Distributing Sequential Control for Manufacturing Automation Systems

Zivana Jakovljevic, *Member, IEEE*, Vuk Lesi, Stefan Mitrovic, and Miroslav Pajic, *Senior Member, IEEE*

***Abstract*— Recent trends in manufacturing require the use of reconfigurable equipment that facilitates rapid and cost-effective change of functionality through modular design, which supports fast integration. Intelligent devices (e.g., sensors, actuators) with integrated computation and communication capabilities enable high-level modularity, not only with the respect to hardware components but also in terms of control functionality; this can be achieved by distributing control to different network-connected devices. Thus, to enable fast and reliable system reconfigurations, in this brief, we introduce a method for distribution of control tasks and generation of control code for the devices in the control network. Our approach is based on the control interpreted Petri nets (CIPNs) formalism. We start from a CIPN capturing the centralized (overall) control system, and the mapping of input and output signals to local controllers (LCs) (i.e., smart devices) that have direct physical access to system sensors and actuators. From these, our method automatically designs distributed control tasks for LCs in the network, as well as generates control code for each LC. The applicability of the proposed method is experimentally verified on two real-world case studies.**

***Index Terms*— Discrete-event systems, distributed control, petri nets, sequential systems.**

I. Introduction

MODERN manufacturing critically depends on fast and economically sustainable release of highly diversified low-cost products [1]. Accordingly, manufacturing systems need to be characterized by extremely high flexibility and adaptability that can be achieved through the development of reconfigurable manufacturing systems (RMS), one of the key Industry 4.0 features [2]. RMS are based on modular equipment that is easily integrated into new configuration and scalable with respect to production capacity [3], [4].

Functional reconfiguration of manufacturing systems can be achieved through flexible scheduling of manufacturing tasks, which has attracted significant research attention [5]. For example, Wan *et al.* [6] proposes a three-layer architecture based on ontology and IEC 61499 for pharmaceutical manufacturing flexible scheduling. A Petri net-based method to design supervisors that ensure deadlock-free distributed control of parallel processes with mutual resources in assembly systems is presented in [7] and [8]. Also, software component reusability during functional system reconfigurations may be facilitated by optimization of control software parameters as in [9].

Reconfiguration of manufacturing systems requires not only functional but also physical reconfiguration of manufacturing equipment, which is enabled by its modular design. Although the mechanical elements are inherently modular and promote reconfigurability, the state of the art controllers are centralized, which requires significant effort during equipment reconfiguration [10]. For example, pneumatic cylinders support easy reconfiguration of mechanical subsystems in a lego-like manner. Still, their integration into a centralized control system requires considerable effort in terms of control valves and limit sensors wiring to the central controller. On the other hand, smart devices (e.g., sensors and actuators) with integrated computational and communication capabilities, effectively enable control system modularity that follows mechanical modules. In the pneumatic cylinder example, mechanical components (cylinder with limit sensors and valves) are easily augmented with a low-level controller that is in charge of local input–output signals acquisition and processing.

In addition to modular equipment, to facilitate the deployment of RMS, it is necessary to develop methods for their fast and easy design, integration, and reprogramming. Unlike centralized control architectures for which there exist systematic approaches to design sequential controllers in manufacturing systems [11]–[14], there is a lack of systematic engineering methods for distributed control systems design. Currently, the design of a distributed control system that will assure desired global behavior requires considerable human intervention and it is mostly the result of trial and error [9], [10]. To address this, standard IEC 61499: Function Blocks [15] was introduced for modeling of distributed control and automation systems through a network of function blocks, each representing functionality of a software or hardware system component [16]. Although the standard provides modeling guidelines, there is no systematic method for the distribution of tasks to the devices.

Accordingly, the problem of control task distribution has attracted significant attention in recent years. For example, [17] proposes a method for manual and *ad hoc* synthesis of distributed supervisory control from local controllers (LCs) that are obtained using supervisory control theory and aggregated into a global system through constraints expressed by Boolean relations on local inputs and outputs. In [18], a method

Manuscript received February 2, 2019; revised April 6, 2019; accepted April 17, 2019. Manuscript received in final form April 19, 2019. This work was supported in part by the Office of Naval Research (ONR) under Grant N00014-17-1-2012 and Grant N00014-17-1-2504, in part by NSF under Grant CNS-1652544, and in part by the Serbian Ministry of Education, Science and Technological Development under Grant TR35004. Recommended by Associate Editor K. Rudie. *(Corresponding author: Zivana Jakovljevic.)*

Z. Jakovljevic is with the Faculty of Mechanical Engineering, University of Belgrade, 11000 Belgrade, Serbia (e-mail: zjakovljevic@mas.bg.ac.rs).

