# Digital-Twin Consistency Checking Based on Observed Timed Events With Unobservable Transitions in Smart Manufacturing

Moon Gi Seok, *Member, IEEE*, Wen Jun Tan, Wentong Cai, *Member, IEEE*, and Daejin Park, *Member, IEEE*

***Abstract*—Smart factories manage digital twins (DTs) to evaluate the performance of various what-if production scenarios. This article presents a DT consistency-checking approach to maintain DT in high fidelity by checking whether each sensed timed event from the physical manufacturing plant is under its corresponding DT-based estimations in runtime. The approach targets DTs developed using time colored Petri net (TCPN). To build the candidates of the next observable event with observable time margins, we considered the stochastic property of the plant, frequent external actuation caused by a new order, machine maintenance, etc., as well as intermediate unobservable state transitions reaching the sensible events. Based on the considerations, we propose an iterative method to build the virtual estimates for streaming physical events using efficiently evolved state-class graphs (SCGs). We also propose a TCPN partitioning method to accelerate the SCG-evolution and make DT maintenance easier by supporting the isolation of inconsistent subnets being diagnosed. We applied the approach to a USB flash-drive factory to prove the concept and evaluated the performance under various situations to show speedups of the SCG evolution, that is the crucial overhead of the estimation.**

***Index Terms*—Digital twin (DT), manufacturing system, reachability analysis, state-class graph (SCG), time petri net (TPN).**

Manuscript received 1 April 2022; revised 19 July 2022; accepted 17 August 2022. Date of publication 22 August 2022; date of current version 22 March 2023. This work was supported in part by the A*STAR Cyber-Physical Production System (CPPS) – Towards Contextual and Intelligent Response Research Program, under the RIE2020 IAF-PP Grant A19C1a0018, in part by NRF under Grant NRF-2018R1A6A1A03025109, and in part by IITP under Grant 2021-0-00944 and Grant 2022-0-01170. Paper no. TII-22-1404. *(Corresponding author: Daejin Park.)*

Moon Gi Seok was with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798. He is now with the Department of Artificial Intelligence, Dongguk University Seoul 04620, Korea (e-mail: mgseok@dgu.ac.kr).

Wen Jun Tan and Wentong Cai are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: wjtan@ntu.edu.sg; ASWTCAI@ntu.edu.sg).

Daejin Park is with the School of Electronics Engineering, Kyungpook National University (KNU), Daegu 702-701, Korea (e-mail: boltanut@knu.ac.kr).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3200598.

Digital Object Identifier 10.1109/TII.2022.3200598

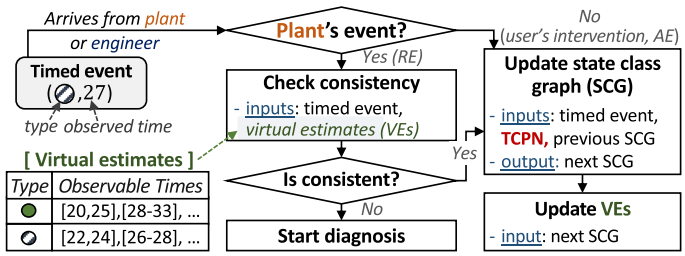



Fig. 1. Checking TCPN-based DT using each timed event.

## I. Introduction

DIGITAL twin (DT) is essential for smart factories to maximize manufacturing plants' (or shop floors') production performance, such as throughput and cycle time, after virtually commissioning various what-if scenarios. For reliable performance estimation, it is crucial to maintain high-fidelity DT that reflects the dynamics of part (or material) flows in the high-mix/low-volume manufacturing lines. For that, DTs should be developed considering various production behaviors of the target plant's components, such as machines, conveyors, human operators, and automated guided vehicles (AGVs). Due to a modeling error, an unreported physical change, or an unmodeled exception, a DT-based prediction might become inaccurate. This article focuses on a DT consistency check that verifies DT using physical events sensed from the plant along with their observed times (called *timed* events). Sufficient fidelity in DT can be guaranteed by iterative synchronization of DT with its physical counterpart throughout the manufacturing life cycle.

To reflect the production dynamics, DT is modeled using discrete-event models (DEMs). DEMs can describe the dynamics of the plant's operations using state transitions and exchanges of the events generated by the transitions. Examples of the events include part loading/unloading, AGV arrival/departure, machine process start/end, setup or operation mode changes, fault detection, and fault recovery start/end. Each physical timed event has corresponding *virtual estimates*, which comprises multiple event types and their observable time margin, as Fig. 1 shows. The consistency between the plant and its DT is examined iteratively by checking whether each event is consistent with its virtual estimates. If all state transitions and related events are observable and the plant's production is deterministic, building

the estimates might be straightforward. However, computing the virtual estimates presents the following three main challenges.

1) Some state transitions that do not produce physical events (which can be monitored online) are unobservable due to the lack of monitoring or communication tools for legacy machines or human operators.
2) There are stochastic state transitions and state-duration variances due to faults in the machines, human operators, or AGVs. For example, a machine's arm can locate a part in the correct position after several attempts, or an obstacle can suspend a moving AGV for a while.
3) External interventions from the control facilities (CFs) to the plant can occur to handle new orders or manage machines for energy efficiency. We aim to compute the virtual estimates of the plant's next physical event based on reachable observable transitions, considering all possible sequences of intermediate unobservable transitions, stochastic state transitions and variable operation periods, and external interventions.

For the reachability analysis, researchers have identified reachable transitions (or states) based on the states' time ranges and all possible transitions (including unobservable transitions) using time Petri net (TPN) [1], [2], [3], [4], [5]. In the studies, various state-class graphs (SCGs) and their evolution methods for the analysis have been developed. In this article, we propose a DT-modeling formalism of a time-colored Petri net (TCPN) to apply the TPN-based analysis methods to high-mix manufacturing plants. It is designed by extending TPN to use colors that distinguish tokens representing various types of parts. By extending a modern TPN-based SCG evolution method, we propose a runtime checking method to update SCG and compute the virtual estimates of the plant's next event based on the recent physical event (for further consistency checking), as Fig. 1 shows. We denote the CF's external interventions as actuation events (AEs) and the plant's sensed events as reactive events (REs). DT's observable transitions can be detected online using associated REs and AEs. Assuming AEs are unpredictable, only REs are used to check DT consistency. AEs are only used to revise SCG to update the virtual estimates.

We summarize the main contributions as follows.

1) We propose an approach to check the consistency of TCPN-based DTs during runtime using streaming timed physical events (monitored online).
2) We extend TPN-based SCG-evolution methods to estimate the next RE, considering AE's unpredictability.
3) We propose a TCPN partitioning method based on the AE's property (unpredictability), which improves the overall SCG evolution computational performance and supports the isolation of problematic subnets (which can examine other consistent subnets during the problem diagnosis).
4) We propose methods to update the SCG iteratively with newly observed event to maintain DT's consistency.

The rest of the article is organized as follows. Section II introduces the related works about TCPN and TPN-based reachability analysis. Section III defines the formalism of TCPN. Section IV specifies the problems related to SCG in the runtime checking. Section V details the overall methods for constructing the SCG and estimating the next REs. Section VI shows the TCPN partitioning method and SCG revision according to the observed AE/RE. Section VII presents a group of experiments as a case study. Finally, Section VIII concludes this article.

## II. Related Works

Smart manufacturing systems can be generally structured in three layers: 1) enterprise resource planning (ERP); 2) manufacturing-execution system (MES); 3) industrial control system (ICS) [6]. At the top level, ERP focuses on integrating organizational functions to provide forecasting and planning, inventory management, and accounting functionalities. In the middle level, MESs have been used to collect and manage the information from manufacturing plants and helps decision makers understand the current status and improve productivity [7]. The decision making requires the evaluations of various what-if manufacturing scenarios using the plant's high-fidelity DT. Our work is meant to increase DT's fidelity by checking the consistency between DT and monitored physical events. At the lowest level, ICSs are used to control and monitor specific equipment.

In previous works, researchers have examined the consistency between DT and physical information to detect abnormal behaviors in the ICS's security domain [8], [9], [10]. They considered DT as a reference model and noticed the adversary's attacks by detecting the inconsistency between physical and virtual states. Compared to the works from the security domain with our method, we aim to check DT's consistency considering unobservable transitions, stochastic state transitions, stochastic operation periods, and users' interventions (which are AEs).

Petri net (PN) has been widely used to describe plants' event-driven behaviors for various purposes, including throughput measurement [11], [12], supervisory control [13], [14], and deadlock-free scheduling [15], [16], because of its advantages in modeling concurrent and synchronization operations. PN has been extended to include various features and functions for specific modeling purposes, such as colored PN (CPN) [17], [18], queueing PN [19], and time PN (TPN) [20], [21] or timed PN [22]. CPN adds an attribute to tokens as color to distinguish between multiple production sequences for different products in a complex plant. Some CPNs can also support the primitives to model specific data manipulation. TPNs or timed PNs help analyze the performance of timed systems.

The main distinction between TPNs and timed PNs is whether each enabled transition fires within a given interval or a given delay value, respectively. We represent the plant's stochastic state durations as intervals based on measured means and standard variations (see Section III). Then, we propose a new modeling formalism of TCPN to: 1) combine TPN's interval-based transition firing; 2) employ CPN's colors (for high-mix manufacturing) and AEs (for users' interventions); 3) reference previous TPN-based reachability-analysis techniques [to build the virtual estimates by identifying all reachable states (or transitions) that generate the REs]. The TCPN's formalism was not previously defined to the best of our knowledge.

Berthomieu and Diaz [1] were the first to propose an abstracted TPN state, called state classes (SCs), to represent the

finite or infinite sets of the TPN's timed states. They presented a graph called a SCG-based on the SCs. The SCG's nodes are the set of SCs, and each edge shows the consequence of a labeled transition execution (or firing) from one source SC to another. Some studies have focused on better abstractions (reducing the number of SC nodes) by proposing a relaxed SCG, contracted SCG, or fault-diagnosis graph (which adds the sequence of transition information to the edges and removes the unnecessary SCs) [2], [3]. In [4], Basile et al. presented a modified SCG (MSCG) that extends edge labels using additional transition information related to timing variables and constraints. Using the MSCG, they proposed a fault-diagnosis method that requires solving a linear programming problem to detect the occurrence of a fault-related transition for a given sequence of observed events. In [5], He et al. showed that the previous MSCG evolution might miss some reachable states. Therefore, they consolidated the MSCG evolution method by defining *deficient SCs* that can generate new paths (series of SCs) using new subsequent SCs.

