible plant configuration, but the authors illustrated only a simple model with uncomplicated scenarios, and they did not detail the color set mapping of the model abstraction step. The work of [20] proposed the test case generation of the BPEL model in which colored Petri and the simple mapping rules are provided. But the mapping rules cannot be applied in a real business because they are so simple and they do not cope with BPEL fault handling.

The IoT manufacturing process involved in the cloud technologies requires the model-based verification approach of web service compositions. Domingos *et al.* [21] proposed the analysis of variable context to monitor and detect the changes in the values within the BPEL process by extending the BPEL language. The proposed approach advocates IoT data awareness for accessing IoT data objects to extract the device's behaviors, including the messaging of applications. Mi *et al.* [22] proposed a reliability attribute and quantitative verification for the BPEL specification. This was emphasized in the web service invocation behavior by describing the BPEL service composition in the Markov model, and the probabilistic model checker tool Prism was used to prove it. However, this work presented a simple case study.

Ana da Silva *et al.* [23] demonstrated that the Topology and Orchestration Specification for Cloud Applications (TOSCA) standard [24] can be set up automatically for Internet of Things environments. The authors showed the deployment of IoT heterogeneous middleware by establishing IoT environments using TOSCA, and the TOSCA model can be reused without further modification.

The works of [21]–[23] did not focus on the verification of IoT service integration; they only provided the solutions and demonstrated that the web service orchestration paradigm can be employed in IoT manufacturing. There are many research studies [25]–[27] that have provided Petri net-based verification approaches for modeling and analyzing the integration of the web service orchestration specification, but they did not consider the performance perspectives. In the model-based development approach, the manufacturing process can be modeled in terms of procedural logic using the semi-formal model representation, such as in the work of [28].

There are many researches providing the qualitative verification approaches for the web service composition that involves the probabilities. The work of [29] proposed the probabilistic pre-computation from the software requirements as parameterized symbolic expressions. This work is difficult to evaluate a set of symbolic expressions. The works of [30]–[32] provided the web service composition techniques associated with both functional and non-functional requirements. They classified the requirements of web service interaction into functional and non-functional requirements. Next, they implemented a tool supporting the probabilistic hierarchical refinement using the semantic principles according to the operational semantics of WS-BPEL. Their techniques are comparable to those of standard qualitative verification. But our work is focused on quantitative verification of the IoT manufacturing process models designed based on existing resources. However, those techniques can be applied for a qualitative verification of our designed formal models in the part of the BPMN controller model.

As in the cited related works, the use of the model checking approach for analyzing the performance and control of the IoT production process is an interesting paradigm for manufacturers who are enhancing the existing system. Most of them only provided qualitative verification. We aim to provide a framework for modeling and quantitative analysis by employing the CGSPN-based verification. Our techniques will be a viable option for the modelers to abstract the system model and demonstrate the system's performance. The stockholders can use the verification's results to make decisions regarding the development plan, manufacturing configuration and maintenance schedule.

III. BACKGROUND

A. A BPMN WITH TIMED AND STOCHASTIC PROCESS

IoT techniques generate viability in terms of process manufacturing in the supply chain management, real-time adjustment, predictive maintenance and inventory management. An example of the IoT process manufacturing, a dairy product factory places sensor on every piece of production equipment for inventory management and uses the IoT sensors to monitor the temperature during milk pasteurization [34]. The IoT also tracks how many times a qualitative process has been rechecked to influence the quality of finished dairy products and so on.

The simplified IoT topology is composed of three main layers [35], [36]: 1) the device layer has things with sensors and actuators; 2) the connectivity layer includes the Internet and gateway devices, including short-range wireless links, Bluetooth, Wi-Fi, and a mix of wireless technologies; and 3) the data center and application layer contain data storage and business applications of manufacturing. Microcontrollers are used for fabricating things, with which modern microcontrollers can make the IoT capable of deep learning to respond to the input data from devices and can send triggers to other devices. All the physical events captured by the things are first processed into information that we want to store, and they will be sent to the data center through the networks.

IoT devices and associated services, applications and manufacturing management processes are available for the factory to consume the resources, for which the available functionalities may dynamically change over time. To integrate the environment's heterogeneous functionalities, resources will be modeled as services. The heterogeneous physical devices and software used in IoT process manufacturing can be designed by the design principles of service-oriented architecture (SOA) [37]. There are many tools and service orchestration languages [38] for specifying the IoT service capabilities. They have a signature containing a set of inputs and outputs, behavioral specifications, preconditions, and postconditions. The capacity can also be orchestrated

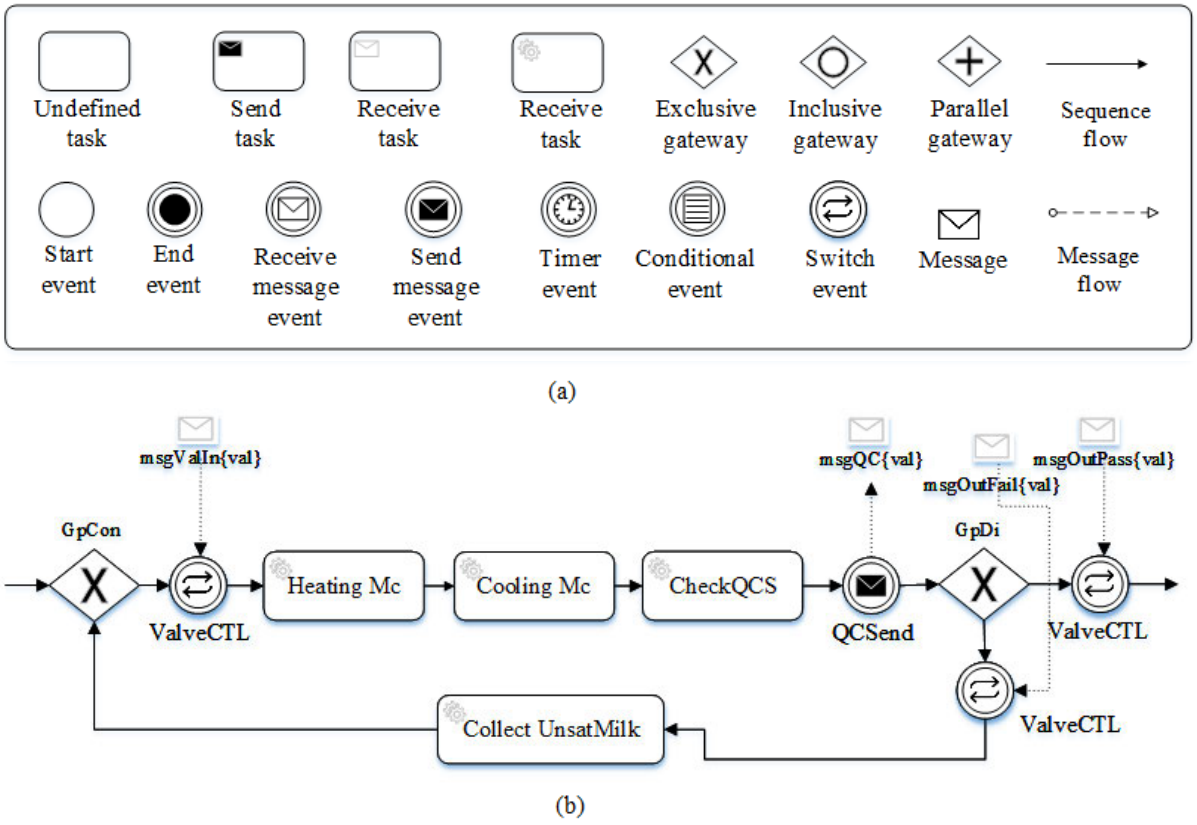

FIGURE 1. (a) Excerpt elements of BPMN, (b) BPMN use case scenario of the IoT milk pasteurization process.

by a IoT controler in a conversation called composite capabilities or composite service [39]. Whereas IoT middleware provides the services to software applications that use data for automatic decision-making. One challenge of smart factory design is how to measure the performance of production process because the data used for decision-making depends on the time and resources dependencies, and relies on random variables over time. Optimizing production and on-time resolution are difficult to evaluate for a process involving time intervals.

Business process model notation is graphical standard notation that comes along with many software modeling tools. The core elements of BPMN are composed of control flows, data flows, events, and tasks. The modelers can use BPMN to describe the IoT process diagram for more sophisticated behavior comprising multiple tasks and events, and these tasks may be executed simultaneously. Moreover, the activity may be synchronized and require resources and priorities to perform an execution. The determination of time constraints and stochastic processes [40] of the IoT manufacturing processes design seem to be intricate fashion because the BPMN time-events are not be designed for modeling a time and stochastic process.

However, BPMN can be extended and applied to describe the stochastic and timed process if it can store a time property, so that the extension of the BPMN task and event properties with time constraints is a crucial feature for this work. Included an applying the BPMN conditional event, the task operation specification based on certain conditions and chance or random variables over time is applied. For example, the arrival trigger data for the heating system depends on the temperature, which is related to the time, the volume of raw milk that pass in the holding tube, and the room temperature. Thus, such trigger data are sent arbitrarily over time from the controller to turn on or turn off the heating system. The core elements of BPMN is shown in Figure 1(a) and the excerpt BPMN specification of the milk pasteurization control is shown in Figure 1(b).

Although the simulation mode of the BPMN designing tool can validate the designed BPMN models, it cannot trace the concurrent process execution and does not support the performance analysis. Model checking [8] is one of formal verification techniques that advocate the concurrent analysis and performance analysis. In model checking process, a time and stochastic BPMN process model are transformed into a formal model written in specific formal language supporting a performance analysis. The derived formal model represents the process where tasks, control flows, events, and state transitions with times and uncertainties are quantified with operational semantics on the state transition that corresponds to

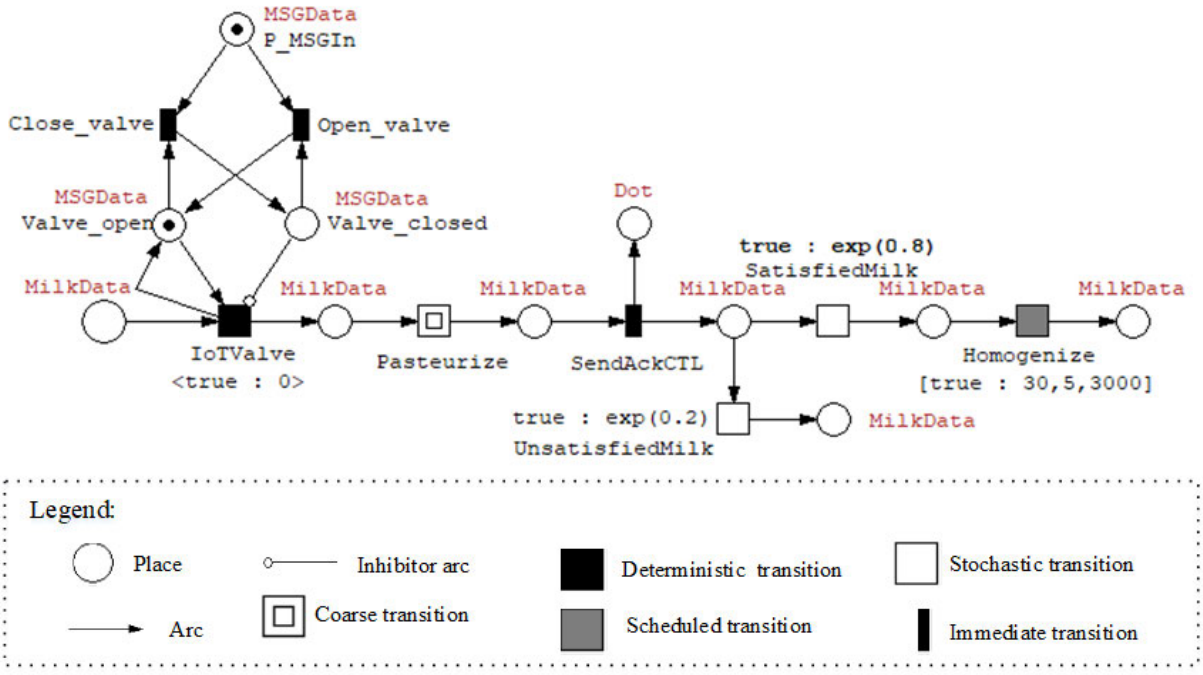

FIGURE 2. An example of the excerpt CGSPN pasteurization process model that was designed by using Snoopy.

the BPMN model. The timed and stochastic transitions of the obtained formal model are determined typically by the state transition semantics decorated with delay-time, scheduled-time and probabilistic weight, which these transitions can be modified arbitrarily.