V. Lesi and M. Pajic are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: vuk.lesi@duke.edu; miroslav.pajic@duke.edu).

S. Mitrovic is with Lola Institute, 11000 Beograd, Serbia (e-mail: stefan.mitrovic@li.rs).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCST.2019.2912776

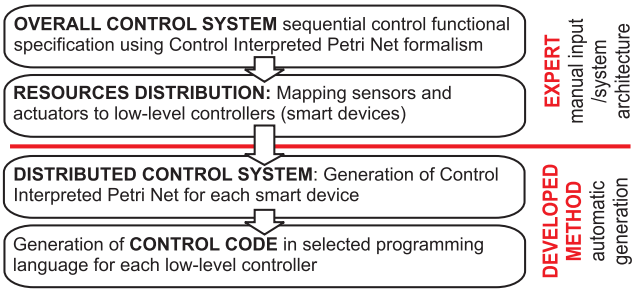

Fig. 1. Distributed control framework for sequential discrete-event systems.

for distributed control of discrete event dynamic systems is introduced, based on the use of timed continuous Petri nets, where the interaction between LCs modeled as Petri nets is captured through special places called buffers. In addition, Zhang *et al.* [19] present a method for modeling of a virtual coordinator and communication protocols for reconfigurable coordination of distributed discrete-event control system using timed net condition/event systems, which is a modular extension of Petri nets. Automatic layout recognition after a reconfiguration, using infrared communication, is proposed in [20], but the control program still needs to be revised by a human operator. Finally, a service-oriented SystemJ programming framework [21] represents another approach to manufacturing systems dynamic reconfiguration.

On the other hand, all such methods, including the IEC 61499 standard, are *bottom-up*. They start from a representation of LCs and generate appropriate interactions between them to achieve seamless operation of the overall system. Consequently, in this brief, we introduce a top-down approach to design distributed control for sequential discrete-event dynamical systems in manufacturing automation (Fig. 1). Unlike the existing bottom-up methods, our approach supports backward compatibility and enables the proliferation of distributed control in everyday practice—we start from a specification of the global controller in an intuitive form frequently employed in practice.

Specifically, we start from a high-level description of the control system based on control interpreted Petri nets (CIPNs), where the system is described in the traditional (centralized) manner, as if all sensors and actuators are connected to the central controller; thus we refer to this representation as a global CIPN. Each smart device due to its communication and computation capabilities represents an LC. Thus, we use physical connections of the system's sensors and actuators to actual smart physical devices, as their mapping to LCs. Since our goal is to decentralize execution of any CIPN-specified global controller by automatically creating a specification of LCs, the presented method automatically generates a CIPN for each LC. Finally, we show how such LCs, defined by CIPNs, can be used to automatically obtain low-level code that can be directly executed on the local smart devices.

This brief is structured as follows. In Section II, we present CIPNs and their use in behavioral specification of sequential control of discrete-event systems in manufacturing automation. Section III presents our method for generation of CIPNs and executable code for LCs, whereas in Section IV, real-world case studies are described, before concluding remarks (Section V).

II. Control Interpreted Petri Nets

Before introducing CIPNs that will be used in this brief, we provide a brief overview of Petri nets. A Petri net is defined [22] as a five-tuple $\mathbf{pn} = (P, T, F, Weig, \mathbf{M}_0)$, where: (1) $P = \{P_1, P_2, \ldots, P_m\}$ is a finite set of places; (2) $T = \{T_1, T_2, \ldots, T_n\}$ is a finite set of transitions with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$; (3) $F \subseteq \{P \times T\} \cup \{T \times P\}$ is the set of arcs between places and transitions; (4) $Weig : F \to \{0, 1 \ldots\}$ is a weight function, and (5) $\mathbf{M}_0 : P \to \{0, 1, 2 \ldots\}$ is the initial marking. Petri nets can be represented as bipartite graphs in which places are denoted by circles and transitions as bars.