SCGs show all reachable SCs caused by possible observable and unobservable transition firings with their range-based time intervals. Researchers have typically utilized SCGs for fault diagnoses that check whether any fault transition might occur based on a given sequence of observed events. The proposed approach shares the existing TPN-based methods' principles in terms of SCG evolution but requires a revision based on TCPN. Moreover, when we use existing SCG-related methods for runtime DT checking, we need additional procedures, which are: 1) building the virtual estimates based on the SCG; 2) updating SCGs iteratively for streaming physical events. An SCG should be revised for the subsequent event estimation in runtime checking whenever a new physical event arrives. For the SCG update, we compute consistent SCs (which become roots for the next SCG) based on the previous SCG and each physical timed event. Among various types of existing SCGs and their evolution methods, our work uses MSCG (in [4] and [5]) because MSCG's timing information annotated with its edges is essential to derive consistent SCs.

For runtime checking, speedup of SCG-evolution (which is the main estimation overhead) is essential because early inconsistency guarantees better problem diagnosis by dispatching human operators to a problematic physical location. Thus, it is recommended to build the virtual estimates using the updated SCG before a new physical event arrives. For the evolution speedup, we resolve computational inefficiency problems of the latest MSCG evolution method in [5] (discussed in Section IV-A) and propose a TCPN partitioning method.

The following section defines a TCPN's formalism for the formal description of our approach and the plant's TCPN-based DT modeling.

## III. Time Colored Petri Net

TCPN is a combination of TPN and CPN for: 1) TPN's transition-associated time intervals; 2) CPN's token distinction. In our work, the tokens represent high-mix parts or other part-flow control signals (e.g., machine availability, delivery request/grants, etc.). Transitions represent part-specific or joint

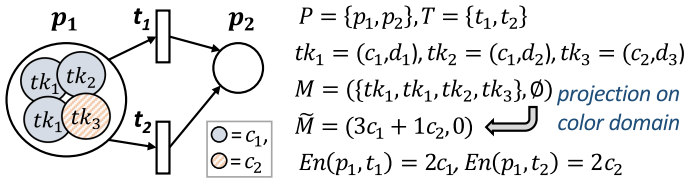



Fig. 2. Example of a logically enabled transition ($t_1$).

operations (with their range-based time duration) for machine operations, flow-control decisions, AGV movements, etc. We simplify a transition's logically enabling condition of TCPN as the required number of colors in the transition's backward places, similar to TPN's enabling condition (the required number of tokens). For that, we represent a token, $tk$, as a pair of colors (for the transition's token distinction) and data (which can be empty), i.e., $tk \in \mathcal{C} \times (\mathcal{D} \cup \varnothing)$, where $\mathcal{C}$ is a set of colors and $\mathcal{D}$ is a set of possible manufacturing-related data. Then, it requires an additional color decision when the tokens are fired for the subsequent transition (if needed).

Based on the concept, we define a TCPN's formalism as follows:

*Definition 1:* The net of TCPN is represented as $N = \langle P, T, En, Pre, Post, Q, \mathcal{L} \rangle$, where

1) $P$ is a set of places.
2) $T = T_{\text{re}} \cup T_{\text{ae}} \cup T_{\text{si}}$ is a set of transitions, where $T_{\text{re}}/T_{\text{ae}}$ is a set of observable transitions using their associated REs/AEs and $T_{\text{si}}$ is a set of silent transitions.
3) $En : P \times T \to (\mathcal{C} \times \mathbb{N}_0)^{|\mathcal{C}|}$ is an enabling-condition function to specify the required numbers of colors for a given transition $t \in T$ from a place $p \in P$ to be logically enabled.
4) $Pre : \{M\} \times T \to \{X^*\}$ is a function to show one or multiple candidates of inflowing tokens when a transition $t \in T$ is fired at the given marking $M$; a marking $M : P \to \{K\}$ is a vector that assigns each place a multiset of tokens; $K$ is a multiset of tokens, and $X : P \to \{K\}$ is a vector of tokens inflowing to $t$ from each place.
5) $Post : \{X\} \times T \to \{Y\}$ is a function to compute a vector of outflowing tokens $Y : P \to \{K\}$ based on a given vector $X$ when transition $t \in T$ fires.
6) $Q$ is a timing function that defines the set of static closed time intervals only for predictable transitions ($T_{\text{re}} \cup T_{\text{si}}$) as $Q : T_{\text{ae}} \to \emptyset$ and $Q : T \setminus T_{\text{ae}} \to \mathbb{Q} \times (\mathbb{Q} \cup \{\infty\})$;
7) $\mathcal{L} : T \to \{\lambda\} \cup \epsilon$ assigns an RE's or AE's type (using a letter, $\lambda$) to each observable transition, $t \in T_{\text{re}} \cup T_{\text{ae}}$, or the empty string $\epsilon$ to each silent transition $t \in T_{\text{si}}$.

Let $\tilde{M}$ be a conversion from marking $M$ after projecting the tokens onto the color domain, as Fig. 2 shows. We denote $\tilde{M}(p)$ as the numbers of token colors in place $p$. A transition $t(\in T)$ is logically enabled if sufficient colors are present in $\tilde{M}$, i.e., $\tilde{M} \geq En(\cdot, t)$, where $En(\cdot, t)$ is the vector of $(En(p_1, t), \ldots, En(p_m, t))$, $p_i$ is the $i$th place, and $m = |P|$. The inequality $\geq$ denotes the relationship: $\forall p_i \in P, \tilde{M}(p_i) \geq En(p_i, t)$. We denote the set of transitions logically enabled at $M$ as $\mathcal{A}(M)$, i.e., $\mathcal{A}(M) = \{t \in T \mid \tilde{M} \geq En(\cdot, t)\}$. This color-based enabling description can make the same-colored

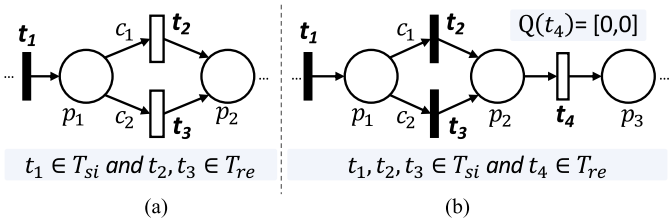



Fig. 3. TCPN examples for different operations ($t_2$ and $t_3$) whose completions lead to the same-type event. (a) Compact TPN. (b) Revised TPN with extra $t_4$ and $p_3$.

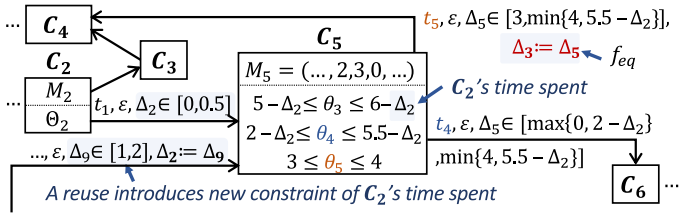



Fig. 4. Examples of time-spent variable ($\Delta$) usages and a node reuse of MSCG (defined in [5]).

tokens with different data move through a shared transition, which helps represent a common process of high-mix parts.

The function $Pre$ is developed considering the corresponding actual system. Let us suppose that a machine randomly selects two parts (which require a $t_1$-related operation) from its buffer, as Fig. 2 shows. Then, $Pre$ is developed for $Pre(M, t_1) = \{X_1, X_2\}$, where $X_1 = (\{tk_1, tk_1\}, \emptyset)$, and $X_2 = (\{tk_1, tk_2\}, \emptyset)$. If the machine selects two parts by the arrival order and token $tk_2$ arrived earlier than $tk_1$, then tokens should contain the arrival-order information in their data and $Pre(M, t_1)$ is equal to $\{X_2\}$.

The function $Post$ can forward each token $tk$ in a given $X$ to the next place after the token's color or data conversion. $Post$ can also drop or generate tokens on purpose. When a token vector, $X \in Pre(M, t)$, is moved by the firing of a transition $t$, $M$ is changed to the next marking, $M' = M \ominus X \oplus Post(X, t)$, where $\ominus$ and $\oplus$ are the element-wise set operations of $\setminus$ and $\cup$ on two vectors, respectively.

The function $Q$ does not return any interval for AE-related transitions because of its unpredictable occurrence. Therefore, an AE-related transition, $t \in T_{\text{ae}}$, is fired only by its real AE observation. For the reactive transitions (which are $T_{\text{re}} \cup T_{\text{si}}$), $Q$ indicates the intervals of those transitions using two rational numbers, namely $Q(t) = (l, u)$, where $l \geq 0$ and $u \geq l$. In our work, a logically enabled transition, $t \in T \setminus T_{\text{ae}}$, must be fired within the interval as a constraint unless it becomes disabled. The constraint is called *strong time semantic* (STS) in [4]. The time interval of $Q$ is decided based on its corresponding operation spans, which are reported online or measured manually. The interval of a transition $t \in T \setminus T_{\text{ae}}$ is represented as $Q(t) = [\hat{s}_i - k\sigma_i, \hat{s}_i + k\sigma_i]$, where $\hat{s}_i$ and $\sigma_i$ are the means and standard variation of measured spans of a $t$-related operation, respectively, and $k$ is a user's empirical constant, such as 3 (for the three-sigma rule of thumb).

We assume that each observable transition in $T_{\text{re}} \cup T_{\text{ae}}$ does not share its label with any of the other transitions. This assumption makes us find a specific transition corresponding to an event type $\lambda$, i.e., $t = \mathcal{L}^{-1}(\lambda)$. Suppose that a machine processes two steps: a common step (denoted by the transition $t_1$) and a variable token-dependent step (among two tasks symbolized by $t_2$ and $t_3$) and we can only detect the completion of the second step using an infrared sensor, as Fig. 3(a) shows. To prevent the duplication s.t. $\mathcal{L}(t_2) = \mathcal{L}(t_3)$, we should model this situation as Fig. 3(b) shows by adding the synthetic step $t_4$.

## IV. Problem Statement in SCG-Based DT Checking

Our approach is intended to check DT's consistency during runtime via three methods: 1) evolving the SCG (to build the next RE's virtual estimates) based on TCPN; 2) splitting the TCPN (for fast SCG evolution and subnet isolation); 3) reupdating the SCG based on each physical event. In this section, we describe each methods' SCG-related problems.

### A. Background and Problems of SCG Evolution

As discussed in Section II, we uses the MSCG in [4] and [5] as a default SCG. First, we introduce the background of MSCG and its evolution. Then, we describe the latest evolution algorithm's computational inefficiencies. Last, we clarify additional requirements for runtime TCPN checking.