The obtained formal model can be verified from both qualitative and quantitative perspectives if the model inscriptions are detailed adequately. In general, an event of the system is a discrete event where the stochastic state occurs only at an increasing sequence of random times. The current events associated with ancestor events to trigger the next state are called state transitions. Each state transition has a stochastic mechanism or time function for determining the next state. For analyzing the BPMN design model before implementation, there are many frameworks and tools [41]–[43] that provide powerful composition features for modeling discrete-event stochastic systems, which design with the building blocks for the state transition expressions, event-scheduling mechanisms, simulation mode, verification mode, and statistical reports.

B. COLORED GENERALIZED STOCHASTIC PETRI NETS (CGSPN)

The colored generalized stochastic Petri net [7], [44], [45] has been shown to be an alternative formal model that is an appropriate representation for describing, verifying and analyzing the performance of concurrent and synchronous systems. The CGSPN is an extension of the stochastic Petri net (SPN) and colored Petri net (CPN) [46] by the combination of the CPN programing language and discrete-time Markov chain of the SPN. The state transitions of the CGSPN model underlie an event-scheduling mechanism or time function associated with each transition. The transition firing may be probability distribution values, delay, zero-delay and absolute points of time.

The core elements of the CGSPN comprise the Place, Transition, Arc and Inscription. The number of Tokens in the place depends only on input and output incident functions called arc inscriptions. Arcs terminating in open dots are used to represent an inhibitor of transitions. a stochastic transition. The immediate transition is the transition that is without a time function or zero-delay because it rapidly fires the instant it becomes enabled. The scheduled transition works under a condition with absolute points of time, while the deterministic transition is the transition associated with delay-time. Figure 2 shows an example of the CGSPN model designed by using Snoopy [42].

The large size of a CGSPN model may result in a difficult and error-prone analysis. The modelers can construct compact models by using the advantages of parameterization and restructuring of the model. The refinement in the hierarchical structure [47] can alleviate the sophisticated behaviors and avoid the state space explosion problem. The modelers should address the complicated behaviors of the components before verification. However, this approach requires experiences with partial and composite verification techniques to address the tremendous number of system states.

Our reason for choosing CGSPN is that an optimization stochastic control problem of the existing manufacturing process fundamentally contains tasks with random behaviour

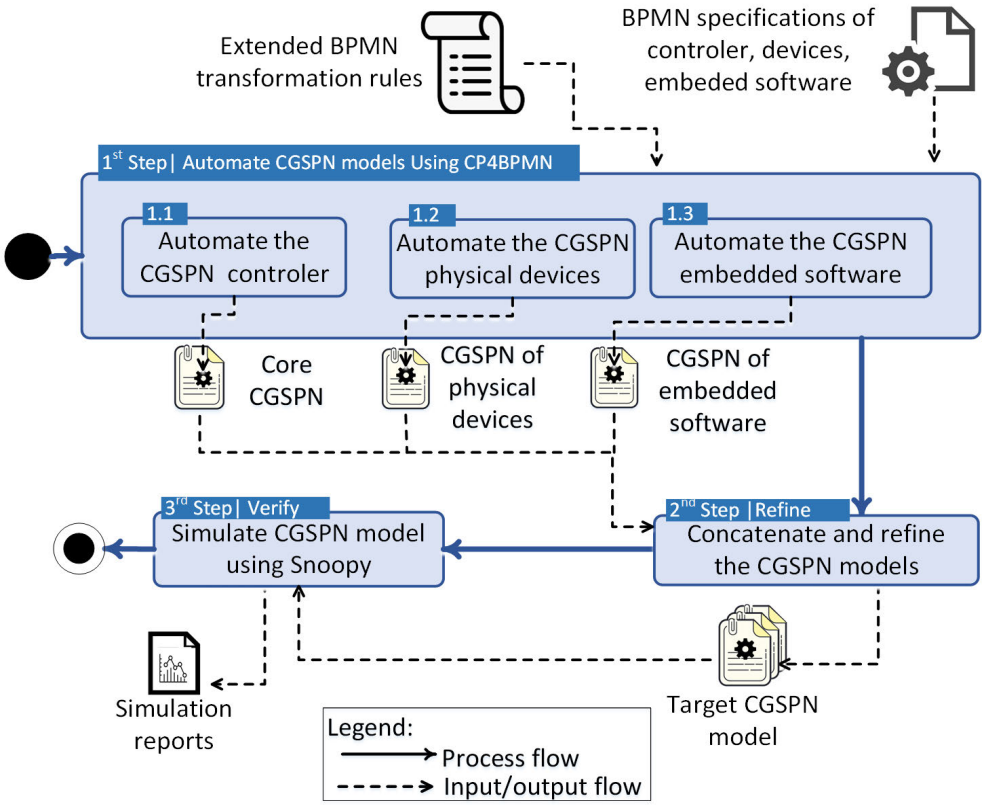

FIGURE 3. Overview of the proposed stepwise verification process.

over time. Its occurrence colors, token colors and colors relation are corresponding to the subscripted variables that are defined by the functions attached to the arcs and corresponding transitions. They are used for describing and representing the properties of resources, things, and messages. These advantages are relevant for building the system behaviors, the priority of the sensors, the message queue handling and overriding policy, including the firing conditions of the transitions. Moreover, we can build a formal model analogous to the manufacturing process by using a CGSPN, whose graphical process manufacturing model with the partitioning clusters would reflect the better understanding.

Snoopy provides the animation mode and simulation mode for analyzing the Petri net-based models. It supports the hierarchy and color modeling concepts that are compact and readable representations. Snoopy has been widely used in many domains, such as biological science and computer science. Quantitative verification with animation and simulation modes is an interesting feature because it provides many functions to make verification easier. For qualitative verification of a CGSPN model, the modelers can use temporal logic to explore the undesirable properties, such as deadlock, unreachable tasks, invariant and soundness properties [48]. Snoopy also provides the external analysis features: Charlie [49], Marcie [33] and CPN tools. These features include structural analysis, invariant checking, and explicit computational tree logic (CTL) and linear temporal logic (LTL) [50] model checking.

IV. TRANSFORMATION RULES AND FRAMEWORK

Our stepwise verification framework is shown in Figure 3. The inputs of the process are the BPMN specification and IoT device specification. We extended the CP4BPMN transformation rules [52] to automate a formal model written in the CGSPN XML format of Snoopy [33]. Next, the CGSPN of IoT devices is created, and the inscriptions of the obtained CGSPN models are refined if the BPMN model input are detailed inadequately. Last, the performance analysis is performed by using Snoopy. The extended BPMN transformation rules are in subsection A and the verification processes are described in subsections B.

A. BPMN TRANSFORM RULES

We extended Eclipse BPMN2 modeler [51] to support the configuration the time-processes for each Task or Event notation. Figure 4(a) shows the BPMN service task extended with the scheduled time properties. The task information of BPMN is shown in Figure 4(b), stored in XML format. The existing transformation rules of [52] cannot be straightforwardly used to transform BPMN models because a target model requires the time transitions supporting a quantitative analysis. Thus, we revise certain BPMN transformation rules and implement

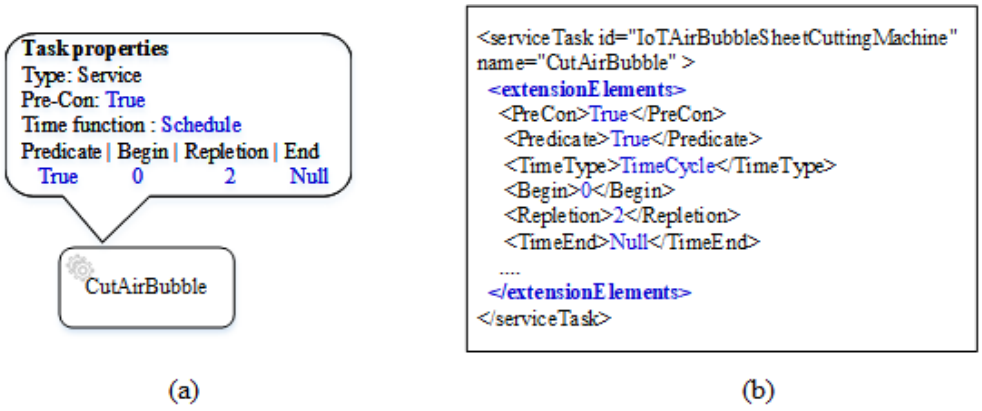

FIGURE 4. The extended BPMN task configuration and XML representation.

the BPMN transformation rules for time event and conditional event notations. Due to space limitations, we describe the part of the core extended BPMN transformation rules for time task, time event and conditional event. We refer the ordinary BPMN formal definition and transformation rules defined in [52], [53] to create the extended transformation rules. The additional formal definitions and the extended transformation rules of the BPMN specification, and the obtained CGSPN model are as follows.

Definition 1 (An Extended BPMN Process Model): A BPMN model is a seventeen-tuple $BP = (N, A, G, fGT, V\_n, V\_a, F, F\_x, E\_start, E\_end, L, M, eA, C, Tc, fA, fT)$ where

N is a finite set of nodes { *event, task and gateway* }.

A is a set of task and event, which is a sub-set of N; $(A \subseteq N)$.

G is a set of gateways, $G \subseteq N$.

fGT is a mapping function used to indicate the gateway type, fGT: G $\rightarrow$ {*exclusive, inclusive, parallel, event-based*}.

V_n is the name of task or gateway.

V_a is an operation of task.

F is a set of sequence flows $(F \subseteq N \times N)$.

F_x is a guard condition of gateway $F \rightarrow F\_x$.

E_start is a set of start events, $E\_start \subseteq N$.

E_end is a set of end events, $E\_end \subseteq N$.

L is a set of swim-lane (If BPMN model is a process diagram, L is an empty set).

M is a set of message flows that cross the swim-lane.

eA is a set of extended time BPMN nodes, $eA \subseteq A$. The extended time BPMN nodes include the Task, Conditional, Event notations along with the time property configuration. For $ea_i \subseteq eA$, $ea_i = (TName, Precon)$, where ea_i.TName means the task name of ea_i described as a string expression. ea_i.Precon means the task pre-condition of ea_i, which is Boolean.

C is a set of conditional BPMN nodes.

Tc is a set of time constraints {*General, Delay, Schedule, Prob*} where *General* is an original BPMN node and the others are the extended time properties of the BPMN node.

fA is a mapping function used to indicate a type of the extended time BPMN node and the conditional BPMN node, $fA: eA \rightarrow Tc$

fT is a labeling function used to indicate a time constraint or probabilistic value configured, $fT: (eA \times Tc) \rightarrow$ *time value*.