The state of a Petri net is defined by the marking represented as an $m \times 1$ vector $\mathbf{M}$ that assigns $\mathbf{M}(P_i) \in [0, \infty)$ tokens to each place P_i. The dynamic behavior, i.e., the evolution of markings $\mathbf{M}_k$ of Petri net is defined by the incidence matrix $\mathbf{W}$, initial marking $\mathbf{M}_0$, and the following state equation:

$$\mathbf{M}_k = \mathbf{M}_0 + \mathbf{W}^T \sum_{i=1}^{k} \mathbf{u}_i. \tag{1}$$

Here, $\{\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_k\}$ represents transitions firing sequence and $\mathbf{W} = [w_{ij}]$ is $m \times n$ matrix whose elements are defined as $w_{ij} = w_{ij}^{+} - w_{ij}^{-}$ where $w_{ij}^{+} = Weig(i, j)$ is the weight of the arc from the transition i to its output place j, and $w_{ij}^{-} = Weig(i, j)$ is the weight of the arc to the transition i from its input place j. Throughout this brief, we use $\bullet T_j$ and $T_j \bullet$ to denote all places preceding and succeeding transition T_j, respectively, and $\bullet P_j$ and $P_j \bullet$ to denote all transitions preceding and succeeding place P_j, respectively.

CIPNs represent a special type of Petri nets in which transition firings are synchronized with the inputs from controlled systems (sensory information), and actions of the system (outputs) are associated with the Petri net places [23]. In this brief, to simplify our notation, we focus on Boolean inputs/outputs. Thus, we define CIPN as follows.

CIPN is a six-tuple CIPN = $(P, T, F, C, A, \mathbf{M}_0)$, where: (1) and (2) P and T are defined as for a Petri net **pn**; (3) $F \subseteq \{P \times T\} \cup \{T \times P\}$ is the set of arcs between places and transitions, all with weights being equal to 1; (4) $C = \{C_1, \ldots, C_n\}$ is the set of logical conditions C_i each associated with the transition T_i, i.e., C_i represents a Boolean function $C_i = f_i(I_1, \ldots, I_k)$, where I_j, $j = 1, \ldots, k$, represent one of k Boolean inputs from the controlled system each corresponding to the sensor X_j; (5) $A = \{A_1, \ldots, A_m\}$ is the set of actions A_i, each associated with place P_i, i.e., A_i represents a set of Boolean or some other functions (e.g., delays) of O_j, i.e., $A_i = \{f_{i,1}(O_1), \ldots, f_{i,l}(O_l)\}$ where O_j, $j = 1, \ldots, l$, represent l outputs to the controlled system, each corresponding to the actuator Y_j; and (6) $\mathbf{M}_0 : P \to \{0, 1\}$ is the initial marking, where exists exactly one $P_i \in P$ such that $\mathbf{M}_0(P_i) = 1$. Note that if condition C_j in transition T_j is not specified, then $C_j = 1$ (i.e., it is always true). Also, if P_j contains no action A_j, then the outputs will not change when P_j is marked.

Thus, CIPNs represent ordinary Petri nets, where all arcs have weights equal to 1. In addition, in a CIPN only one place

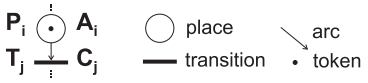

Fig. 2. Graphical representation of places (P_i) and transitions (T_j) in CIPNs. A_i: corresponding actions. C_j: conditions.

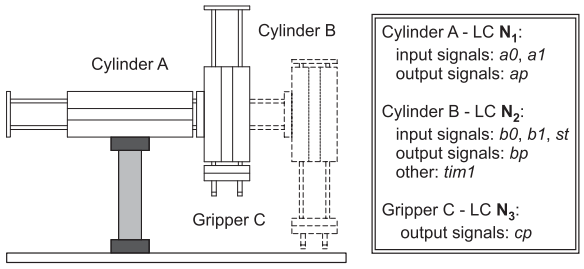

Fig. 3. Example 1: configuration of the manipulator.

initially contains a token. Graphical representation of places, transitions and arcs in CIPNs along with associated conditions and actions is shown in Fig. 2 [23]. In the following example, we illustrate how a CIPN can be used to specify a centralized control system for a simple pneumatic manipulator.