The SCG is a directed graph whose nodes are called SC. Each SC is associated with the net's state and represented as a pair of a reachable marking ($M$) and a set of inequalities ($\Theta$) that define the timing constraints of logically enabled transitions at $M$, i.e., $C_k = (M_k, \Theta_k)$, where $C_k$ is the $k$th SC. Each inequality in $\Theta$ means the remaining time before its associated transition's ($t$'s) firing if $t$ does not become logically disabled due to a token preemption by another transition firing. The inequality can depend on a certain number of variables, denoted $\Delta$ variables, which consider how long a transition has been enabled. Let $\theta_i \in \Theta$ be transition $t_i$'s timing constraints. Then, we generally represent $\theta_i$ as

$$\theta_i = [l_i^\circ, u_i^\circ] = \left[ l_i - \sum_{\ell=1}^{d_i} \Delta^{(-\ell)}, u_i - \sum_{\ell=1}^{d_i} \Delta^{(-\ell)} \right] \quad (1)$$

where $[l_i, u_i] = Q(t_i)$; $\Delta^{(-\ell)}$ is the time spent of the $\ell$th previous SC $C^{(-\ell)}$; and $d_i$ is the number of intermediate SCs after $t_i$ was newly enabled. In the case of SC $C_5$ (which is one of $C_2$'s next SCs) in Fig. 4, $\Delta^{(-\ell)}$ and $d_3$ of $\theta_3$ ($\in \Theta_5$) are $\Delta_2$ (that is, $C_2$'s time spent) and 1, respectively. This indicates that transition $t_3$ was newly logically enabled at $C_2$. No timing variable in $\theta_5 \in \Theta_5$, i.e., $d_5 = 0$ means that transition $t_5$ is newly logically enabled at $C_5$.

Each edge $e$ of SCG has an extensible label $\mathfrak{L}(e) = \langle t_i, \mathcal{L}(t_i), \Delta_k \in [l_k^*, u_k^*], \ldots \rangle$, where $t_i$ is the transition whose firing leads to the target SC's marking; $\mathcal{L}(t_i)$ is $t_i$'s label; and $\Delta_k$ is a constraint of the source SC's time spent. The constraint bounds $l_k^*$ and $u_k^*$ are functions of $\Delta$ variables, as Fig. 4 shows. As in [5], the label can include a function, $f_{\text{eq}} : \Delta_n \to \Delta_k$, if

an existing SC can be reused as the edge's next after one or multiple existing time-spent variables $\{\Delta_k\}$ (depicted in the edge's source) are renamed with their pairs $\{\Delta_n\}$ of the *reusable* SC. In Fig. 4, $C_5$'s forwarding edge contains $f_{\text{eq}}$ to reuse $C_3$ after renaming $\Delta_5$ as $\Delta_3$. The reuse can reduce the evolution overhead because the reused SC will not be further explored if it is not a deficient SC. Definition 1 in [5] defines the deficient SC, which could generate a new path from the deficient SC when a new deficient SC reuse (for a new path reaching the deficient SC) lessens a firing constraint of a logically enabled but previously preempted transition (called *deficient* transition) enough to be fireable.

Fig. 4 depicts the deficient SC example, which is $C_5$, because it has the *deficient* transition $\theta_3$ preempted by the prior transition $t_5$ based on $\max(\theta_5) < \min(\theta_3)$. However, if $C_5$ is reused and its new backward edge contains a time spent, such as $\Delta_2 \in [1, 2]$, then $t_3$ can be fired first as an *imminent* transition, which will generate a new subsequent SC.

Based on the concept of SCs' deficiency and reuse, the previous SCG evolution algorithms in [5] have the following two computational inefficiencies:

1) Whenever an SC (from a queue) is called, imminent transitions (which can be fired before the others) are computed based on all the previous paths reaching the SC (as Algorithm 2 in [5]).
2) All deficient SCs are called whenever the queue for the waiting SCs (to be evolved) is empty to find possible new paths (as Algorithm 1 in [5]).

Because deficient SCs can be called multiple times, computing the imminent transitions based on all previous paths can lead to serious computational overhead. The iterative checking of all deficient SCs is inefficient because most deficient SCs would not lead to new paths as the SCG evolution cycle elapsed. To prevent each SC's all-path checking, the proposed method schedules each SC with its new paths and timing information, defined as follows:

*Definition 2:* Given an SC $C_k$ in the queue, $C_k$ has its own new paths $\Pi_k^+$ and related previous SCs' time-spent intervals $\Phi_k^+$, where

1) $\Pi_k^+ = \{\pi_a^{[1:n]} \mid \pi_a^{[1:n]} = C^{(1)} \dots C^{(n)} \subset \pi_a \in \Pi\}$ is a set of new paths reaching $C_k$ from the roots; $\Pi$ is the set of *full* paths (that comprise all SC sequences from each root to terminal SCs that do not have any next reachable SC) based on SCG; $\pi_a$ is the $a$th path in $\Pi$; $C^{(1)}$ is the root on $\pi_a$; and $C^{(n)} (= C_k)$ is the $n$th nodes.
2) $\Phi_k^+ = \{I(\pi_a^{[1:n]}) \mid \pi_a^{[1:n]} \in \Pi_k^+\}$ is a set of the series of time constraints of $C_k$'s previous SCs following each new path; $I(\pi_a^{[1:n]}) = \langle \phi_a^{(1)}, \dots, \phi_a^{(n-1)} \rangle$; $\forall \ell \in \{1, \dots, n-1\}, \phi_a^{(\ell)} = [\mathfrak{l}_a^{(\ell)}, \mathfrak{u}_a^{(\ell)}]$ is the constraint of the time spent of $\ell$th SC $C^{(\ell)}$; $\mathfrak{l}_a^{(\ell)}$ and $\mathfrak{u}_a^{(\ell)}$ are two *constant* bounds of $\phi_a^{(\ell)}$.

Based on $\Phi_k^+$, we define the path-dependent newly observable time range of $C_k$ as follows.

*Definition 3:* Given a path $\pi_a^{[1:n]} = C^{(1)} \dots C_k \in \Phi^+$, $C_k$ is newly observable in a range of $I_{\text{acc}}(\pi_a^{[1:n]})$, where $I_{\text{acc}}(\pi_a^{[1:n]})$ is a function to accumulate the lower and upper *constant* bounds of each time constraint of previous SCs on $\pi_a^{[1:n]}$, i.e.,

$$I_{\text{acc}}(\pi_a^{[1:n]}) = \left[\sum_{i=1}^{n-1} \mathfrak{l}_a^{(\ell)}, \sum_{i=1}^{n-1} \mathfrak{u}_a^{(\ell)}\right]. \tag{2}$$

The range of $C_k$'s newly observable time based on a given path $\pi_a^{[1:n]}$ is the same as the time range of $C_k$'s last transition $t^{(n-1)}$, where $(t^{(n-1)}, \dots) = \mathfrak{L}(e^{(n-1)})$; $e^{(n-1)}$ is the edge between $C^{(n-1)}$ and $C_k$.

To prevent the iterative invocation of deficient SCs, we schedule particular deficient SCs when they are engaged in a new path as a necessary condition. In addition, the proposed evolution method considers: 1) TCPN (in Definition 1); 2) different stopping criteria sufficient to find all candidates of the next observable transitions (related to REs), without reaching all possible states.

### B. Net Partitioning and Event Estimation Problems

First, we introduce why TCPN partitioning is required for fast SCG evolution and the requirements of TCPN partitioning. Then, we present preliminary notations for virtual estimates.

The number of SCs in an SCG increases exponentially with TPN system complexity (related to the net structure and tokens in the initial marking), as in [4]. Let $\mathcal{C}_0$ be original TPN's ($N_0$'s) system complexity, and let $size(\mathcal{C}_0)$ be the expected number of SCs induced by $\mathcal{C}_0$. If the net and marking of $N_0$ can be split into two nets, $N_1$ and $N_2$, with distributed markings as $|\mathcal{C}_0| = |\mathcal{C}_1| + |\mathcal{C}_2|$, then the overall SC nodes are greatly reduced, for example, $size(\mathcal{C}_1) + size(\mathcal{C}_2) \ll size(\mathcal{C}_0)$ based on the exponential node growth by the system complexity. The net splitting for the complexity distribution should support the independent evolutions of multiple subnets' SCGs that lead to the same virtual estimates (which are computed using the orignal net's SCG). The detailed partitioning method that satisfies this requirement will be described in Section VI-A).

After partitioning, there can be multiple subnets $\{N_m\}$, where $N_m = (P_m, T_m, \dots)$. Each subnet $N_m$ has its own SCG $G_m$, which leads its own virtual estimates. Each subnet's virtual estimates consist of a set of observable RE's types $\Lambda_m$ and their observable time intervals $\mathcal{T}_m$. To compute the estimates, each subnet $N_m$ manages its own full paths $\Pi_m$ of SCG $G_m$ (see Definition 2), path-related time constraints $\Phi_m = \{I(\pi_a) \mid \pi_a \in \Pi_m\}$, and last event's observed time $\tau_m^-$. The estimation methods will be described in Section V-B.

### C. SCG Update Problem

If a physical event is observed, its dedicated SCG handles the physical event. The proposed net-splitting method enables subnets to share observable transitions whose backward and forward places are located in different subnets. If the physical event's transition is shared, the number of related SCGs can be two. After checking DT's consistency using the event (if the event is an RE), a related SCG $G_m$ is updated by three processes:

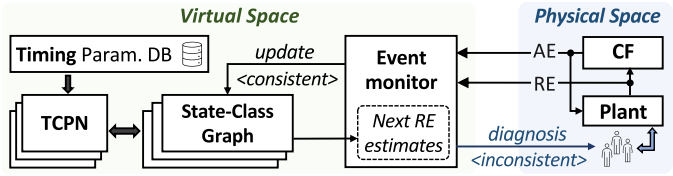



Fig. 5. Abstracted overview of the framework.

1) finding the consistent SCs; 2) revising the SCs to meet a root's property (defined below); 3) starting the SCG evolution based on the roots. In [4] and [5], an initial marking (as a root) is given. In our approach, an initial marking should be given only if a previous SCG does not exist. However, more than one root is iteratively derived during runtime using the previous SCG and each physical event.

An RE-consistent SC has a backward edge whose transition is related to the RE and can occur at the RE-observed time. An AE-consistent SC is an SC that has a marking where an AE-related transition is logically enabled and can stay at the AE-observed time. We will formally define the consistent SCs in Sections VI-B and VI-C. As in (1), each SC's timing constraint $\theta_i$ can depend on previous SCs' time constraints ($\{\Delta^{(-\ell)}\}$, where $\ell > 0$). Thus, we should examine each SC's timing consistency considering all paths $\Pi_m$ using their time-spent information $\Phi_m$.

After finding the consistent SCs, we should revise the SCs so that each SC's constraint $\theta_i$ does not include previous SCs' constraints $\{\Delta^{(-\ell)}\}$ because there are no more previous nodes of the SC. For that, we define roots' property as follows.

*Definition 4:* Given a root $C_k = (M_k, \Theta_k)$, each constraint $\theta_i \in \Theta_k$ is a pair of two constants, i.e., $d_i = 0$.