Definition 2 (Colored Generalized Stochastic Petri Net Model) A CGSPN model is a 10-tuple $CGSPN = (P, T, fT, fF, A, AI, \Sigma, V, fC, fG, fE, fI)$ where:

P is a set of places.

T is a set of transitions such that $P \cap T = \theta$.

fT is a mapping function used to indicate the type of transition fT: $T \rightarrow$ { *Stochastic, Immediate, deterministic, Scheduled* }.

fF is a function that indicates the time function of a transition. For each $t \subseteq T$, if $fT(t)$ is Stochastic, the transition firing rate of t is the probability value or mathematical function stored in a lookup table. If $fT(t)$ is Immediate, the timed function value of the transition is time zero. If $fT(t)$ is deterministic, the transition firing is dependent on a positive amount. If $fT(t)$ is Scheduled, the transition firing is rooted on the periodic list of time.

A is a set of arcs, $A \subseteq ((P \times T) \cup (T \times P))$.

AI is a set of inhibited arcs, $AI \subseteq (P \times T)$

Σ is a finite set of color sets.

V is a finite set of typed variables such that $Type(v) \subseteq \Sigma$ for all variables $v \subseteq V$.

fC is a color function used to assign a color set to each place, fC: $P \rightarrow \Sigma$.

fG is a labeling function used to assign a guard to each transition, fG: $T \rightarrow$ *conditional expression* and $Type(fG(t)) =$ *Boolean* such that $t \subseteq T$.

fE is an arc expression function used to assign an arc expression to each arc, fE: $A \rightarrow$ *arc expression* such that $Type(fE(a)) = fC(p)$, where p is the place connected to the arc a. The arc expression may be a list of variables or conditional expressions along with the lists of variables.

fI is an initialization function used to assign an initialization expression (initial marking) to each place, fI: $P \rightarrow$ *init expression* such that $Type(fI(p)) = fC(p)$ for all places $p \subseteq P$.

Definition 3 (A Hierarchical CGSPN Model): A hierarchical SCPN model is a tuple $HCGSPN = (CGSPN, TS, PS)$ where:

CGSPN is a set of non-hierarchical models.

TS is a set of macro nodes or macro transitions used as agents of the subgraph *CGSPN* where $CGSPN \in CGSPN$.

PS is a set of interface places of the macro transitions.

The CGSPN structures obtained from the transformation rules are called the CGSPN constructs. The extended transformation rules consist mainly of twelve rules, and the obtained CGSPN constructs are shown in Figure 5 through Figure 16.

Rule no. 1: Time task transformation: To automate the CGSPN constructs from a task notation, the control flow or gateways including the BPMN elements associated with the task are transformed into the CGSPN constructs by using the extended transformation rules of [52], [54]. The obtained CGSPN construct of the BPMN task contains three transitions: *T_Task_Str,T_Task_Act, and T_Task_Com* , representing *Start*, *Action* and *Completion* state of a task execution. The corresponding transition type relies on the time properties determined in the BPMN task, indicated by using the mapping function *fA()*. The transition type of *T_Task_Str* depends on the extended BPMN task properties, which the mapping cases are shown as Equations 1. For example, the transition type of *T_Task_Str* is mapped into a deterministic transition if the task properties is delay time. Whereas the transition type of *T_Task_Act, and T_Task_Com* is the deterministic transition with zero delay. The CGSPN construct of BPMN task is shown in Figure 5. However, the designers have options to transforming the BPMN task into the CGSPN construct with a single transition in line with Rule no. 4 in order to reduce the model complexity.

$$f(eA) = \begin{cases} DeterministicTran. & fA(eA) = Delay \\ ScheduledTran. & fA(eA) \in \{General, Schedule\} \\ StochasticTran. & fA(eA) = Prob \end{cases} \quad (1)$$

Rule no. 2: Probabilistic task transformation:

For each BPMN task determined as a probabilistic activity, the CGSPN construct is similar to that of the time task detailed in the transformation rule no. 1; but the transition *T_Task_Str* is the stochastic transition attached with a probabilistic function and firing rate that are automated from the BPMN task properties. Figure 6 shows the CGSPN construct of the BPMN task where "exp()" is the exponential probability distribution function and 0.2 is the firing rate of the stochastic transition *T_Task_Str*. The inscription "[Count>5]" is the guard condition of transition, and the other transitions are the deterministic transition with zero delay.

Rule no. 3: Task with boundary event transformation:

For each boundary event on the task notation, the CGSPN construct contains from two deterministic transitions: *T_Ev_B4Act* and *T_Ev_Aft_act*, representing the events

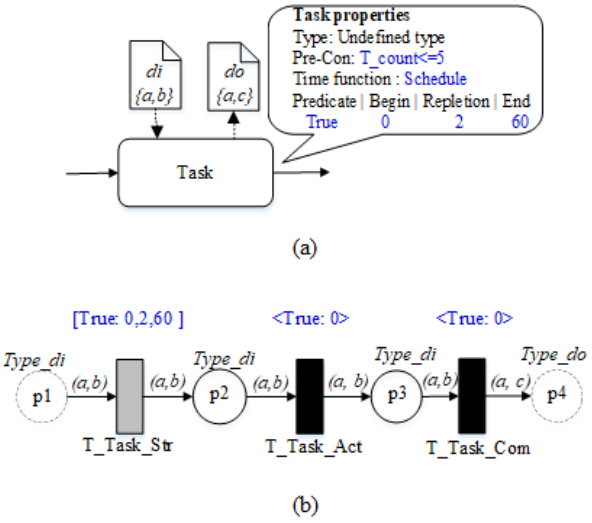

FIGURE 5. Transformation rules of the task notation. (a) BPMN undefined type Task with time scheduled configuration. (b) CGSPN construct of BPMN scheduled task. T_Task_Str is scheduled transition, while T_Task_Act and T_Task_Com are deterministic transition.

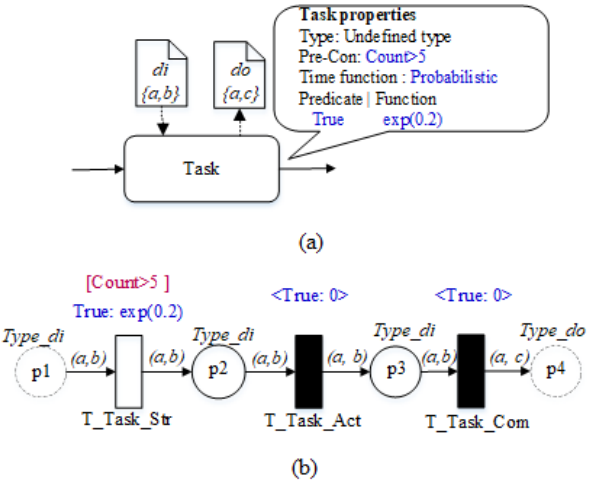

FIGURE 6. Transformation rules of the probabilistic task notation. (a) BPMN undefined type Task with probabilistic activity configuration. (b) CGSPN construct of BPMN probabilistic task.

that occur before and after the task's execution. The default delay time of them is zero, if BPMN boundary event is the event without the time property determination. The delay time of transition *T_Ev_Aft_act* can be refined to represent the task that computes or aborts an operation because a predefined interval of time has passed. Figure 7 shows the CGSPN construct of the task with boundary event in which the deterministic transitions *T_Ev_B4Act* and *T_Ev_Aft_act* with zero delay connect to the core transitions of task.

Rule no. 4: Undefined-time task:

For each undefined-time task, it must be automated into the delay transition by default due to the quantitative verification purpose, thus the CGSPN construct of the undefined-time task contains only one deterministic transition with zero

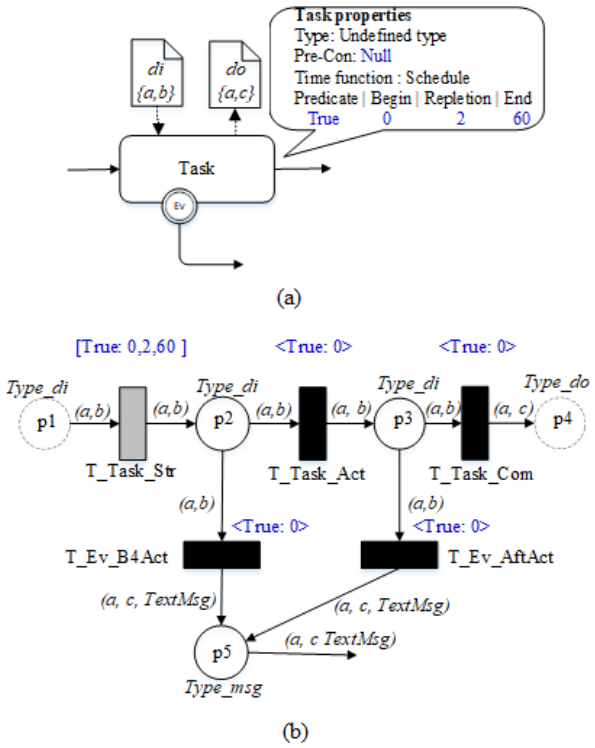

FIGURE 7. Transformation rules of the boundary event notation. (a) the boundary event on BPMN undefined type task with time schedule. (b) CGSPN construct of boundary event that connects to CGSPN task construct.

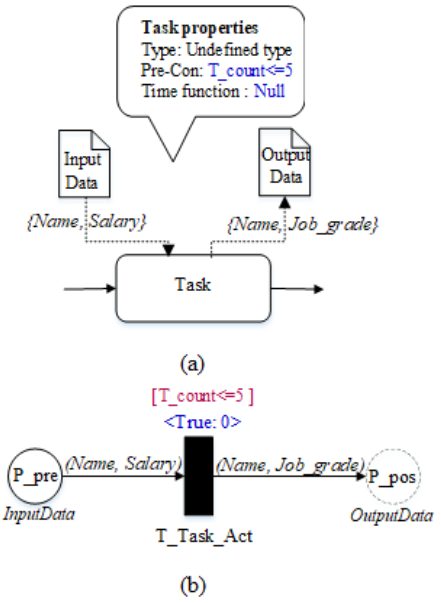

FIGURE 8. Transformation rules of the undefined-time task notation. (a) the BPMN undefined-time task. (b) CGSPN construct of undefined time task.

delay. Figure 8(b) illustrates the CGSPN construct of the undefined-time task derived from the BPMN undefined-time task in Figure 8(a).

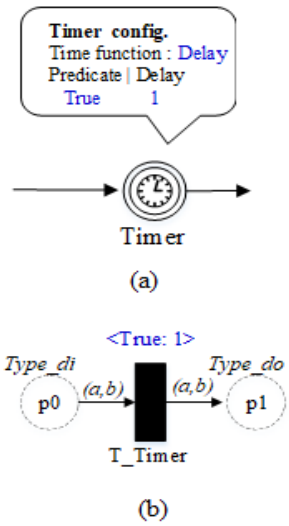

FIGURE 9. Transformation rules of the timer notation. (a) the BPMN delay-timer, (b) CGSPN construct of delay-timer.

Rule no.5: Timer event:

For each timer event, the target CGSPN construct contains the deterministic transition attached by the delay that comes from the BPMN timer configuration. If the timer type is *TimeDuration* or Delay, the CGSPN transition is the deterministic transition. If the timer type is *TimeDuration* or Schedule, the CGSPN transition is the schedule transition. While BPMN timer configured with a *TimeDate* expression is mapped into the deterministic transition by default because the verification tool does not support the TimeDate process. Figure 9 shows the CGSPN construct of the BPMN timer event that the modeler determines the timer properties is a delay time with "*<True: 1>*", which the transition *T_Timer* is the deterministic transition that the guard condition of the transition is Boolean "True" and delay is "1".