Example 1: Consider a pneumatic manipulator (Fig. 3) that consists of two double-acting cylinders, A and B, and a gripper C; the cylinders provide translational degrees of freedom, and both the cylinders and the gripper are controlled by monostable dual control valves 5/2 (i.e., with five ports and two positions) activated by output signals $xp \in \{ap, bp, cp\}$.[1] The double-acting cylinders are also equipped with proximity sensors for detecting retracted (home)—$x0 \in \{a0, b0\}$ and advanced (end)—$x1 \in \{a1, b1\}$ limit position. Timer $tim1$ ensures gripping/releasing of a part before return stroke of cylinder B. Finally, the system has a start switch (st) to start the operation.

The work cycle of the pneumatic manipulator, started by pressing the start switch, can be described by the sequence

$$B + C + B - A + B + C - B - A - . \qquad (2)$$

Note that $X+$ denotes advancing and $X-$ retracting cylinder X (in this example, $X \in \{A, B, C\}$), while $\left\{ \begin{matrix} X \\ Y \end{matrix} \right\}$ denotes parallel (i.e., concurrent) actions and XY sequential actions.

A global CIPN representing sequential control functionality of the manipulator is shown in Fig. 4(a), capturing the behavior of the system from (2). In this CIPN, P_1 is initially marked and from this place commands ($ap = 0$, $bp = 0$, and $cp = 0$) for initial retracting of all cylinders are issued. When the start switch st is pressed ($st = 1$), transition T_1 fires and marks P_2 which issues the command $bp = 1$ for cylinder B advancing [$B+$ in (2)]. After cylinder B reaches the end position, condition $b1 = 1$ is fulfilled, and thus T_2 fires and marks P_3. P_3 realizes $C+$ from (2) by issuing command $cp = 1$ and activates timer $tim1$ to ensure gripping of the part before return stroke of cylinder B. On the rising edge of $t1$ associated with $tim1$, T_3 fires and marks P_4, from which the command $bp = 0$ for retraction of cylinder B [$B-$ in (2)]

[1]In this brief, we use lower letter x to denote the signals related to the asset denoted in capital letter X

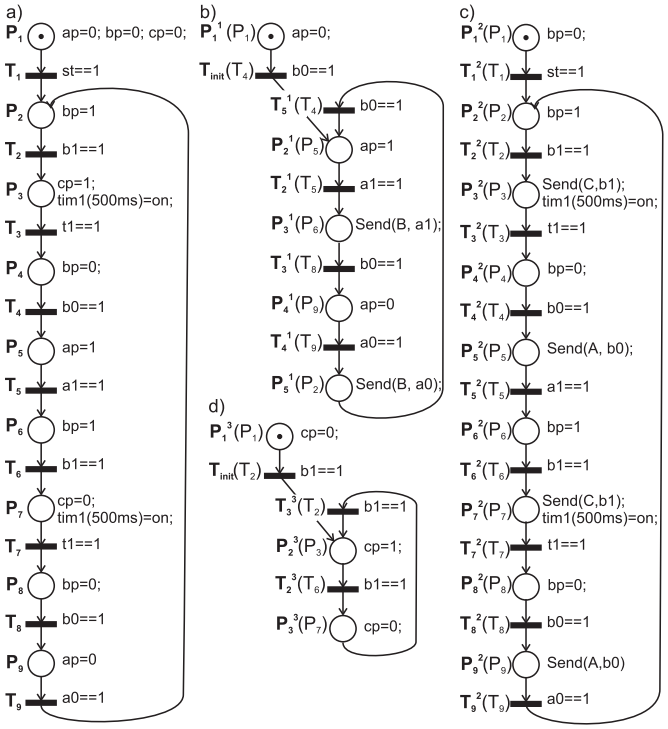

Fig. 4. CIPNs for Example 1. (a) Global CIPN for centralized control. (b) CIPN_1. (c) CIPN_2. (d) CIPN_3. In (b)–(d), places and transitions from the global CIPN are denoted in parenthesis. We use the standard notation: $y = 1$ denotes an assignment (i.e., action), while $x = 1$ denotes a logical condition.

is issued. In a similar manner, the remaining transitions and places along with the associated conditions and actions ensure cyclic realization of the sequence from (2) without additional st pressing. Note that logical conditions and actions assigned to transitions and places in the global CIPN are designed by an expert. □