When the physical event is related to a shared transition $t_i$ between two subnets, a subnet that has $t_i$'s forward places (denoted by $t_i^\bullet$) could need the inflowing token information from the other subnet's SCG to compute $t_i$-induced next marking. To specify the exchanging token information between subnets, we extend the label of SCG's each edge $e$ to contain inflowing tokens $X_j$, such as $\mathfrak{L}(e) = \langle t_i, \ldots, X_j \rangle$.

## V. DT-Based Event Estimation

This section presents the methods for the proposed DT consistency checking. First, we discuss the evolution of SCG using TCPN in Section V-A. Next, the estimation of next REs are discussed in Section V-B.

The overall architecture is illustrated in Fig. 5. Before checking, the modelers should develop the TCPN $N$ based on plant-specific knowledge. Then, they should collect operation timing data related to the transitions manually or automatically to build the TCPN's time function $Q$. Depending on the transition's observability, $N$ can be split into multiple subnets $\{N_m\}$, where $N_m = (P_m, T_m, \ldots)$ is the $m$th subnet. The subnets only share some observable transitions, and we will discuss the detailed methods of TCPN partitioning in Section VI-A. Each subnet represents the production dynamics in a specific location of the plant. The subnet $N_m$ is used to compute its own SCG $G_m$ based on initial consistent SCs to estimate the subnet's next RE. The event monitor (EM) converts the streaming physical events and the external interventions into related REs and AEs, respectively.

When an $\lambda$-type RE is observed at time instant $\tau$, the EM's specific procedures are as follows.

1) The EM selects an SCG $G_m$, whose net $N_m$ includes the $\lambda$-related transition $t_i$, that is $\exists t_i \in T_m$ s.t. $t_i = \mathcal{L}^{-1}(\lambda)$. If $t_i$ is shared by two subnets, then $N_m$ satisfies ${}^\bullet t_i \in P_m$, where ${}^\bullet t_i$ is the set of the backward places of $t_i$.
2) The EM checks the consistency condition using the estimated RE types $\Lambda_m$ and their observable times $\mathcal{T}_m$ based on last event's observed time $\tau_m^-$. The condition is $\lambda \in \Lambda_m \wedge (d\tau = \tau - \tau_m^-) \in \mathcal{T}_m(\lambda)$, where $\mathcal{T}_m(\lambda)$ is the estimated observable time interval of $\lambda$ after $\tau_m^-$.
3) If it is consistent, the EM starts the $G_m$ evolution using identified consistent SCs for further RE estimation. If $t_i$ is shared, then $t_i$'s forward SCG $G_n$, s.t. $t_i^\bullet \in P_n$, is also reupdated based on the candidates of moving tokens from $G_m$ to $G_n$.
4) If it is inconsistent, the EM activates an inconsistency diagnosis.

If an AE arrives, the EM does not check the AE's consistency because users (in the CF) can generate the AE at any time in their purpose. The AE only reupdates its associated SCG based on identified consistent SCs. If the AE's transition is shared, then two SCGs are reupdated considering possible moving tokens.

For DT consistency checking, an inconsistency can stem from a TCPN error or any physical change or error, so cross-validation is required to diagnose the actual reason. We categorize the main reasons as follows:

1) a modeling error in the TCPN structure;
2) a new physical exception;
3) an operation-logic change that requires a TCPN revision;
4) a statistical change of an operation period;
5) a packet loss.

Then, the EM initiates a manual diagnosis by looking into subsystems related to the inconsistent subnet.

### A. SCG Evolution

In this section, we describe the SCG evolution procedures in detail, as depicted in Algorithm 1. A subnet $N_m$'s SCG $G_m$ is computed based on root SCs, which are consistent with the recent physical timed event, meet the root property (see Definition 4), and denoted by $S_{\text{root}}$. Root $C_k$ has one initial path $\pi^{[1:1]} = C^{(1)} (= C_k)$ in its new path set $\Pi_k^+$ and its emptied previous time-spent series, i.e., $I(\pi^{[1:1]}) = \langle\rangle$, in $\Phi_k^+$ (see Definition 2). $C_k$'s newly observable time range is zero, i.e., $I_{\text{acc}}(\pi^{[1:1]}) = [0{:}0]$. SCG $G_m$'s full path set $\Pi_m$ is initialized as $\bigcup_{C_k \in S_{\text{root}}} \Pi_k^+$. $\Pi_m$'s time-spent intervals, $\Phi_m$, consist of emptied time-spent information for each path in $\Pi_m$. $S_{\text{root}}$ is stored in a set $S_{\text{wait}}$ that consists of waiting SCs (to be evolved) in the insertion order. Each SC $C_k = (M_k, \Theta_k)$ in $S_{\text{wait}}$ is evolved, as depicted in Algorithm 1.

First, the EM attempts to identify the set of *imminent* transitions (which can be fired first), denoted by $T_{\text{imm}}$, among candidates $T_k$ as Line 4. $T_k$ is a set of logically enabled transitions, except for $T_{\text{ae}}$ (whose firing is triggered by physical AEs) and the

previously identified imminent transitions (which are denoted by $T^{-}_{\text{imm}}$ and specified on forward edges). Then, time-bound $u_{\min}$, s.t $\min\{u^{\circ}_i \mid \theta_i = (\cdot, u^{\circ}_i) \in \Theta_k\}$, is computed to examine the following necessary proposition.

*Proposition 1:* Transition $t_i \in T_k$ s.t. $l^{\circ}_i > u_{\min}$ cannot be an imminent transition.

*Proof:* Let $t_j$ be a transition s.t. ${}^{\circ}u_j = u_{\min}$. If all transitions in $T_k \setminus \{t_j\}$ are not fired until $u_{\min}$, then $t_j$ is eventually fired first based on the STS assumption. Therefore, a first transition always occurs within $u_{\min}$. ■

As (1) shows, each upper bound $u^{\circ}_i$ can include any previous SCs' time-spent variables $\{\Delta^{(-\ell)}\}$, and the intervals vary depending on the path, which changes $u_{\min}$ and $T_{\text{imm}}$ accordingly. Thus, the necessary condition to be imminent is checked for each transition $t_i \in T_k$ based on Proposition 1, considering all new previous paths in $\Pi^{+}_k$, as Lines 6–7. From (1), $l^{\circ}_i(\pi^{[1:n]}_a)$ and $u_{\min}(\pi^{[1:n]}_a)$ are given by

$$l^{\circ}_i(\pi^{[1:n]}_a) = l_i - \sum_{\ell=1}^{d_i} \mathfrak{u}^{(n-\ell)}_a \tag{3}$$

$$u_{\min}(\pi^{[1:n]}_a) = \min_{t_i \in T_k} \left\{ u_i - \sum_{\ell=1}^{d_i} \mathfrak{l}^{(n-\ell)}_a \right\}. \tag{4}$$

If a previous path, $\pi^{[1:n]}_a \in \Pi^{+}_m$, makes transition $t_i$ imminent as Line 7, $\pi^{[1:n]}_a$ is extended by $t_i$-induced subsequent SC(s) from $C_k$. In addition, SC $C_k$ adds $\pi^{[1:n]}_a$ to set $T_{\text{imm}}$ and stores all previous paths and corresponding interval series (making $t_i$ imminent) in a dataset, $\mathfrak{D}_i = \{\langle \pi^{[1:n]}_a, I(\pi^{[1:n]}_a)\rangle\}$, to save the path candidates for their extension. As mentioned in Section IV-C, the set of full paths, $\Pi_m$, and its set of time-spent intervals, $\Phi_m$, are important to deriving the next RE's virtual candidates. If $\pi^{[1:n]}_a \in \Pi_m$ (i.e., $|\pi_a| = n$), then $\pi_a$ and $I(\pi_a)$ are removed from $\Pi_m$ and $\Phi_m$ as Line 10. After examining all new paths using their own time-spent information, $C_k$ clears $(\Pi^{+}_k, \Phi^{+}_k)$. Let $S_{\text{def}}$ be the set of deficient SCs. $C_k$ updates $S_{\text{def}}$ as Lines 12 and 13.

Using each new imminent transition $t_i$, we derive the candidates of tokens inflowing to $t_i$, $\{X_j\}$. Then, the EM finds the next SC, $C' = (M', \Theta')$. Thus, $M'$ is derived as Line 16, and $\Theta'$ is computed by considering newly disabled transitions (as Line 17), the time spent of $C_k$ (as Line 19), and newly enabled transitions (as Line 21).

If $C'$ is reusable using a function $f_{\text{eq}}$, then we reuse existing $C'$ (in $G_m$) by making an edge $e_i$ from $C_k$ to $C'$. If not, a new node for $C'$ is connected to $C_k$ by edge $e_i$. If $C'$ is reusable, then edge $e_i$'s label is extended by adding $f_{\text{eq}}$. Unlike in the TPN-based SCG evolution, there can be multiple candidates for leaving tokens at the transition $t_i$ firing (i.e., $|Pre(M_k, t_i)| \geq 1$), so an SC can have multiple forward edges for the $t_i$ firing, as in SC $C_3$ in Fig. 6.

Using each new edge (caused by each transition $t_i$) from SC $C_k$ to $C'$, $C_k$ computes the set of new full paths (denoted by $\Pi^{\circ}$) using the saved paths in $\mathfrak{D}_i$. $C'$ is one of a newly added or a reused SC. If $C'$ was newly added, $\Pi^{\circ}$ is set by the following: $\{\pi^{[1:s]} | \pi^{[1:s]} = \pi^{[1:n]}_a \frown (C'), s = n+1, \langle \pi^{[1:n]}_a, \cdot\rangle \in \mathfrak{D}_i\}$, where $\frown$ concatenates the given sequences.

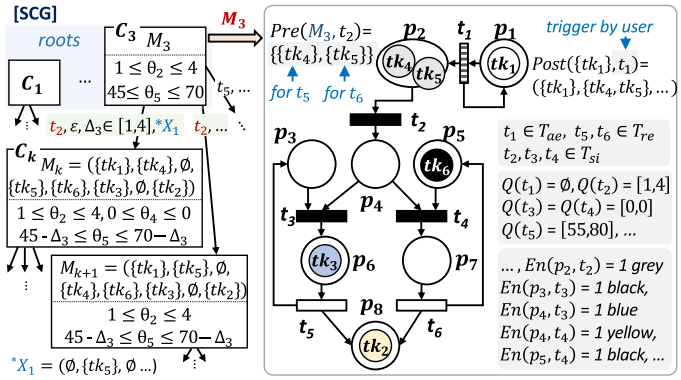



Fig. 6. SCG example based on a TCPN and initial roots $(C_1, \ldots, C_3)$.