Rule no. 6: Gateways (except Event-based gateway):

For each gateway including divergent gateway and convergent gateway, the CGSPN construct and CPN construct of [52], [54] are the same but the transition type of the CGSPN construct is the immediate transition. The immediate transitions of the CGSPN gateway represent that the transition fires the token(s) immediately when the token arrives. Figure 10 shows the transformation rule of the BPMN parallel gateways, which the transition *T_GpD* and *T_GpC* are the immediate transition.

Rule no. 7: Event-based gateway:

For each event-based gateway, the CGSPN construct is similar to the CPN construct of [52] but the transition depends on the consecutive task or event of the gateway. Figure 11 illustrates the transformation rule of event-based gateway, which the events rely on the receive Task and Event. Thus the transitions in the CGSPN construct in Figure11(b) contains the scheduled transition *T_Task_str* derived from the BPMN task and the deterministic transition *T_Event_read* comes from the BPMN event in (a). The CGSPN construct possesses

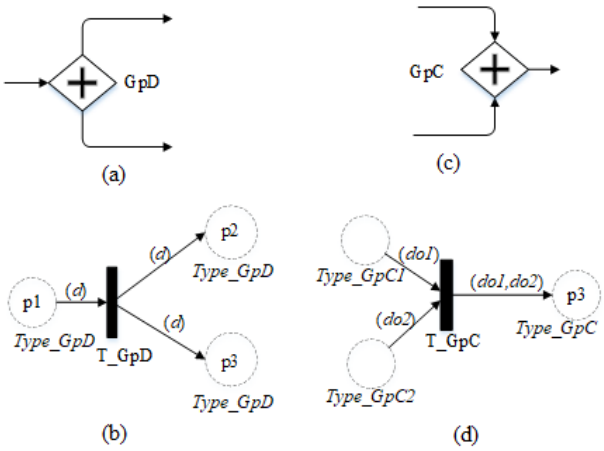

FIGURE 10. Transformation rules of the gateway notation, (a) the BPMN divergent gateway, (b) CGSPN construct of divergent gateway, (c) the BPMN convergent gateway, (d) CGSPN construct of convergent gateway, which T_GpD and T_GpC are immediate transition.

the inhibitor arc(s) to avoiding multiple-triggering events as well.

Rule no. 8: Divergent exclusive gateway with probability distribution function:

The divergent exclusive gateway is extended to describe the stochastic distribution task because another objectives of this work is designing and verifying the stochastic process model. The exclusive gateway properties will be automated to be the CGSPN construct containing the stochastic transitions if the time function of the BPMN gateway properties is ''Probabilistic''. Figure 12 shows the transformation rule of the BPMN exclusive gateway determined as the stochastic distribution process by using an exponential probability distribution function ''exp()''. Figure 12(b) shows the derived CGSPN construct of the BPMN divergent exclusive gateway in (a), the guard condition ''[Count>=5]'' and ''[Count<=5]'' comes from the guard condition on the outgoing sequence flow of the gateway, and the probability distribution function is derived from the BPMN gateway properties.

Rule no. 9: Conditional event:

For each conditional event, the CGSPN construct consists of the single deterministic transition. The guard condition of the transition is derived from the BPMN condition configuration and the delay is zero (''<True:0>'') by default. The transformation rule of the BPMN conditional event is shown in Figure 13. However, this transformation rule support the single-condition determination only.

Rule no. 10: Send event and receive message event:

For each send task or receive task, the deterministic transition is used to represent send or receive event. The delay time is assigned to ''<True:0>'' by default. The transformation rule of send and receive event are shown in Figure 14, which Figure 14(a) and (b) are the BPMN send event and BPMN receive event. Figure 14(c) and (d) are the CGSPN construct of the send and receive event respectively.

Rule no. 11: Send task and receive task:

For each send task and receive task, the obtained CGSPN construct consists of three transition that are similar to the construct of the BPMN task transformation rule No.1 shown in Figure 5. Figure 15 (c) is the obtained CGSPN construct of the BPMN receive task and Figure 15 (d) is the CGSPN construct of the BPMN send task. The transition type of transition *T_Task_str* depends on the task properties. As the examples in Figure 15(c) and (d), it indicates that the modelers set the time function of BPMN task to be ''Schedule'' because the transition *T_Task_str* shows the scheduled transition with the firing schedule [True: 0,2, 60].

Rule no. 12: Switch event:

Switch event is the novel BPMN notation for describing the valve of manufacturing process model. The transformation of the switch event gives the CGSPN construct along with the deterministic transition controlling the transition firing by inhibitor arc. Figure 16 shows the transformation rule of the BPMN switch event; the BPMN notation shows in Figure 16(a) and the obtained CGSPN shows in Figure 16(b). The transition *T_On* and *T_Off* represent the valve state ''On'' and ''Off'' respectively. They are the immediate transition consuming the token received from a controller and firing the token to control the valve closing and opening. The obtained CGSPN construct of the switch event also consider the priority and ordering of the On-Off message. The modelers can set the priority and queue of the messages for the transition firing of the transition ''Priority'', and it sends an acknowledgement message or current status back to the controller for each corresponding time unit.

B. THE PROPOSED STEPWISE VERIFICATION FRAMEWORK

As the stepwise verification process shown in Figure 3, the process manufacturing models comprise the controller, embedded software, and physical devices. It can be observed that the verification steps are separated accordingly with the components of process manufacturing model. The simplified topology of the milk separation process is shown in Figure 17. The IoT topology of the milk preparation process comprising three parts is shown in Figure 17 (a), and the target CGSPN model obtained from the transformation of the specifications in (a) into CGSPN models is shown in Figure 17 (b), which the connection to each other is at the corresponding transition with the arcs. The blue place is an input/output place of the CGSPN construct. The details of each verification process are as follows.

1) AUTOMATE THE CGSPN CONSTRUCT OF THE CONTROLLER, PHYSICAL DEVICES, AND EMBEDDED SOFTWARE

The controller is an important component driving the physical manufacturing process. A controller may control a single device or multiple devices. For example, the controller of pasteurization manages two subsystems that comprise the heating system and cooling system with four physical devices.

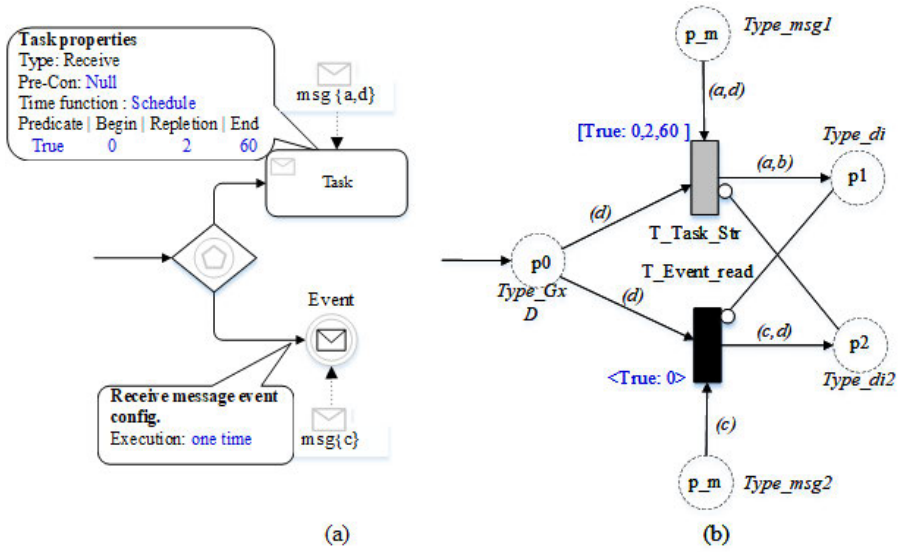

FIGURE 11. Transformation rules of the event-based gateway notation.

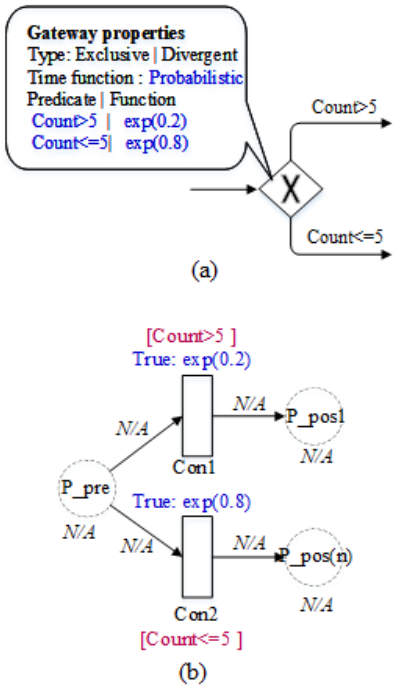

FIGURE 12. Transformation rules of the exclusive gateway notation.

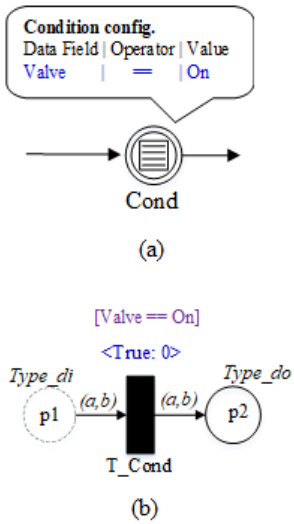

FIGURE 13. Transformation rules of the conditional event notation.

The management of message queue and priority in the controller is a straightforward manner. In practice, the multi-level and multi-type of events are defined in a lookup table which contains rule-based conditions. Our framework covers the logical controller and physical devices of the manufacturing process described in the BPMN representation. We transform the BPMN specifications of the controller, physical devices, and embedded software into the CGSPN models one by one.

The CGSPN constructs of embedded software will be the mediator between the CGSPN models of physical manufacturing and the CGSPN models of controllers. We can represent device's behaviors in three ways.

2.1) Using a transition $t \in T$ if it is a simple event or atomic task.

2.2) Manual creating the subnet if the device has complicated events and setting it up to be the macro transition $ts \in TS$.

2.3) Automating the CGSPN models of the embedded software from the existing specification documents described in BPMN representation by using CP4BPMN.

To reduce the model complexity, the embedded software should be considered and modeled only on the considerable events. The application of an equivalence class partitioning technique [56] is applied for abstracting the device behaviors. The behaviors of embedded software are mimicked with input

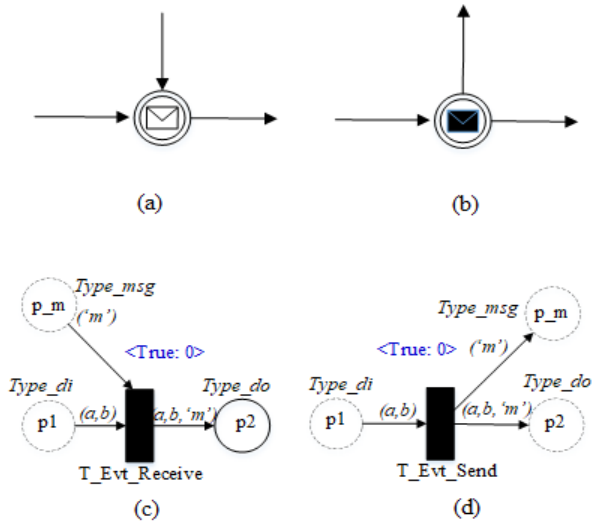

FIGURE 14. Transformation rules of the send and receive event notation.