Finally, note that the CIPN actions associated with its places represent actuation inputs, whereas conditions associated with the transitions represent sensor outputs of the controlled system (i.e., the plant). In this brief, the plants are sequential systems whose outputs depend on the current input and sequence of past inputs, i.e., which are causal. This implies that the current input to the CIPN (transition conditions fulfillment) will depend on the sequence of actions, i.e., on the sequence of the CIPN markings. If we represent the whole set of CIPN arcs as $F = F_P \cup F_T$ where $F_P \subseteq \{P \times T\}$ and $F_T \subseteq \{T \times P\}$, we can observe that the arcs from F_T represent the reaction of the control system (CIPN) to the plant's outputs, whereas the arcs in F_P represent the reaction of the plant to the sequence of actions imposed by the control system (CIPN), e.g., for the manipulator from Fig. 3, action $B-$ must precede activation of sensor $b0$ that detects when the cylinder is fully retracted.

III. Generating LCs From Global CIPNs

Control system built on intelligent devices can be regarded as a network of LCs that communicate with each other, where each LC is executed on a smart device with direct physical access to some system sensors or actuators; we assume that each LC contains at least one sensor or actuator. Our goal is to

distribute control functionality to LCs by deriving local control behaviors such that the obtained decentralized control systems comply with the overall centralized controller specification.

Our method starts from a global CIPN that describes the overall system, with the goal of obtaining local CIPN$_i$, that describe functioning of LC $\mathbf{N}_i$, for each $i \in \{1, \ldots, Q\}$. We assume that each LC $\mathbf{N}_i$ has already been assigned sensors that provide set of input signals $\mathbf{I}_i = \{I_{i,j}\},\ j \in \{1, \ldots, k_i\}$ and/or actuators that are operated using set of output signals $\mathbf{O}_i = O_{i,j},\ j \in \{1, \ldots, l_i\}$. In addition, it holds that

$$\sum_{i=1}^{Q} k_i = k, \quad \bigcup_{i=1}^{Q} \mathbf{I}_i = \{I_1, \ldots, I_k\}, \ \bigcap_{i=1}^{Q} \mathbf{I}_i = \emptyset \tag{3}$$

$$\sum_{i=1}^{Q} l_i = l, \quad \bigcup_{i=1}^{Q} \mathbf{O}_i = \{O_1, \ldots, O_l\}, \ \bigcap_{i=1}^{Q} \mathbf{O}_i = \emptyset. \tag{4}$$

The method to derive each local CIPN$_i$, $i = 1, \ldots, Q$ from the global CIPN is summarized in Procedure 1. First, the initialization step of the procedure adds the place initially marked in CIPN to CIPN$_i$; this is the place P_1. Furthermore, this step finds the succeeding transitions and adds them along with arcs from P_1 to CIPN$_i$, if these transitions will not be added in Step 2. The main part of the procedure (Step 2) extracts all the places from the global CIPN whose actions' outputs are associated with the LC $\mathbf{N}_i$ (Step 2a.i). Places with empty actions are also considered; if any succeeding transition from such a place contains inputs assigned to $\mathbf{N}_i$, or if there exists a succeeding transition with the empty condition, then the place is also extracted (Step 2a.ii). The extracted places are, along with the preceding transitions, mapped into CIPN$_i$.

Transitions from the global CIPN whose conditions' inputs are associated with the LC $\mathbf{N}_i$ are also considered (Step 2b); they are, along with the succeeding places, mapped into CIPN$_i$ if this was not done in Step 2a. If the identified succeeding places contain output signals assigned to other LCs (not $\mathbf{N}_i$), then communication commands are added to these places to send information about the change of sensor signals during the transition; the information is sent to the LCs that are assigned actuators from the succeeding place. In addition, if a succeeding place has an empty action, then the information is sent to all LCs. This way, the information about change of inputs in one LC is made available at the transitions receptive to these inputs in other LCs. Finally, in Step 3, all missing arcs between places and transitions on the token paths from the global CIPN are established within the CIPN$_i$.

Note that in the given procedure, places P_j in CIPN$_i$ keep only those parts of actions A_j that involve actuators assigned to $\mathbf{N}_i$, while transitions T_j keep all the conditions C_j if any of the succeeding places has action with an actuator assigned to $\mathbf{N}_i$, or only those parts of C_j that involve sensors assigned to $\mathbf{N}_i$, otherwise. Thus, the aforementioned information transmissions are necessary; in conditions that keep all sensory signals, input signals from sensors assigned to $\mathbf{N}_i$ are directly obtained within its LC, while the signals from remote LCs are acquired using the employed communication protocols.