If SC $C'$ is reused, $C'$ can already have its subsequent paths. Then, we extend the previous paths saved in $\mathfrak{D}_i$ based on $C'$'s subsequent paths. We denote $C'$'s subsequent paths as $\Pi_{\text{sub}} = \{\pi_{\text{sub}} \mid \pi_{\text{sub}} = C' \ldots C^{(\ell)} \subset \pi_a \in \Pi_m, \ell = |\pi_a|\}$. A full path $\pi_a \in \Pi_m$ can lead to multiple subpaths (reaching $\pi_a$'s terminal) when $C'$ exists multiple times in $\pi_a$. Using $\Pi_{\text{sub}}$, we represent the new path set by $\Pi^{\circ} = \{\pi^{[1:s]} \mid \pi^{[1:s]} = \pi^{[1:n]}_a \frown \pi^{[1:q]}_b, s = n+q, \langle \pi^{[1:n]}_a, \cdot\rangle \in \mathfrak{D}_i, \pi^{[1:q]}_b \subset \pi_b \in \Pi_{\text{sub}}, q = \max_{1 \le q \le |\pi_b|}\{q \mid \forall \ell \in \{1, \ldots, q\} \exists \phi^{(\ell)} (\mathfrak{l}^{(\ell)} \le \mathfrak{u}^{(\ell)})\}\}$. When merging a previous path $\pi^{[1:n]}_a$ and a path $\pi_b \in \Pi_{\text{sub}}$, an SC $C^{(\ell)} \in \pi_b$ might not be accessible from $\pi^{[1:n]}_a$ because of the violation of Proposition 1, which results in $\mathfrak{l}^{(n+\ell)} > \mathfrak{u}^{(n+\ell)}$, where $[\mathfrak{l}^{(n+\ell)}, \mathfrak{u}^{(n+\ell)}] = \phi^{(n+\ell)}$.

We denote $\Pi^{\circ}$'s corresponding time-spent intervals as $\Phi^{\circ} = \{I(\pi_a) \mid \pi_a \in \Pi^{\circ}$. The derivation method of $\Phi^{\circ}$ based on the calculation of $I(\pi_a)$ will be discussed in the following section. Based on $(\Pi^{\circ}, \Phi^{\circ})$, $G_m$'s full path information $(\Pi_m, \Phi_m)$ is updated as Lines 28. In our work, the next SC $C'$ evolves if the fired transition $t_i$ is silent. If $t_i$ is observable, $C'$ evolves after $t_i$ is confirmed as being consistent with the next RE and after being revised to satisfy the property in Definition 4. If $C'$ was not reused and scheduled in $S_{\text{wait}}$ for further evolution, new path information $(\Pi^{\circ}, \Phi^{\circ})$ is delivered to $C'$ as Line 32. If $C'$ was reused, $C_k$ traverses subsequent deficient SCs on new full paths to schedule them as Lines 34–36. This active scheduling of deficient SCs can avoid inefficiency in iterative examinations of all deficient SCs whenever there are no new SCs waiting for further evolution, as in [5]. Finally, a scheduled SC starts a new evolution.

## B. Estimation of Next Reactive Events

The proposed approach utilizes full path and time-spent information of SCG $G_m$, $(\Pi_m, \Phi_m)$, to estimate the next timed RE. Let $e^{(\ell)}_a$ be the $\ell$th edge on path $\pi_a \in \Pi_m$, connecting $C^{(\ell)}$ and $C^{(\ell+1)}$, where $1 \le \ell < |\pi_a|$. Then, the set of the next REs'

**Algorithm 1:** $C_k$.evolve() in SCG $G_m$.

```
// Find the imminent transitions T_imm
let C_k^• be a set of C_k's forward edges.
T_imm^- = {t_i | (t_i ...) = 𝔏(e_i), e_i ∈ C_k^•}; T_imm ← ∅.
T_k ← 𝒜(M_k) \ T_ae \ T_imm^-
u_min ← min{u_i^° | θ_i=(·, u_i^°) ∈ Θ_k}
foreach (π_a^[1:n], t_i) ∈ Π_k^+ × T_k do      // Π_k^+ := new paths
    if max{0, l_i^°(π_a^[1:n])} ≤ u_min(π_a^[1:n]) then
        T_imm ← T_imm ∪ {t_i};
          𝔇_i ← 𝔇_i ∪ {⟨π_a^[1:n], I(π_a^[1:n])⟩}
        if π_a^[1:n] = π_a then
            Π_m ← Π_m \ {π_a}; Φ_m ← Φ_m \ {I(π_a)}
Π_k^+ ← ∅; Φ_k^+ ← ∅
if C_k ∉ S_def ∧ |T_k| > |T_imm| then S_def ← S_def ∪ {C_k}
if C_k ∈ S_def ∧ |T_k| = |T_imm| then S_def ← S_def \ {C_k}
// Find the subsequent SCs
foreach X_j ∈ Pre(M_k, t_i) s.t. t_i ∈ T_imm do
    M' ← M_k ⊖ X_j ⊕ Post(X_j, t_i); Θ' ← Θ_k
    remove {θ_ℓ} from Θ' s.t. M' < En(·, t_ℓ).      // Sec. III
    foreach θ_ℓ = (l_ℓ^°, u_ℓ^°) ∈ Θ' do
        θ_ℓ ← [l_ℓ^° − Δ_i, u_ℓ^° − Δ_i]
    foreach newly enabled transition t_ℓ at M' do
        add θ_ℓ = Q(t_ℓ) to Θ'.
    let C' = (M', Θ') and e be an edge s.t
      𝔏(e) = ⟨t_i, ℒ(t_i), Δ_i ∈ [max{0, l_i^°}, u_min], X_j⟩.
    if C' is reusable using f_eq then add f_eq to 𝔏(e_i),
    else
        add C' to G_m as a new node.
    add edge e between C_k and C'.
    compute new full path info. (Π^°, Φ^°) using 𝔇_i.
    Π_m ← Π_m ∪ Π^°; Φ_m ← Φ_m ∪ Φ^°
    let C' be the qth SC in G_m, i.e., C' = C_q.
    if t_i ∈ T_si then      // C' scheduling condition
        // Schedule a new SC or path-changed
           deficient SCs with changed path info.
        if C' was not reused then
            Π_q^+ ← Π_q^+ ∪ Π^°; Φ_q^+ ← Φ_q^+ ∪ Φ^°; add C' to S_wait.
        else
            foreach π_b^[1:s] = ... C_k C' ... C^(s) ⊂ π_b ∈ Π^°
              s.t. C^(s) ∈ S_def do let C^(s) = C_r.
            Π_r^+ ← Π_r^+ ∪ {π_b^[1:s]}; Φ_r^+ ← Φ_r^+ ∪ {I(π_b^[1:s])}
            add C_r to S_wait.
if S_wait ≠ ∅ then C_k ← popfront(S_wait); C_k.evolve()
```

types from SCG $G_m$ is given by

$$\Lambda_m = \{\mathcal{L}(t) \mid \forall \pi_a \in \Pi_m, \langle t, \ldots\rangle = \mathfrak{L}(e_a^{(|\pi_a|-1)}) \text{ s.t } t \in T_{\text{re}}\}. \tag{5}$$

Let $\Pi_m(\lambda)$ be the set of paths in SCG $G_m$, leading to specific RE type $\lambda$. Based on subnet $N_m$'s recent event-arrival time, $\tau_m^-$, the expected observable time range of the $\lambda$-type RE, $\mathcal{T}_m(\lambda)$, is given by

$$\mathcal{T}_m(\lambda) = \bigcup\nolimits_{\pi_a \in \Pi_m(\lambda)} I_{\text{acc}}(\pi_a). \tag{6}$$

Thus, the expected observable time of any RE related to $N_m$ (from time $\tau_m^-$) is $\cup_{\lambda \in \Lambda_m} \mathcal{T}_m(\lambda)$. To calculate $\mathcal{T}_m(\lambda)$, we need to compute each time-spent interval $\phi_a^{(\ell)} = [\mathfrak{l}_a^{(\ell)}, \mathfrak{u}_a^{(\ell)}] \in I(\pi)$. For

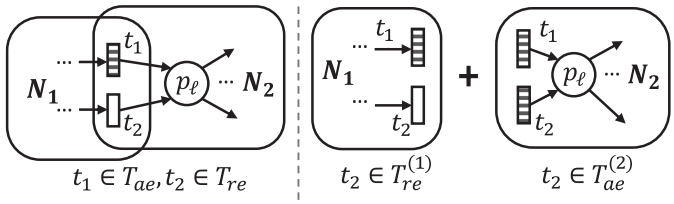



Fig. 7. Example of TCPN partitioning. (a) Before partitioning. (b) After partitioning.

this, we reformulate the lower and upper bounds as follows:

$$\mathfrak{l}_a^{(\ell)} = \max\{0, l_i^\circ\} = \max\left\{0, l_i - \sum_{j=1}^{d_i} \max(\Delta^{(\ell-j)})\right\}$$
$$= \max\left\{0, l_i - \sum_{j=1}^{d} \mathfrak{u}_a^{(\ell-j)}\right\}, \tag{7}$$

$$\mathfrak{u}_a^{(\ell)} = \min_{k\in\mathcal{K}}\left\{{}^\circ u_k\right\} = \min_{k\in\mathcal{K}}\{u_k - \sum_{j=1}^{d_k} \min(\Delta^{(\ell-j)})\right\}$$
$$= \min_{k\in\mathcal{K}}\left\{u_k - \sum_{j=1}^{d_k} \mathfrak{l}_a^{(\ell-j)}\right\} \tag{8}$$

where $i$ is the transition index in $e_a^{(\ell)}$, i.e., $\langle t_i, \ldots\rangle = \mathfrak{L}(e_a^{(\ell)})$; and $\mathcal{K}$ is the set of indices of fireable transitions, i.e., $\mathcal{K} = \{i \mid \theta_i \in \Theta^{(\ell)}, (\cdot, \Theta^{(\ell)}) = C^{(\ell)}\}$. As such, $\mathfrak{l}^{(\ell)}$ and $\mathfrak{u}^{(\ell)}$ can be derived using the previous values of $\{\mathfrak{l}^{(\ell-j)}\}$ and $\{\mathfrak{u}^{(\ell-j)}\}$ in a recursive manner. Based on the property in Definition 4, the first transition $t_i$ (i.e., $\langle t_i, \ldots\rangle = \mathfrak{L}(e_a^{(1)})$) has time constraint $\theta_i = [l_i^\circ, u_i^\circ]$, which is an interval having two constraint bounds and leading to $\phi_a^{(1)} = \theta_i$. The derivation of first time-spent interval, $\phi_a^{(1)}$ on $\Pi_a$, enables subsequent time-spent intervals ($\phi_a^{(2)}, \phi_a^{(3)}$, and so forth) to be recursively calculated as constant bounds.