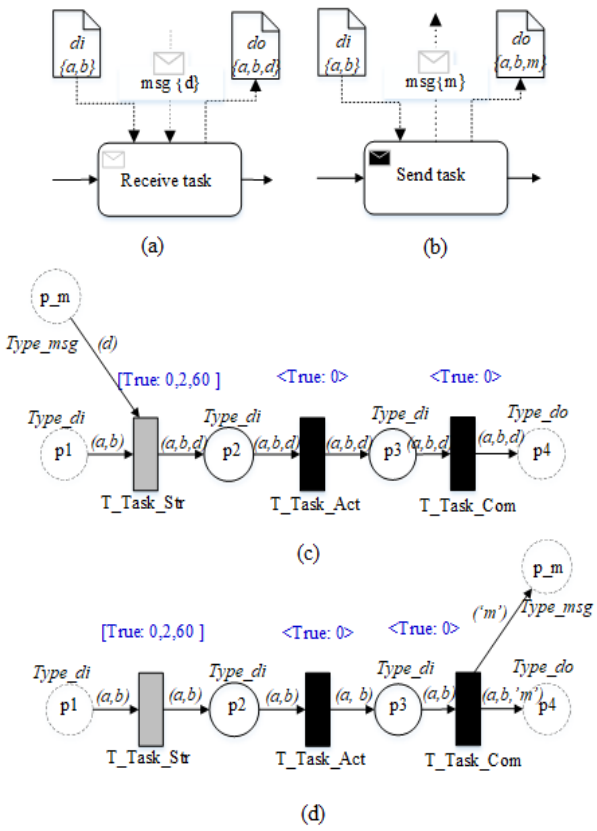

FIGURE 15. Transformation rules of the conditional event notation.

and output sets as the token colors message received from the controllers, and the token sent to the CGSPN construct of the physical devices.

The considerable events of embedded software are imitated as a data set to be a *subTask* as *stub* or *mock-up task*. For instance, the thermometer on the milk holding tube sends temperature measurement data every one second to the data

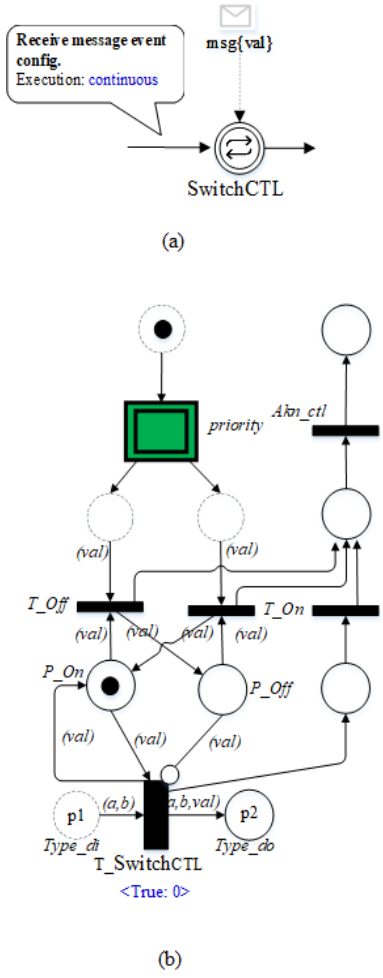

FIGURE 16. Transformation rules of the switch event notation.

center, but it triggers the heating system to turn on and turn off when the temperature is lower than 80 or greater than 100 degrees Celsius, respectively. Thus, considerable cases of the thermometer are the values of the *invalid lower bound, max, norm, max* and *invalid upper bound*, i.e., 79, 80, 90, 100 and 101, respectively. In the case that a subservice requires the result of a consensus algorithm from multiple devices, the considerable cases are a set of possible decision cases of the consensus algorithm.

Due to the system having the commands priority, the CGSPN construct of the embedded software must have a message queue handler and an overriding policy. The CGSPN construct receives the messages (tokens) from the other devices or system control operators (SCOs), where the messages of SCOs usually have higher priority. We use the token colors to determine the message priority. For instance, a set of messages with three color tokens *1' (0, 'X')++1' (0, 'Y')++1' (1, 'Z')* is in the *InputPlace* of the transition *t*. The transition *t* will consume token *1' (1, 'Z')* first, since it is set to be the first priority with color "1". This practice is used for the message command requested from SCOs to reset a system or the messages produced from the event handling flow and

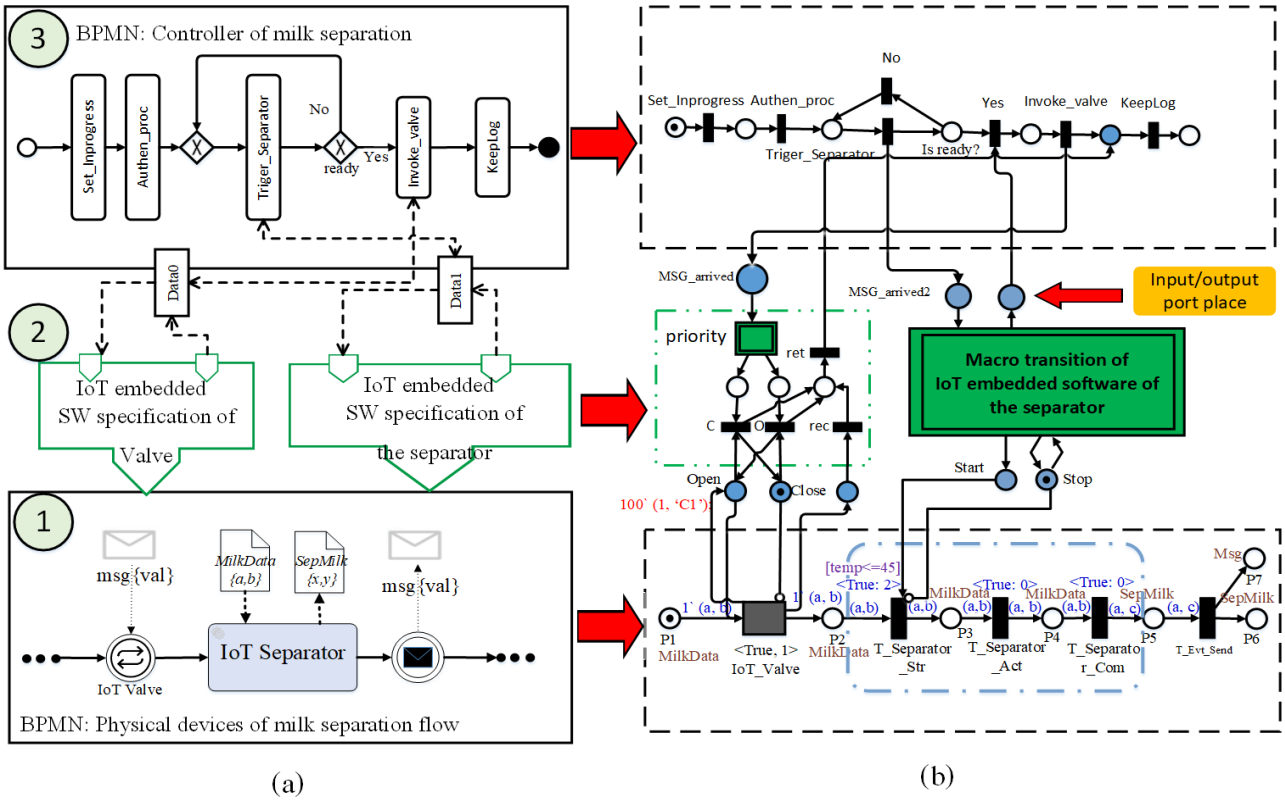

FIGURE 17. The example of IoT topology of the milk preparation process.

from the system running into abnormal cases. The green double dashed-line box in Figure 17(b) shows an example of a CGSPN construct of IoT embedded software of a valve.

2) CONCATENATE AND REFINE THE CGSPN CONSTRUCTS

After all the component specifications are transformed into the CGSPN constructs, they will be verified separately, and be manually connected to each other at the corresponding transition with the arcs. Figure 17(b) shows an example of the CGSPN milk separation process comprising three parts; the CGSPN physical devices and manufacturing flow, the CGSPN embedded software specifications, and the CGSPN controller. These CGSPN constructs are derived from the BPMN specifications in Figure 17(a). The arc(s) connected to the other construct (e.g. the incoming and outgoing arcs of the place "MSG_arrived") require the arc inscription expression that conforms to the Backus-Naur Form (BNF) [55]. At the transition representing the synchronization process, we must manually refine the transitions output arcs for choosing the correct tokens and data version.

The target CGSPN model of the integrated CGSPN constructs that are obtained from subsections 1) through 4) the CGSPN automation in Subsections 1) is a flat model. The target CGSPN model may be large and sophisticated because of the substantial size of the input specifications and devices involved. The hierarchical verification can be applied to alleviate the complexity and verification time consumption. There are three approaches to rearrange the CGSPN model in the hierarchical structure.

- Rearranging the part of the CGSPN model obtained from the BPMN specification. The modelers can partition the CGSPN model by using *the physical subsystem*. For example, the dairy production process comprises four subsystems: the preparation, pasteurization, homogenization and packaging. The modelers can verify all of subsystems separately and gradually, and can determine the verified subsystem to be the macro transition.

- Reducing the CGSPN construct of embedded software as macro transition. The mentioned CGSPN construct in Figure 17(b) indicated by the double green line box represents the embedded software of the milk separation process, which is reduced into the hierarchical structure with a macro transition.

- Combining the above approaches is used to convey the topmost model for integration verification.

3) VERIFY THE OBTAINED CGSPN MODEL IN SNOOPY

Since the proposed framework emphasizes a quantitative verification, we would like to omit the details of qualitative verification, such as deadlock checking and livelock checking.

TABLE 1. The details of the primary IoT devices of the MPF.

No.	Description
D1	Valve controller and ultrasonic level sensor for amount measurement and controlling of milk in the holding tank.
D2	Liquid control ball valve for cheese making.
D3	Liquid control ball valve for measuring and controlling of the raw milk into the production system.
D4	Separating machine controller.
D5	Liquid control ball valve for butter making.
D6	Standardization machine controller.
D7	Optional addition of vitamins and flavors.
D8	Liquid control ball valve that is controlled by the pasteurization controller for measuring and controlling of raw milk into the pasteurization system.
D9	Resistance temperature detector for real-time measuring and recording temperature (heating measurement).
D10	Controller of the water-heating system.
D11	Controller of the cold-water cooling system.
D12	Resistance temperature detector for real-time measurement and temperature recording (cooling measurement).
D13	Homogenization machine controller.
D14	Liquid control ball valve for packaging lines.
D15	Controller of the automated conveyor system.
D16	I-Code-RFID for inventory management and logistic.
D17	Liquid control ball valve of cleaning agents.
D18	Cleaning in-place system controllers

After refinement the CGSPN constructs, we set and run the simulation parameters for the obtained CGSPN model in Snoopy. The tool show the simulation results as a table or plot of colored places or transitions.

V. FRAMEWORK VALIDATION AND RESULTS

In this section, we validate our framework by using a case study with the IoT manufacturing system models in the master plan of milk products factory (MPF). The BPMN specifications are automated into the CGSPN constructs by using CGSPN. Next, the obtained CGSPN constructs are taken to be an input models of Snoopy for refinement and analysis. Figure 18 shows the topmost physical IoT manufacturing process and physical devices of dairy production manufacturing.

The raw milk produced by the cooperative members will be collected by the suppliers of raw milk storage units. The raw milk details of every farm are recorded in a specific database and periodically sent to the factory. When the raw milk is transported to a MPF by tanker trucks, it will be transfused into the holding and pumped through the main tubes to the other machines. Based On the real factory, approximately two hundred thousand litres a day of raw milk enter this production process. During the milk production process, the system control operators independently monitor and control each mechanism underlying the data of the production plan and materials plan. Table 1 shows the details of the primary IoT devices (D1-D18) of the MPF. The devices receive and send the data packages via the intra-web service. There are five core steps of the production process: the raw milk preparation, the pasteurization, the homogenization, the packaging and the cleaning. The cursory production flow is as follows.

First, the raw milk is tested in the physical and chemical laboratory by the quality control officers (QCOs). This process requires three hours. If the raw milk satisfies the quality standard, it will be partly distributed for the cheese-making process. Before dairy product production, the separator or centrifugal machine separates milk into cream and skimmed milk. The cream will be sent for butter-making, whereas the skimmed milk is transmitted into the standardization process to add the milk components that have been previously separated by the separator back into the milk in precise standardized amounts and to add the optional vitamins and flavors. Next, the pasteurization process is performed based on the capacity, the target product type and available pasteurizers, in which the pasteurizer is under the control of the water-heating system and the water-cooling system. The pasteurization process takes approximately one minute with precise controlling of the heating and cooling systems and the amount of milk in the holding tube. Then, the fat molecules in the milk are broken down by the homogenization machine. Last, the milk of desirable standard will be packaged by the packaging system. As the system runs, every twentieth hours, the SCOs will request cleaning of the in-place system to clean the holding tube, tank and process equipment.

Figure 18 shows the topmost physical IoT process manufacturing the physical things with sensors of the process; Figure 18 (a) shows the process shows only the case that the step of standardization is performed before pasteurization. Figure 18 (b) shows the topmost dairy process manufacturing model depicted by using BPMN. However, the process described in Figure 18 (a) is only one manufacturing process of the dairy product. In general, one manufacturing process can manufacture multiple dairy products. The system also supports other production processes called multi-purpose production. The step of each product is dependent on the requirements of the product owner. For example, the raw milk often passes the pasteurization process after standardization and flavor addition, but some product owners need to pasteurize the raw milk before the optional addition of vitamins and flavor. Thus, the milk circulation is modeled by the BPMN process as shown in Figure 18 (b), in which the operation sequence is controlled by the system control operators via the controllers that are designed in BPMN representation.

Although the physical manufacturing process seems to be simple and sequential, the controllers designed as BPMN middleware are complicated. The BPMN models have many more details with many control flows and data flows, including the fault handling flows or the exception events that handle the likely inevitable abnormal events to maintain the process stability. We address these challenges by separating the whole system into subsystems and using a bottom-up verification approach. We transform the physical flows, IoT embedded software specifications and BPMN processes of each controller into CGSPN models. The obtained CGSPN constructs of each subsystem or component are assigned to be the subnets. Next, we manually refine the transition's type and corresponding time function and connect the CGSPN constructs to each other with the arcs. Table 2 shows the

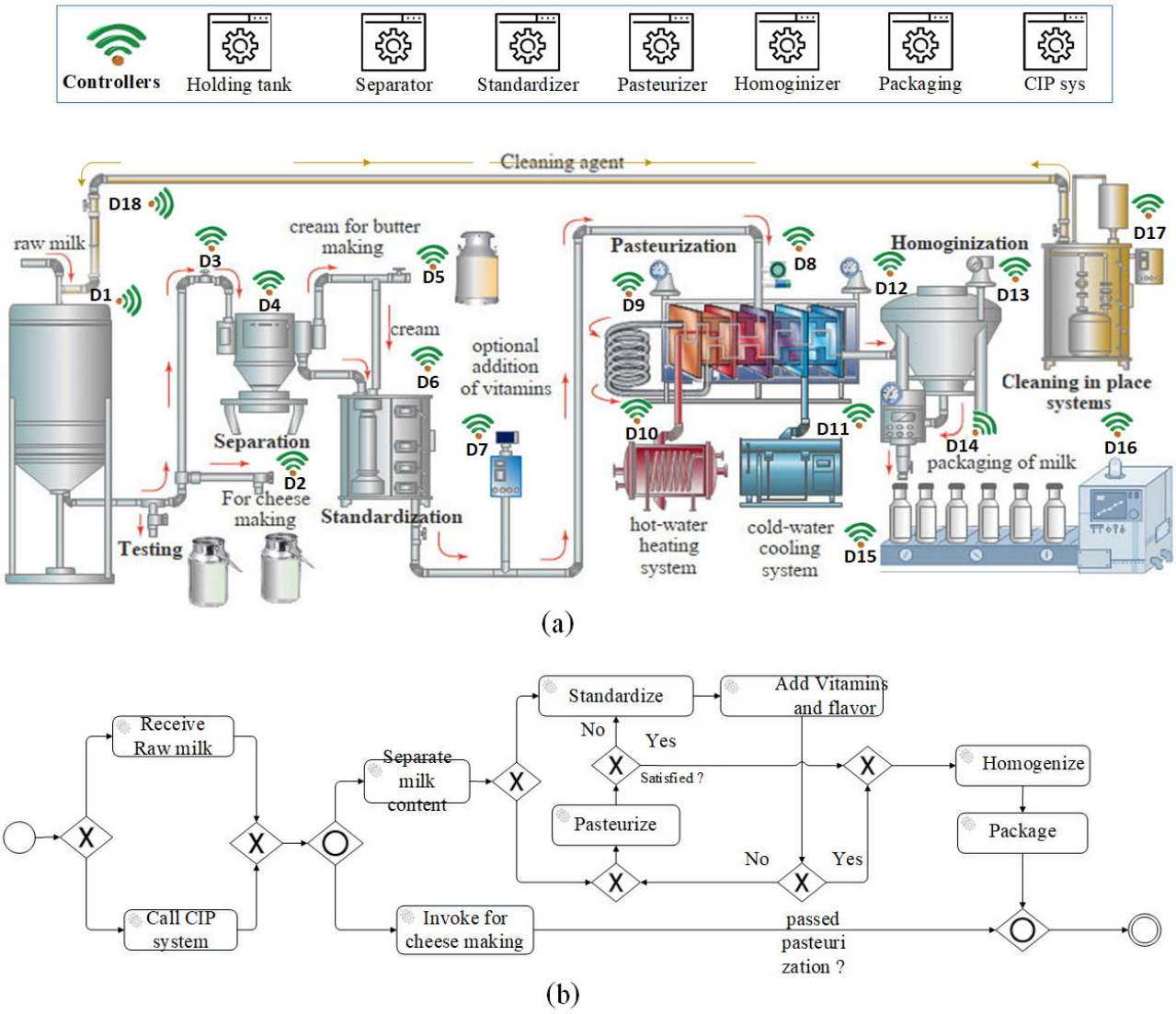

FIGURE 18. The topmost physical IoT process manufacturing and the dairy process manufacturing model depicted by using BPMN.

TABLE 2. Result of applying CP4BPMN tool with the proposed transformation rules to all existing dairy production subsystems.

No.	CGSPN Subnets	Places	Transitions
M1	Receive raw milk (D1)	12	9
M2	Cleaning system (D17-D18)	62	41
M4	Separation (D3-D4)	31	25
M5	Cheese making (D2)	10	9
M6	Standardization	13	5
M7	Pasteurization (D8-D12)	65	48
M8	Vitamins and flavor addition (D7)	13	5
M9	Homogenization (D13)	16	9
M10	Packaging (D15-D16)	21	12

TABLE 3. The CGSPN model of dairy production process after refinement.

Nodes	Before refinement	After Refinement
Place	217	225
Stochastic transition	29	29
Immediate transition	101	106
Deterministic transition	32	32
Scheduled transition	6	6
Arc	274	288
Inhibitor arc	14	14
Reset arc	0	0

CGSPN constructs derived from applying the CP4BPMN tool with the proposed transformation rules to all existing dairy production subsystems. The statistical report of the refined CGSPN model components is detailed in Table 3.

Figure 19 shows the obtained CGSPN model in part of the pasteurization process, which the cluster of the construct is composed of three parts highlighted by colored elements. The red construct is derived from the physical devices, and the black construct is derived from the controller, while the

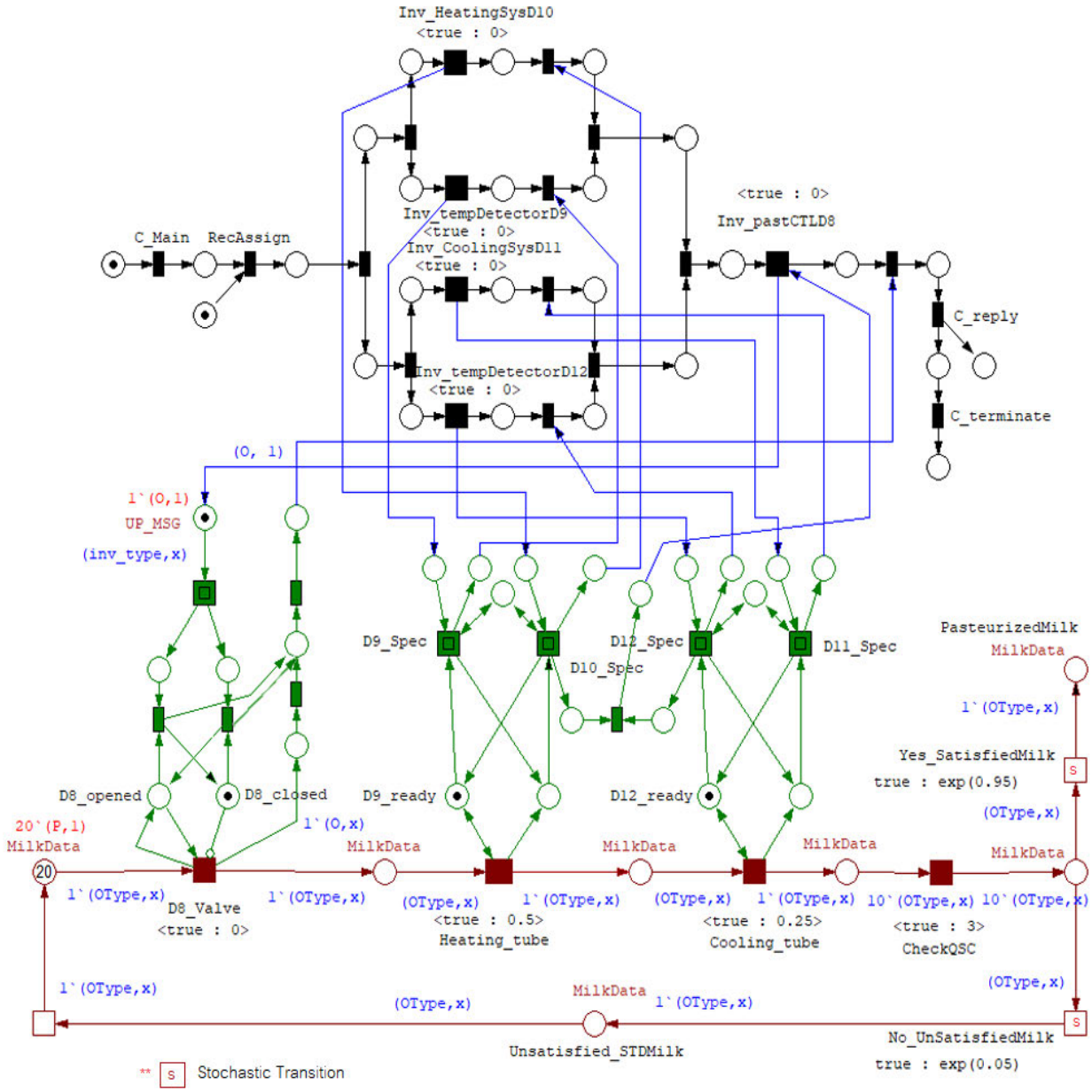

FIGURE 19. Excerpt refined CGSPN model of the milk pasteurization process.

green construct is derived from the IoT embedded software specifications. The blue arcs are the interface between the controller and the IoT embedded software. Due to the many inscriptions within the model, they are omitted to show full inscriptions.