## VI. Revision of TCPN and SCG

### A. Partitioning of TCPN

As discussed in Section IV-B, the sizes of TPN-based SCGs in [4] and [5] increase exponentially with TPN system complexity. Compared to the TPN-based evolution, the size of the proposed TCPN-based SCG grows more rapidly than that of a TPN-based SCG because each imminent transition $t_i$ of each TCPN-based SC $C_k$ can lead to multiple subsequent SCs—as many as $|Pre(M_k, t_i)|$ (see SC $C_3$ in Fig. 6). Thus, the speed up evolution of TCPN-based SCGs is essential for the runtime analysis. Hence, we propose partitioning TCPN to speed up the SCG evolution, described as follows.

We denote the set of the backward and forward transitions of place $p$ as ${}^\bullet p$ and $p^\bullet$, respectively. Let us look at the example in Fig. 7(a), where ${}^\bullet p_\ell \subset T_{\text{ob}} = T_{\text{re}} \cup T_{\text{ae}}$. Suppose that two subnets of $N_1 = (P_1, T_1, \ldots)$ and $N_2 = (P_2, T_2, \ldots)$ have the

following properties: $P_1 \cap P_2 = \emptyset$ and $T_1 \cap T_2 = {}^\bullet p_\ell$. Based on the SC scheduling condition of Algorithm 1, any path from a root in an SCG is terminated when an observable transition is encountered. Thus, $N_1$'s SCG $G_1$ cannot affect the evolution of $N_2$'s SCG $G_2$ because any path reaching ${}^\bullet p_\ell$ is stopped. Let $T_{\text{ae}}^{(i)}$ and $T_{\text{re}}^{(i)}$ be the sets of subnet $N_i$'s AE- and RE-related transitions, respectively. Then, $G_2$ can also evolve independently to estimate an RE related to $(T_2 \cap T_{\text{re}}) \setminus {}^\bullet p_\ell$ after assuming ${}^\bullet p_\ell$ to be an unpredictable transition in $T_{\text{ae}}$ (i.e., ${}^\bullet p_\ell \in T_{\text{ae}}^{(2)}$), as Fig. 7(b) shows. $G_2$ is influenced by $G_1$ only when ${}^\bullet p_\ell$-related physical events are observed, which triggers the $G_2$ reupdate. Generally, the set of subnets $\mathcal{N}$ from an original TCPN $N_0 = (P_0, T_0, \cdots)$ must meet the following definition:

*Definition 5:* The set of subnet $\mathcal{N} = \{N_\ell\}$, where $N_\ell = (P_\ell, T_\ell, \cdots)$ is the $\ell$th subnet, $1 \le \ell \le n$, and $n = |\mathcal{N}|$, should meet the following properties: $\bigcup_{\ell=1}^{n} P_\ell = P_0$; $\bigcup_{\ell=1}^{n} T_\ell = T_0$; and $\forall i, j \in \{1, \ldots, n\}(i \neq j) \rightarrow (P_i \cap P_j = \emptyset) \wedge (T_i \cap T_j \in T_{\text{ob}})$.

To compute the subnets, we initially make temporary and minimum-sized subnets using each place in $P_0$ of the original net $N_0$: e.g., $\mathcal{N} = \{N_\ell\}$ s.t. $|\mathcal{N}| = |P_0|$; $|P_\ell| = 1$, and $T_\ell = {}^\bullet p_\ell \cup p_\ell^\bullet$, where $p_\ell$ is the $\ell$th place in $P_0$. Each subnet iteratively merges other subnets in $\mathcal{N}$ when two paired subnets share any silent transition. If there are no longer any subnets to merge, then the remaining subnets are the final subnets.

### B. SCG Update Based on Reactive Events

When detecting a new $\lambda$-type event at time $\tau^{(0)}$, the EM selects the SCG $G_m$ whose subnet $N_m$ contains transition $t_i$ s.t. $t_i = \mathcal{L}^{-1} \in T_{\text{re}}^{(m)}$. Then, the EM decides the consistency by checking $(\lambda \in \Lambda_m) \wedge (d\tau \in \mathcal{T}_m(\lambda))$ from (5) and (6), where $d\tau = \tau^{(0)} - \tau_m^-$, not one subsequent SC in the TPN-based SCG. If it is inconsistent, the EM triggers the inconsistency diagnosis for the subnet. If it is consistent, the EM finds root SCs for the further SCG evolution by following two main steps.

1) Select the paths whose terminals are consistent SCs $\{C_k = (M_k, \Theta_k)\}$ with the timed RE.
2) Revise each time constraint $\theta_i \in \Theta_k$ of each consistent SC $C_k$ by resolving previous SCs' time-spent variables $\{\Delta^{(\ell)}\}$ to satisfy the property of Definition 4.

Let $\Omega$ be a set of the pairs of consistent SC and its path. Then, $\Omega$ is represented by

$$\Omega = \{(C^{(n)}, \pi_a) | \pi_a = C^{(1)} \ldots C^{(n)} \in \Pi_m, e_a^{(n-1)} = \lambda, d\tau \in I_{\text{acc}}(\pi_a)\}.$$

From a physical event, if we can get more information than type $\lambda$, which helps distinguish the processed tokens (annotated in the last edge $e^{(n-1)}$) based on their data, we can reduce the number of consistent SCs additionally, which lessens the evolution overhead.

For each $(C^{(n)}, \pi_a) \in \Omega$, where $n = |\pi_a|$, the previous time-spent terms in each time constraint $\theta_i$ can be simply changed into a constant interval from $\sum_{\ell=1}^{d_i} \Delta^{(n-\ell)}$ to $\sum_{\ell=1}^{d_i} \phi_a^{(n-\ell)}$ (which is $I_{\text{acc}}(\pi_a^{[n-d_i:n]})$. However, we should consider one additional constraint, $d\tau$.

Suppose that consistent path $\pi_a$, s.t $|\pi_a| = 5$ and $(\cdot, \pi_a) \in \Omega$, meets the following properties: $I_{\text{acc}}(\pi_a^{[1:4]}) = [20 : 24]$, $I_{\text{acc}}(\pi_a^{[4:5]}) = \psi_a^{(4)} = [2 : 4]$, and $I_{\text{acc}}(\pi_a) = [22 : 28]$. For constraint $\theta_i$, if $d_i = 1$ and $d\tau = 26$, then $\sum_{\ell=1}^{d_i} \Delta^{(n-\ell)}$ (which is $\Delta^{(4)}$) is the same as $I_{\text{acc}}(\pi^{[4:5]}) = [2 : 4]$ because $\Delta^{(4)}$ can be 2 if $\sum_{j=1}^{3} \Delta^{(j)} = 24 \in I_{\text{acc}}(\pi_a^{[1:4]})$ and can be 4 if $\sum_{j=1}^{3} \Delta^{(j)} = 22$. However, if $d\tau = 22$, $\sum_{j=1}^{3} \Delta^{(j)}$ and $\Delta^{(4)}$ should be $[20 : 20]$ and $[2 : 2]$, respectively, which makes the interval of $\Delta^{(4)}$ smaller than $I_{\text{acc}}(\pi^{[4:5]})$. In a similar manner, $d\tau = 28$ reduces the interval of $\Delta^{(4)}$ to $[4 : 4]$. Considering the $d\tau$ constraint, we can generally formulate the interval of $\sum_{\ell=1}^{d_i} \Delta^{(n-\ell)}$ based on $d\tau$ and $\pi_a$, as follows:

$$\min \sum_{\ell=1}^{d_i} \Delta^{(n-\ell)} = \max \left\{ \sum_{\ell=1}^{d_i} \mathfrak{l}_a^{(n-\ell)}, d\tau - \sum_{j=1}^{n-d_i-1} \mathfrak{u}_a^{(j)} \right\} \tag{9}$$

$$\max \sum_{\ell=1}^{d_i} \Delta^{(n-\ell)} = \min \left\{ \sum_{\ell=1}^{d_i} \mathfrak{u}_a^{(n-\ell)}, d\tau - \sum_{j=1}^{n-d_i-1} \mathfrak{l}_a^{(j)} \right\}. \tag{10}$$

From (9) and (10), each $\Delta$-included constraint $\theta_i \in \Theta_k^{(j)}$ in each $C^{(n)}$ is renewed by

$$\theta_i = \left[ \max \left\{ 0, l_i - \max \sum_{\ell=1}^{d_i} \Delta^{(n-\ell)} \right\}, u_i - \min \sum_{\ell=1}^{d_i} \Delta^{(n-\ell)} \right]. \tag{11}$$

Then, renewed consistent SCs are saved to the root set, $S_{\text{root}}$, as initial SCs for the next $G_m$.

### C. SCG Update Based on Actuation Events

When observing a $\lambda$-type event at time $\tau^{(0)}$, the EM selects one or two SCG $\{G_m\}$ s.t. $t_i = \mathcal{L}^{-1}(\lambda) \in T_{\text{ae}}^{(m)}$; two SCGs can be found when $t_i$ is shared by two subnets. Similarly, in Section VI-B, the EM aims to find the consistent SCs and their paths $\Omega$ based on $d\tau (= \tau^{(0)} - \tau_m^-)$ to derive the roots $S_{\text{root}}$ for the next SCG. Compared to the RE-consistent SCs (which are the terminals of specific paths in $\Pi_m$), any SCs $\{C^{(j)} = (M^{(j)}, \Theta^{(j)})\}$ on paths s.t. $C^{(j)}$ is observable at $d\tau$ and $t_i$ is logically enabled at $M^{(j)}$ are consistent SCs. Thus, we represent the pairs of consistent SCs and their belonging paths, $\Omega$, by $\Omega = \{(C^{(j)}, \pi_a^{[1:j]}) \mid \pi_a^{[1:j]} = C^{(1)} \ldots C^{(j)} \subset \pi_a \in \Pi_m, \sum_{\ell=1}^{j-1} \mathfrak{l}_a^{(\ell)} \le d\tau \le \sum_{\ell=1}^{j} \mathfrak{u}_a^{(\ell)}, \forall \ell \in \{1, \ldots, j-1\} (t_i^{(\ell)} \in T_{\text{si}}), t_i \in \mathcal{A}(M^{(j)})\}$, where $\langle t^{(\ell)}, \cdots \rangle = \mathfrak{L}(e^{(\ell)})$. If an SC $C^{(j)}$ is a path $\pi_a$'s terminal and its last transition is silent (i.e., $t^{(j-1)} \in T_{\text{si}}$), then $\mathfrak{u}_a^{(j)}$ is $\infty$. If $t_i$ is shared and a subnet $G_m = (P_m, \ldots)$ has $t_i$'s forward places (i.e., $t_i^\bullet \in P_m$), then $t_i$'s logical enabling is not checked. After deriving $\Omega$, the EM performs the following two main steps to obtain the roots.

1) Renew $\Delta$-dependent timing constraints of consistent SCs in $\Omega$ for Definition 4.
2) Update the markings and add new timing constraints of renewed SCs based on the $t_i$ firing.

Compared to the renewals of RE-driven timing constraints, we should consider the elapsed time of each SC from previous transition $t^{(j-1)}$ up to $\tau^{(0)}$. For each $(C^{(j)}, \pi_a^{[1:j]}) \in \Omega$, we add new

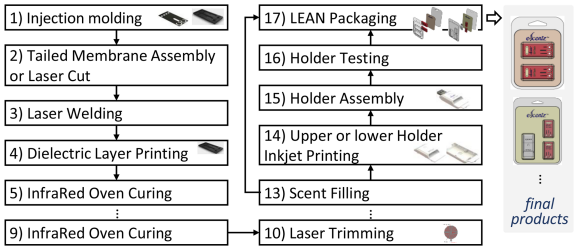



Fig. 8. Overview of the plant's manufacturing processes.

$j$th time-spent variable $\Delta^{(j)}$ to each constraint $\theta_i \in \Theta^{(j)}$, which symbolizes the interval of $C^{(j)}$ elapsed time. This leads to $\phi_a^{(j)} = [\max\{0, d\tau - \sum_{\ell=1}^{j-1} \mathfrak{u}^{(\ell)}\}, d\tau - \sum_{\ell=1}^{j-1} \mathfrak{l}^{(\ell)}]$. The $\Delta^{(j)}$ insertion resets the relative time-spent term in $\theta_i$ to $\sum_{\ell=0}^{d_i} \Delta^{(j-\ell)}$ from $\sum_{\ell=1}^{d_i} \Delta^{(j-\ell)}$. Then, constraint $\theta_i$ becomes $\Delta$ free using (9) and (10).

After the constraint renewal, the EM begins the marking update. If $t_i$ is not a shared transition, the marking of each consistent SC $C_k = (M_k, \Theta_k)$ s.t. $(C_k, \cdot) \in L$ is revised using the next markings $\{M' \mid X_j \in Pre(M_k, t_i), M' = M_k \ominus X_j \oplus Post(X_j, t_i)\}$. If the size of $\{M'\}$ is larger than 1, multiple SCs are generated from $C_k$ and their constraints are also updated based on their markings.

If $t_i$ is shared by two subnets $G_m$ and $G_n$, s.t. ${}^\bullet t_i \in P_m$ and $t_i^\bullet \in P_n$, two subnets exchange token candidates inflowing to $t_i$ from $G_m$. Let $S_{\text{con}}^{(m)}$ and $S_{\text{con}}^{(n)}$ be the constraint-renewed consistent SCs from $G_m$ and $G_n$, respectively. If $t_i \in T_{\text{ae}}$, then each $C_k \in S_{\text{con}}^{(m)}$ is revised for the next markings $\{M' \mid X_j \in Pre(M_k, t_i), M' = M_k \ominus X_j\}$. Then, each SC $C_r \in S_{\text{con}}^{(n)}$ is revised or duplicated based on the next markings $\{M' \mid M' = M_r \oplus Post(X_j, t_i), X_j \in \mathcal{X}_i\}$, where $\mathcal{X}_i = \{X_j \mid X_j \in Pre(M_k, t_i), C_k \in S_{\text{con}}^{(m)}\}$. If $t_i \in T_{\text{re}} \wedge t_i \in T_{\text{ae}}^{(n)}$ and $\mathcal{L}(t_i)$-type RE arrives, each SC $C_r \in S_{\text{con}}^{(n)}$ is revised in the same manner in which the token candidates between the $t_i$-shared subnets are exchanged.

## VII. Case Study

We applied the proposed approach to maintaining a DT of a factory located on the Singapore Institute of Manufacturing Technology (SIMTech) that has produced various research prototypes, such as customized USB devices [23]. The USB device requires a customer-selected logo and emits a scent, such as lavender, from a scent-filled cartridge when the device is active. Overall, the manufacturing processes consist of 17 steps: Steps 1 to 13, which are common to all products, and the following optional steps (including Step 14 for customer-specific sentences), as Fig. 8 shows. When a customer's order arrives, the CF generates an AE for specific parts of trays to be processed by custom steps. An operator in the CF occasionally generates an AE for some parts of a tray to complete specific steps (e.g., Steps 3–6) to reduce the makespan of the final product. If a tray does not undergo a process, it stays in a work-in-progress (WIP) conveyor or a machine's output dock(s).

Twelve machines are used for the production, and some machines, such as an infrared oven and a screen printer, are involved

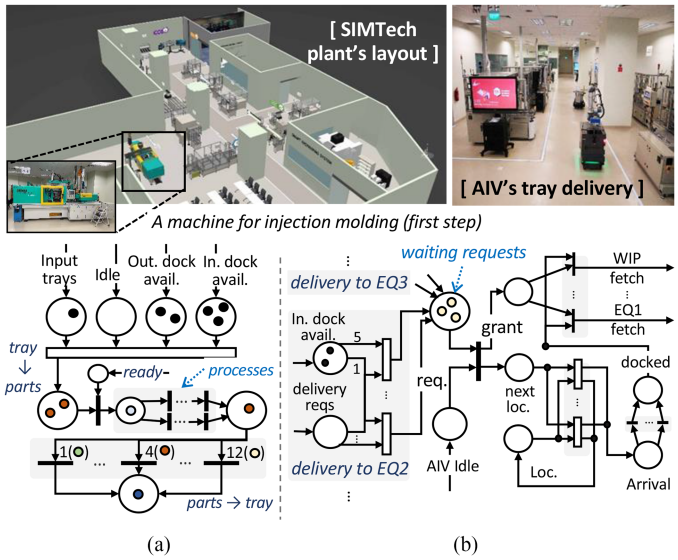



Fig. 9. Examples of token-color conversions in the plant's TCPN. (a) Subnet for a tool's part-specific molding process. (b) Common subnets for tray-delivery requests, request grant, and AIV driving.

in more than one step. The processed trays in a machine's output dock are transferred by an automatic intelligence vehicle (AIV) to a destination's input dock or WIP. The TCPN for the factory consists of 220 places and 381 transitions, representing the tray unloading/loading by AIV, single- or multiple-part-specific processes of each machine, the delivery request and its grant, AIV movements along paths, and so on. The tokens represent the actual trays (for the part exchanges between machines), parts (for machines' processes), and synthetic control units (for the delivery requests, machines' availability, etc.). Each tray can carry up to 12 different types of parts. The machines, excluding the oven, process the input tray's specific parts based on the tray token's data. For example, if four parts of a tray token are required to be processed based on the tray token's part-related data, a $Post$ function changes the tray token to four part tokens, as Fig. 9(a) shows. In the case of an injection-molding machine, depending on the part type (whether it is a reservoir or a bottom cartridge), its processing time varies; thus, part tokens' colors change for their subsequent processes. The part tokens would be merged into a tray token to be moved. As illustrated in Fig. 9(b), the target TCPN has common subnets for a tray-delivery request and its grant using an AIV. In the net, feasible requests are identified considering the availabilities of destinations' input docks. Among the tokens symbolizing feasible requests, a top-priority token is selected based on the tokens' data.

Using a daily RE/AE-observation history, we evaluated the DT-checking performance by varying the degree of the RE observation (OB). The degrees are categorized as follows.

1) OB1: All REs for all machines' tray process starts and trays' arrivals at the input and output docks.
2) OB2: All REs related to OB1, process endings/failures, and delivery requests.
3) OB3: All REs related to OB2 and AIV's idles/failures.

OB3 represents all possible REs from the factory. Examples of silent transitions include order-related setups and

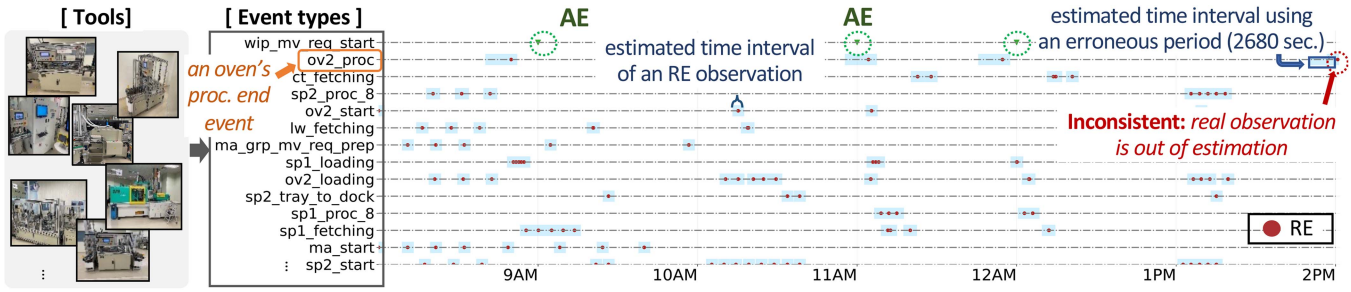
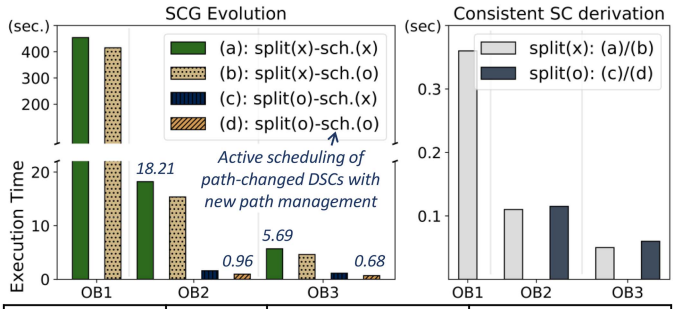



Fig. 10. Example of an inconsistency detection when the average of the operation,ov2_proc, is significantly different from its real measurement.




| | OB1 | | OB2 | | | | OB3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | (a) | (b) | (a) | (b) | (c) | (d) | (a) | (b) | (c) | (d) |
| # of physical events | 405 | | 555 | | | | 713 | | | |
| $\lvert T_{ob}\rvert/\lvert T\rvert$ | 0.23 | | 0.54 | | | | 0.58 | | | |
| # of SCG evolution | 405 | | 555 | | 737 | | 713 | | 895 | |
| Avg. of SCG size | 254.5 | | 38.6 | | 7.74 | | 17.4 | | 4.8 | |
| Avg. of path size (# of RE estimates) | 393.3 | | 36.9 | | 3.8 | | 18.4 | | 2.0 | |
| Avg. # of SC iteration | 171 | 96.8 | 35.7 | 19.9 | 5.2 | 3.8 | 18.4 | 8.3 | 5.1 | 2.9 |

Fig. 11. Performance comparisons varying the evolution methods (between the previous state-of-the-art and the proposed approaches) and TCPN splitting.

subprocesses of machines and AIV's multiple attempts (with extant
maximums) to arrive at specific positions. The transition durations are measured manually or can be calculated based on observed events. Due to the stochastic property of the transition duration, there can be more than one next RE candidate with different time occurrence intervals.