The beginning of the pasteurization controller is node Main and consecutive nodes of Receive Request and AssignVar1 that are transformed into the CGSPN transition *C_Main* and *RecAssign*, respectively. The controller's information (*CTLData*) is passed by the arc inscriptions with variables. The communications between the controller and the IoT embedded software are represented by the refined blue arcs. For example, transition *Inv_pastCTLD8* has an outgoing arc sending data object *UP_MSG* to itself to command an IoT embedded software to perform an action by the physical device *D8_Valve*. After the valve performs an action, it will send the current state back to the controller every minute via the IoT embedded software.

At the red CGSPN construct of the physical device flow, the value of *MilkData* stores the information of raw milk and the operation type *OType*. The operation type is the kind of finished product, and x is the unit of milk in the pasteurization tube. The transition *D8_Valve* represents the physical device valve D8. It is enabled only when the temperature in the

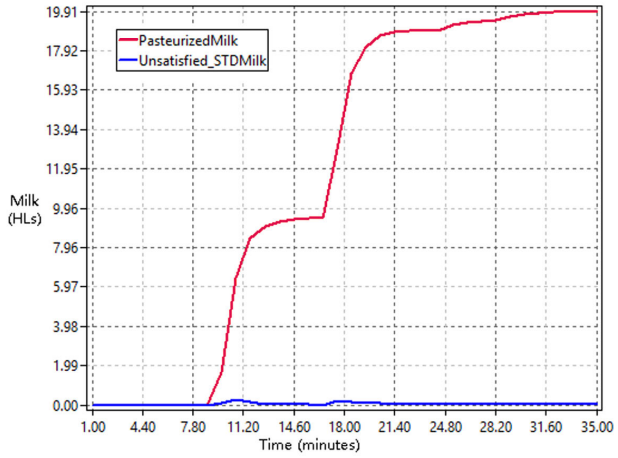

FIGURE 20. The simulation result of the CGSPN model in Figure 19. The pasteurized milk of satisfied standard will require 33 minutes for two thousand liters of raw milk.

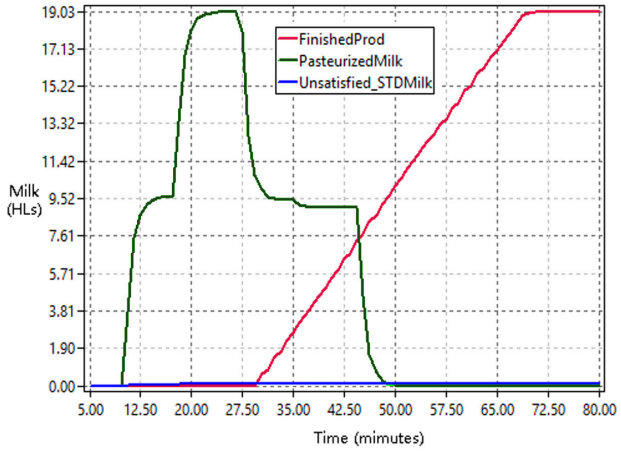

FIGURE 21. The simulation result of the CGSPN model from an overall perspective.

heating and cooling tube is under normal working of the pasteurizing devices (D9 - D12). The delay of transition *Heating_tube*, *Cooling_tube* with 0.5 and 0.25 represents that the raw milk requires 30 seconds in the heating tube and 15 seconds in the cooling tube. Next, the pasteurized milk will be collected in the holding tank for the pasteurizing standard measurement. The results of the measurement are represented by the stochastic transition with the transition firing rate of 0.95 for the milk satisfying the pasteurization standard and 0.05 for the milk that does not satisfy the standard. The random value of the unsatisfied standard milk is uniformly distributed between 0 and 5. It means that one thousand liters of the pasteurized milk is possible to have the unsatisfied standard milk not exceed fifty liters. In the case that the pasteurizer status is not ready to work or in the event of the temperature in the holding tubes not satisfying the lookup table configured, the CGSPN construct of the IoT embedded software of thermometers (D9, D12) will send commands to the controller to close valve D8 to pause the milk circulation.

To simulate the CGSPN model of the pasteurization process, we determine the initial markings with twenty tokens representing two thousand liters of milk. The number of simulation runs is 1000. This parameter value comes from the number of data records of the system runs. The system runs reports reveal the system outcomes consisting of the number of satisfied and unsatisfied pasteurization standard milk, the system issues and so on. The comparison of the number of satisfied and unsatisfied pasteurization standard milk from the simulation reports and that of the existing system logs report help us indicate whether the CGSPN models are realistic or not. The plot simulation results are shown in Figure 20. The graph shows the amount of milk satisfying the pasteurization standard and that of the unsatisfied pasteurization standard, which the curves are the average metrics of 1000 simulation runs. The pasteurizer requires approximately 16 minutes for a thousand liters of milk. Thus, the pasteurizer requires approximately 33 minutes for two thousand liters of milk. Two thousand liters of milk is determined for simulating the case of overloaded milk streaming into the pasteurizer and holding tank with the capacity of one thousand liters. The report shows the capacity of the pasteurizer, which conforms to the actual capacity of the existing system. From the simulation report, we observe that the CGSPN model is quite realistic, because the number of satisfied and unsatisfied pasteurization standard milk of the model simulation and existing report is agreeable. Given the assumption that IoT can reduce the gap and time between subsystems of the heating and cooling systems, we believe that, after the transformation of the existing pasteurizer to the IoT pasteurizer, the percentage of unsatisfied pasteurization standard milk may become quite close to zero. Thus, we adjust the firing rate of the transitions by decreasing from 0.05 to 0.02 for unsatisfied standard milk and increasing the firing rate from 0.95 to 0.98 for satisfied standard milk. The simulation result shows that the pasteurizer takes less than 2 minutes for two thousand liters of milk or 1 hour 30 minutes for a day.

To verify the overall system model, we partition the target model into the subnets following the subsystems shown in Table 3 and separately simulate and incrementally compose a subnet to verify them from an overall perspective. Each subnet coming from the subsystem will be independently refined based on the dependencies among the subnets. For example, the subnet of the pasteurization process receives the milk from a tank of the separation process. Thus, the data dependencies of the pasteurization processes are the amount of milk in the separation tank and the flow-out valves of the separation tanks. We manually refine the CGSPN model based on these constraints, including the addition of the CGSPN construct the user tasks or the other tasks that did not appear in the specification models to complete the CGSPN model.

Figure 21 shows the simulation result of the CGSPN model from an overall perspective. An initial marking is determined

TABLE 4. Comparison of contributions in related works and our work.

	Zhang, Wang et al [13]	Denno, Dickerson et al [10]	Zhang, Wang et al [16]	Zhou, Wang et al [17]	Our approach
Objective	To provide the production performance analysis technique.	To produce the model useful for line balancing and job sequencing.	To monitor and control the shop flow (real-time sensor).	To provide a simulation tool for simulating the stochastic timed Petri nets.	To provide a framework for automating and analyzing the IoT process manufacturing.
Formal model/ Language	HTCPN, Decision Tree	CPN, GSPN, PNN, Genetic programming	CPN	STPN	CGSPN
Model checking tool	CPN tool	MJPdes simulation engine	CPN tool, CPN-based prototype	STPN-based framework	CP4BPMN, Snoopy
Level of abstraction	Low-level	Low-level	Low & high level	Low-level	Low & high level
Verification approaches					
Type of analysis	Real-time	Model based	Real-time	Model based	Model based
Qualitative verification	N/A	No	N/A	Deadlock, Soundness	Deadlock, Soundness, consistency
Quantitative verification	Yes	Yes	Yes	Yes	Yes
Hierarchical verification	Yes	No	No	No	Yes
Partial verification	No	No	No	No	Yes
How to create the formal model					
Inputs of the formal model abstraction	No	System logs	No	N/A	BPMN
Automatic abstraction	No	Partial	No	N/A	Yes
Layer or module of a formal model	Controller	Controller	Middleware	Controller	All
Up-to-date model	No	Dynamic adaptation	No	N/A	No
Modules/Behaviors					
Stochastic process	Yes	Apply the random mutation operator	No	Yes	Yes
Sensor management	No	No	Yes	N/A	N/A
Queue	Yes	Yes	No	Yes	Yes
Priority	No	Yes	No	Yes	Yes
Advantages	1) The exception diagnosis model that is represented as a multi-level and multi-type of events.	1) The obtained model can manage the exceptional events meticulously.	1) The proposed technique is appropriate to manage and configure the multiple sensors.	1) The transition firing can be determined as race and preselection policies.	1) The CGSPN model is automated from the semi-formal models.
	2) The model acts as a controller for managing the heterogeneous IoT devices.	2) The update of the model as the situation on the production floor changes is viable technique for the dynamic production system.	2) The application services seem to be readable, and they work based on the sensors sent from the actual devices.	2) Model simulation supports multi-server transition (continuous firing).	2) Three layers of the obtained CGSPN model are easier representation and advocate the hierarchical and partial verification.

at the beginning of the production process, and the amounts of the finished dairy products are monitored at the end of the packaging process. It can be observed that the first finished dairy product will be produced at the 30th minute and completely packaged at the 70th minute for two thousand liters of raw milk. This simulation did not include the consumption time of the user tasks for the gravity separation of 1 day and the microbiological checking of 3 hours. We calculate the electric power consumption based on the times used of an overall perspective by the comparing it with the times used of the existing system. The results show that the production time decreases by 15 minutes. It represents that the productivity increases by 2.97 % and the product cost in the part of electrical power decreases by 3.12 % a year.

We compare the relevant contributions of the four related works and proposed approach. Table 4 shows the comparison details: ''Yes'' means that the technique supports the contribution and ''No'' is opposite. ''N/A'' is that the authors did not detail but we believe it can handle or support the determined contribution because of the capability of formal modeling languages or verification tools.

VI. CONCLUSION

IoT technologies can be applied to the existing system to reduce the operational gaps, time and flaws of the process manufacturing control. These technologies can use BPMN to detail the orchestration and composition of the heterogeneous IoT devices. Model checking can be used to verify the

IoT design models from the perspectives of both qualitative and quantitative verification. But the time and stochastic processes cannot be designed by using the existing BPMN notations. This work extended Eclipse BPMN2 modeler and transformation rules of CP4BPMN to automate BPMN notation into a CGSPN construct. The stepwise quantitative verification processes and framework are provided for modeling, simulating and analyzing the process's performance. The design models consisting of three clusters (controller, IoT embedded software and physical manufacturing flow) are transformed into CGSPN models.

We validate proposed framework by using the case study of the process manufacturing of dairy product factory. All three clusters are transformed by the extended CP4BPMN tool. The obtained CGSPN models are imported and refined in the model checking tool named Snoopy. In the parts of the IoT embedded software, we used equivalence class partitioning techniques to determine the considerable input and output cases in order to bound the input and output range of CGSPN construct. The CGSPN model is simulated based on the parameters automated from the BPMN process model. From the case study, the experimental results show that the productivity of the whole dairy manufacturing process is increasing by approximately 3 %, and the timed gaps between subprocesses are significantly decreasing. This observation indicates that the IoT dairy manufacturing design model exhibits not only increasing productivity but also decreasing production cost in the aspect of electrical power.

Based on the results of validation, our framework is a viable option that assists the system analysts and software modelers who need to analyze the performance on a time and stochastic BPMN process model. The CGSPN models support qualitative verification, performance analysis and flow animation. The shortcoming is that the model checking still remains difficult to perform for large model verification because Snoopy is time-consuming to check the model syntax before simulation, and the queueing and prioritizing management is using non-preemptive scheduling. Our ongoing work is directed towards an enhancing and prioritizing management of the controller to be preemptive scheduling, and will apply LSTM network for the faults prediction.

ACKNOWLEDGMENT

The authors would like to thank Wang Nam Yen Dairy Cooperative Ltd., Thailand, which gave advice regarding the dairy production process.

REFERENCES

[1] E. Johann, E. Panagos, and M. Rabinovich, "Time constraints in workflow systems," in *Seminal Contributions to Information Systems Engineering*. Berlin, Germany: Springer, 2013, pp. 191–205.

[2] L. Henrik, M. Niepert, M. Weidlich, J. Mendling, R. Dijkman, and H. Stuckenschmidt, "Probabilistic optimization of semantic process model matching," in *Proc. Int. Conf. Bus. Process Manage.*, 2012, pp. 319–334.

[3] F. Dieter, H. Lausen, A. Polleres, J. D. Bruijn, M. Stollberg, D. Roman, and J. Domingue, *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Berlin, Germany: Springer-Verlag, 2006.

[4] O. M. G. OMG, "OMG unified modeling language TM (OMG UML) version 2.5," Object Manage. Group, Needham, Massachusetts, USA Tech. Rep. Formal, Mar. 2015.

[5] M. Huth and M. Kwiatkowska, "Quantitative analysis and model checking," in *Proc. 12th Annu. IEEE Symp. Log. Comput. Sci.*, Jun. 1997, pp. 111–122.

[6] K. Jensen and G. Rozenberg, *High-Level Petri Nets: Theory and Application*. Berlin, Germany: Springer, 2012, pp.219–235.

[7] N. Gharbi, C. Dutheillet, and M. Ioualalen, "Colored stochastic Petri nets for modelling and analysis of multiclass retrial systems," *Math. Comput. Model.*, vol. 49, nos. 7–8, pp. 1436–1448, Apr. 2009.

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

[9] S. O. El Mehdi, R. Bekrar, N. Messai, E. Leclercq, D. Lefebvre, and B. Riera, "Design and identification of stochastic and deterministic stochastic Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 4, pp. 931–946, Jul. 2011.

[10] P. Denno, C. Dickerson, and J. A. Harding, "Dynamic production system identification for smart manufacturing systems," *J. Manuf. Syst.*, vol. 48, pp. 192–203, Jul. 2018.

[11] W. B. Langdon and R. Poli, *Foundations of Genetic Programming*. Berlin, Germany: Springer, 2013.

[12] M. Kudlek, "Probability in Petri nets," *Fundamenta Informaticae*, vol. 67, nos. 1–3, pp. 121–130, 2005.

[13] Y. Zhang, W. Wang, N. Wu, and C. Qian, "IoT-enabled real-time production performance analysis and exception diagnosis model," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 3, pp. 1318–1332, Jul. 2016.

[14] H. Z. Huang and X. Zu, "Hierarchical timed colored Petri nets based product development process modeling," in *Proc. CSCWD*, 2004, pp. 378–387.

[15] Y. Sheng and S. M. Rovnyak, "Decision tree-based methodology for high impedance fault detection," *IEEE Trans. Power Del.*, vol. 19, no. 2, pp. 533–536, Apr. 2004.

[16] Y. Zhang, W. Wang, W. Du, C. Qian, and H. Yang, "Coloured Petri net-based active sensing system of real-time and multi-source manufacturing information for smart factory," *Int. J. Adv. Manuf. Technol.*, vol. 94, nos. 9–12, pp. 3427–3439, Feb. 2018.

[17] J. Zhou, J. Wang, and J. Wang, "A simulation engine for stochastic timed Petri nets and application to emergency healthcare systems," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 4, pp. 969–980, Jul. 2019.

[18] 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.

[19] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "LTSA-WS: A tool for model-based verification of web service compositions and choreography," in *Proc. 28th Int. Conf. Softw. Eng.*, May 2006, pp. 771–774.

[20] H. Jahan, S. Rao, and D. Liu, "Test case generation for BPEL-based web service composition using colored Petri nets," in *Proc. Int. Conf. Prog. Informat. Comput. (PIC)*, Dec. 2016, pp. 623–628.

[21] D. Domingos, F. Martins, C. Cândido, and R. Martinho, "Internet of Things aware WS-BPEL business processes context variables and expected exceptions," *J. Univers. Comput. Sci.*, vol. 20, no. 8, pp. 1109–1129, 2014.

[22] C. Mi, H. Miao, J. Kai, and H. Gao, "Reliability modeling and verification of BPEL-based web services composition by probabilistic model checking," in *Proc. IEEE 14th Int. Conf. Softw. Eng. Res., Manage. Appl. (SERA)*, Jun. 2016, pp. 149–154.

[23] A. C. Franco da Silva, U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, B. Mitschang, and R. Steinke, "Internet of Things out of the box: Using TOSCA for automating the deployment of IoT environments," in *Proc. 7th Int. Conf. Cloud Comput. Services Sci.*, 2017, pp. 330–339.

[24] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "TOSCA: Portable automated deployment and management of cloud applications," in *Advanced Web Services*. Germany, Europe, Univ. Stuttgart, Inst. Archit. Appl. Syst., 2014, pp. 527–549.

[25] X. Li, Y. Fan, Q. Z. Sheng, Z. Maamar, and H. Zhu, "A Petri net approach to analyzing behavioral compatibility and similarity of web services," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 41, no. 3, pp. 510–521, May 2011.

[26] Z. J. Ding, J. L. Wang, and C. J. Jiang, "An approach for synthesis Petri nets for modeling and verifying composite web service," *J. Inf. Sci. Eng.*, vol. 24, no. 5, pp. 1–20, 2008.

[27] P. Xiong, Y. Fan, and M. Zhou, "QoS-aware web service configuration," *IEEE Trans. Syst., Man, A, Syst. Humans*, vol. 38, no. 4, pp. 888–895, Jul. 2008.

[28] T. Eterovic, E. Kaljic, D. Donko, A. Salihbegovic, and S. Ribic, "An Internet of Things visual domain specific modeling language based on UML," in *Proc. 25th Int. Conf. Inf., Commun. Autom. Technol. (ICAT)*, Oct. 2015, pp. 1–5.
[29] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *Proc. 33rd Int. Conf. Softw. Eng.*, May 2011, pp. 341–350.
[30] M. Chen, T. H. Tan, J. Sun, Y. Liu, J. Pang, and X. Li, "Verification of functional and non-functional requirements of web service composition," in *Proc. Int. Conf. Formal Eng. Methods*, 2013, pp. 313–328.
[31] M. Chen, T. H. Tan, J. Sun, Y. Liu, and J. S. Dong, "VeriWS: A tool for verification of combined functional and non-functional requirements of web service composition," in *Proc. Companion Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 564–567.
[32] É. André, T. Huat Tan, M. Chen, S. Liu, J. Sun, Y. Liu, and J. Song Dong, "Automated synthesis of local time requirement for service composition," 2020, *arXiv:2003.08116*.
[33] M. Schwarick, M. Heiner, and C. Rohr, "MARCIE–model checking and reachability analysis done efficiently," in *Proc. 8th Int. Conf. Quant. Eval. Syst.*, Sep. 2011, pp. 389–399.
[34] N. Yildirim and S. Genc, "Thermodynamic analysis of a milk pasteurization process assisted by geothermal energy," vol. 90, pp. 987–996, 2015, doi: 10.1016/j.energy.2015.08.003.
[35] M. R. Abdmeziem, D. Tandjaoui, and I. Romdhani, "Architecting the Internet of Things: State of the art," in *Robots Sensor Clouds*. Cham, Switzerland: Springer, 2016, pp. 55–75, doi: 10.1007/978-3-319-22168-7_3.
[36] P. Sethi and S. R. Sarangi, "Internet of Things: Architectures, protocols, and applications," *J. Elect. Comput. Eng.*, vol. 2017, Jan. 2017, doi: 10.1155/2017/9324035.
[37] J. S. Hurwitz, R. Bloor, M. Kaufman, and F. Halper, *Service Oriented Architecture (SOA) for Dummies*. Hoboken, NJ, USA: Wiley, 2009.
[38] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, Oct. 2003.
[39] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "QoS-aware replanning of composite web services," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2005, pp. 121–129.
[40] G. Ciardo, R. German, and C. Lindemann, "A characterization of the stochastic process underlying a stochastic Petri net," *IEEE Trans. Softw. Eng.*, vol. 20, no. 7, pp. 506–515, Jul. 1994.
[41] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks, "Uppaal 4.0," Dept. Inf. Technol. Uppsala Univ., Sweden, Uppaal, Tech. Rep. 4.0, 2006.
[42] M. Heiner, M. Herajy, F. Liu, C. Rohr, and M. Schwarick, "Snoopy—A unifying Petri net tool," in *Proc. Int. Conf. Appl. Theory Petri Nets Concurrency*. Berlin, Germany: Springer, Jun. 2012, pp. 398–407.
[43] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic model checking for performance and reliability analysis," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 40–45, Mar. 2009.
[44] K. Jensen and G. Rozenberg, *High-Level Petri Nets: Theory and Application*. Berlin, Germany: Springer, 2012.
[45] M. Li and N. D. Georganas, "Coloured generalized stochastic Petri nets for integrated systems protocol performance modelling," *Comput. Commun.*, vol. 13, no. 7, pp. 414–424, Sep. 1990.
[46] 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.
[47] P. Huber, K. Jensen, and R. M. Shapiro, "Hierarchies in coloured Petri nets," in *Proc. Int. Conf. Appl. Theory Petri Nets*, 1989, pp. 313–341.
[48] W. M. Van Der Aalst, K. M. Van Hee, A. H. Ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, "Soundness of workflow nets: Classification, decidability, and analysis," *Formal aspects Comput.*, vol. 23, no. 3, pp. 333–363, 2011.
[49] J. Wegener, M. Schwarick, and M. Heiner, "A plugin system for Charlie," in *Proc. CS&P*, 2011, pp. 531–554.
[50] M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic*. Hoboken, NJ, USA: Wiley, 2011.
[51] *BPMN2 Modeler Project*. Accessed: Dec. 29, 2021. [Online]. Available: https://www.wikibooks.org
[52] 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.
[53] C. Dechsupa, W. Vatanawood, and A. Thongtak. (2020). *Technical report: Mapping of BPEL and Examples*. [Online]. Available: https://www.researchgate.net/profile/Chanon_Dechsupa2
[54] 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.
[55] L. Fei, H. Monika, and R. Christian, "The manual for colored Petri nets in snoopy–QPN C/SPN C/CPN C/GHPN C," Computer Sci. Rep. Brandenburg Univ. Technol., Cottbus, Germany, Tech. Rep. Report02-12, 2012.
[56] C. Braunstein, A. E. Haxthausen, W. L. Huang, F. Hübner, J. Peleska, U. Schulze, and L. V. Hong, "Complete model-based equivalence class testing for the ETCS ceiling speed monitor," in *Proc. Int. Conf. Formal Eng. Methods*. Cham, Switzerland: Springer, Nov. 2014, pp. 380–395.

C. DECHSUPA received the B.S. degree in computer information system from Burapha University, Thailand, in 2008, the M.S. degree 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 a Senior System Analyst with many private sectors. His research interests include the formal method in software engineering and workflow design and an applying AI in the formal verification approaches.

W. VATANAWOOD received the Ph.D. degree in computer engineering from Chulalongkorn University, Thailand. He is currently an Associate Professor of computer engineering at the Faculty of Engineering, Chulalongkorn University. 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 at the Department of Computer Engineering, Chulalongkorn University, Thailand. His research interests include asynchronous logic design and verification, dependable computing, and computer architecture.