We implemented the environment for the proposed TCPN-based modeling and SCG-based runtime checking using C++. If we intentionally create a DT inconsistency by considerably lowering the average of a specific duration, such as changing an infrared-oven operation from 2987 to 2680 s, an inconsistency can be detected, as Fig. 10 shows.

Aside from OB1, OB2, and OB3 enabled us to split the TCPN into 16 subnets for each of the machines, WIP conveyors, and AIVs, based on the partitioning condition (see Definition 5). Among the subnets, a subnet for AIV and delivery decision, with a place size of 145, is dominant. The overall workload consists of SCG evolution and finding the consistent SCs based on each RE/AE. We measured the performances using a machine with an Intel Xeon E5-1650 3.6 GHz CPU and 32 GB of memory. The results are displayed as Fig. 11 shows.

As the degree of event observability decreases (from OB3 to OB1), the length and number of paths created by firing possible silent transitions subsequently increases, which increases the numbers of the next RE candidates and computational overhead. Compared to the previous SCG evolution method in [5], the proposed SCG evolution method that supports active scheduling (for path-changed deficient SCs) and new path management (which prevents redundant computations in examining deficient transitions), thus decreasing the number of iterations.

TCPN partitioning requires additional consistent SC derivation and SCG evolutions whenever an RE related to a shared transition occurs, which consequently increases the overhead of consistent SC derivation. However, the average SCG size becomes significantly smaller, which drastically reduces the overall computation times. In our OB1 experimentation, the new evolution method reduced the overhead by about 8%. In the OB2 and OB3 cases, the final speedups (which are the ratios of the conventional evolution times to the active-scheduling-based evolution times after partitioning) were $18.97(= 18.21/0.96)$ and $8.28(= 5.69/0.68)$ times, respectively.

Future work will discuss the proposed work's scalability by measuring the SCG evolution times and their speedups caused by TCPN partitioning, varying the plants' scales.

## VIII. Conclusion

This work presents a novel approach for checking DT's consistency (for its high fidelity) by comparing streaming physical events with corresponding DT-based virtual estimates during runtime. For that, we aimed to build the virtual estimates of the plant's each physical event based on reachable observable transitions (that can produce physical events), considering all possible sequences of intermediate unobservable transitions, stochastic state transitions, stochastic operation periods, and external interventions. The previous TPN-based reachability-analysis methods facilitate deriving the reachable transitions and states by identifying all possible sequences of transitions based on their time ranges. To model the plant's high-mix manufacturing using CPN's colors and allow unpredictable users' interventions (AEs), we first designed the TCPN formalism. Then, we extended the previous SCG evolution method to 1) resolve some computational inefficiencies; 2) consider the TCPN formalism (from TPN); 3) alter the evolution stopping criteria sufficient to build the next event's virtual estimates. For better SCG evolution speedup and problematic-subnet isolation, we proposed a TCPN-partitioning method. We also proposed

an iterative SCG update based on the previous SCG and each observed physical event for the next-RE estimation. We applied the approach to a USB device factory. Under a given experimentation set, the TCPN partitioning and revised SCG evolution algorithm accelerated the performance of the state-of-the-art evolution up to 18.97 times.

## REFERENCES


[1] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Trans. Softw. Eng.*, vol. 17, no. 3, pp. 259–273, Mar. 1991.

[2] X. Wang, C. Mahulea, and M. Silva, "Diagnosis of time Petri nets using fault diagnosis graph," *IEEE Trans. Autom. Control*, vol. 60, no. 9, pp. 2321–2335, Sep. 2015.

[3] R. Hadjidj and H. Boucheneb, "Efficient reachability analysis for time Petri nets," *IEEE Trans. Comput.*, vol. 60, no. 8, pp. 1085–1099, Aug. 2011.

[4] F. Basile, M. P. Cabasino, and C. Seatzu, "State estimation and fault diagnosis of labeled time Petri net systems with unobservable transitions," *IEEE Trans. Autom. Control*, vol. 60, no. 4, pp. 997–1009, Apr. 2015.

[5] Z. He, Z. Li, A. Giua, F. Basile, and C. Seatzu, "Some remarks on "state estimation and fault diagnosis of labeled time Petri net systems with unobservable transitions," *IEEE Trans. Autom. Control*, vol. 64, no. 12, pp. 5253–5259, Dec. 2019.

[6] M. Younus, C. Peiyong, L. Hu, and F. Yuqing, "MES development and significant applications in manufacturing-A review," in *Proc. 2nd Int. Conf. Educ. Technol. Comput.*, 2010, vol. 5, pp. V5-97–V5-101.

[7] X. Chen and T. Voigt, "Implementation of the manufacturing execution system in the food and beverage industry," *J. Food Eng.*, vol. 278, 2020, Art. no. 109932.

[8] C. Gehrmann and M. Gunnarsson, "A digital twin based industrial automation and control system security architecture," *IEEE Trans. Ind. Inform.*, vol. 16, no. 1, pp. 669–680, Jan. 2020.

[9] M. Eckhart and A. Ekelhart, "A specification-based state replication approach for digital twins," in *Proc. Workshop Cyber- Phys. Syst. Secur. PrivaCy*, New York, NY, USA, 2018, pp. 36–47.

[10] A. Erba and N. O. Tippenhauer, "No need to know physics: Resilience of process-based model-free anomaly detection for industrial control systems," Dec. 2020, *arXiv: 2012.03586 [cs]*.

[11] I. Hatono, K. Yamagata, and H. Tamura, "Modeling and online scheduling of flexible manufacturing systems using stochastic Petri nets," *IEEE Trans. Softw. Eng.*, vol. 17, no. 2, pp. 126–132, Feb. 1991.

[12] M. Jeng, X. Xie, and M. Peng, "Process nets with resources for manufacturing modeling and their analysis," *IEEE Trans. Robot. Automat.*, vol. 18, no. 6, pp. 875–889, Dec. 2002.

[13] Y. Chen, Z. Li, K. Barkaoui, and M. Uzam, "New Petri net structure and its application to optimal supervisory control: Interval inhibitor arcs," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 44, no. 10, pp. 1384–1400, Oct. 2014.

[14] Y. Chen, Z. Li, and M. Zhou, "Optimal supervisory control of flexible manufacturing systems by petri nets: A set classification approach," *IEEE Trans. Automat. Sci. Eng.*, vol. 11, no. 2, pp. 549–563, Apr. 2014.

[15] J. Luo, K. Xing, M. Zhou, X. Li, and X. Wang, "Deadlock-free scheduling of automated manufacturing systems using Petri nets and hybrid heuristic search," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 45, no. 3, pp. 530–541, Mar. 2015.

[16] H. Liu, K. Xing, W. Wu, M. Zhou, and H. Zou, "Deadlock prevention for flexible manufacturing systems via controllable siphon basis of Petri nets," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 45, no. 3, pp. 519–529, Mar. 2015.

[17] K. Jensen, "Coloured Petri nets," in *Petri Nets: Central Models and Their Properties*. Springer, 1987, pp. 248–299.

[18] J. Ezpeleta and J. Colom, "Automatic synthesis of colored Petri nets for the control of FMS," *IEEE Trans. Robot. Automat.*, vol. 13, no. 3, pp. 327–337, Jun. 1997.

[19] F. Bause, "Queueing Petri Nets-A formalism for the combined qualitative and quantitative analysis of systems," in *Proc. 5th Int. Workshop Petri Nets Perform. Models*, 1993, pp. 14–23.

[20] Y. Q. Lv, C. K. M. Lee, Z. Wu, H. K. Chan, and W. H. Ip, "Priority-based distributed manufacturing process modeling via hierarchical timed color Petri net," *IEEE Trans. Ind. Inform.*, vol. 9, no. 4, pp. 1836–1846, Nov. 2013.

[21] J. Wang, Y. Deng, and G. Xu, "Reachability analysis of real-time systems using time Petri nets," *IEEE Trans. Syst., Man, Cybern., Part B*, vol. 30, no. 5, pp. 725–736, Oct. 2000.

[22] S. Gaubert and J. Mairesse, "Modeling and analysis of timed Petri nets using heaps of pieces," *IEEE Trans. Autom. Control*, vol. 44, no. 4, pp. 683–697, Apr. 1999.

[23] SIMTech, "Model factory simtech," 2022. Accessed: Mar. 22, 2022. [Online]. Available: https://www.a-star.edu.sg/simtech/model-factory@simtech/overview


**Moon Gi Seok** (Member, IEEE) received the B.S. degree in electronics engineering from Korea University, Seoul, Korea, in 2009, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2011 and 2017, respectively.

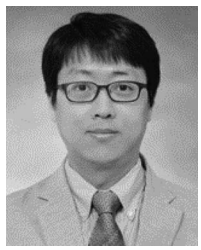
From 2017 to 2019, he was a Postdoctoral Researcher with both KAIST and Arizona State University (ASU), Tempe, AZ, USA. From 2019 to 2023, he was a Research Fellow and Senior Research Fellow at Nanyang Technological University (NTU), Singapore. Since 2023, he has been an Assistant Professor with the Department of Artificial Intelligence at Dongguk University, Seoul. His research interests include digital twinning for manufacturing systems, parallel and distributed simulations, model verification, and hardware/software codesign.

**Wen Jun Tan** received the Ph.D. degree in computer science from Nanyang Technological University (NTU), Singapore, in 2020.

He is a Research Fellow with the School of Compute Science and Engineering (SCSE), NTU, Singapore. His research interests include cyber-physical systems (including virtual model design and data assimilation), modeling and simulation of large-scale complex systems (traffic, supply chain, manufacturing), and application and system support for parallel and distributed simulation on heterogeneous and high-performance computing platforms.

**Wentong Cai** (Member, IEEE) is a Professor with the School of Compute Science and Engineering (SCSE), Nanyang Technological University (NTU), Singapore. His research interests include modeling and simulation, and parallel and distributed computing.

Prof. Cai is a Member of the ACM. He is an Associate Editor for *ACM Transactions on Modeling and Computer Simulation* (TOMACS), an Editor of the *Future Generation Computer Systems* (FGCS), and an Editorial Board Member of the *Journal of Simulation* (JOS).



**Daejin Park** (Member, IEEE) received the B.S. degree in electronics engineering from Kyungpook National University, Daegu, Korea, in 2001, the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2003 and 2014, respectively.

He was a Research Engineer with SK Hynix Semiconductor, Samsung Electronics over 12 years from 2003 to 2014, and has worked on designing low-power embedded processors architecture and implementing fully AI-integrated system-on-chip with intelligent embedded software on the custom-designed hardware accelerator, especially for hardware/software tightly coupled applications, such as smart mobile devices and industrial electronics. Since, 2014, he has been a Full-Time Processor with the School of Electronics and Electrical Engineering and School of Electronics Engineering, Kyungpook National University. He has authored or coauthored over 200 technical papers and 40 patents.

Dr. Park was nominated as one of the Presidential Research Fellows 21, the Republic of Korea, in 2014.