Furthermore, as result of the procedure, one place can belong to multiple CIPN$_i$ depending on its action and preceding transition, i.e., place P_j will belong to each CIPN$_i$ corresponding to $\mathbf{N}_i$ that contains actuators with output signals used in action A_j, as well as to each CIPN$_i$ corresponding to $\mathbf{N}_i$ that contains sensors with input signals used in condition allocated to preceding transition. Similarly, a transition can belong to multiple CIPN$_i$, depending on the sensors used in associated conditions and succeeding places.

Procedure 1 Deriving CIPN$_i$ for LC $\mathbf{N}_i$ Starting From the Overall Control Specification CIPN

Step 1. Start from the place P_1 containing the token in the initial marking $\mathbf{M}_0$ and take it into CIPN$_i$; if P_1 action A_1 contains output signal $O_{i,j}$ assigned to $\mathbf{N}_i$ take $\bullet P_1$ along. If a $T_l \in P_1\bullet$ condition C_l does not contain inputs $I_{i,l}$ assigned to $\mathbf{N}_i$ and neither of $P_l \in T_l\bullet$ contains output signal $O_{i,l}$ assigned to $\mathbf{N}_i$ keep moving the token through the CIPN as if all transitions' conditions are true, until a place P_j with action A_j that contains output signal $O_{i,j}$ assigned to $\mathbf{N}_i$, or transition T_j with condition C_j that contains input signal $I_{i,j}$ assigned to $\mathbf{N}_i$ is found; take $\bullet P_j$ or T_j (the first encountered) into CIPN$_i$ and create the arc from P_1 to this transition; mark the transition as initialization transition T_{init}.

Step 2. Start from the $P_1\bullet$ and keep moving the token through the CIPN as if all transitions' conditions are true, until all places and transitions in the CIPN have been visited.

a) If we are considering a place, let us denote the place as P_j.
   i) If the corresponding action A_j contains output signal $O_{i,j}$ assigned to $\mathbf{N}_i$, add the place P_j and all transitions $\bullet P_j$ (with whole associated condition) along with the arc(s) between them to CIPN$_i$.
   ii) If P_j does not contain associated action A_j (i.e., action is empty), and there exists a transition $T_k \in P_j\bullet$ whose condition C_k contains input signal assigned to $\mathbf{N}_i$ or C_k is an empty condition (i.e., always true), then add P_j and all transitions $\bullet P_j$ (with the whole condition) along with the arc(s) between them to CIPN$_i$.

b) If we are currently considering a transition, let us denote the transition as T_j. Now, if T_j's corresponding condition C_j contains input signal $I_{i,j}$ assigned to $\mathbf{N}_i$ add T_j into CIPN$_i$. If none of the $P_h \in T_j\bullet$ actions contains output signals from $\mathbf{N}_i$, in transition condition C_j keep only parts referring to $I_{i,j}$ from $\mathbf{N}_i$, otherwise take whole C_j. Furthermore:
   i) If for a place $P_h \in T_j\bullet$ corresponding action A_h contains output signals $O_{l,h}$ that are assigned to nodes $\mathbf{N}_l$ (not to $\mathbf{N}_i$), add P_h to CIPN$_i$ with action containing command to send the information about $I_{i,j}$ change to all $\mathbf{N}_l$.
   ii) If for a place $P_h \in T_j\bullet$, the corresponding action A_h is empty, add P_h to CIPN$_i$ with action containing command to send the information about $I_{i,j}$ change to all LCs containing inputs from $P_h\bullet$ conditions, i.e., to all LCs if $P_h\bullet$ has empty condition, except to $\mathbf{N}_i$.

Step 3. Connect all missing arc loops on each token path between places and transitions assigned to CIPN$_i$ based on the arc loops from CIPN. Furthermore, for each place P_j in CIPN$_i$ add parts of action A_j that contain output signal $O_{i,j}$ assigned to $\mathbf{N}_i$.

In addition, the procedure will map all places and transitions from the global CIPN to at least one of local CIPN$_i$. For example, Step 2a assures that both all places with nonempty actions, as well as all places with empty actions are assigned to at least one LC; specifically, any place with the empty action, no matter whether it is followed by a transition with the empty condition or with a condition whose input signals are assigned to some LCs, will be mapped to the corresponding LCs (i.e., the place will be mapped to at least one LC). Finally, since places are always assigned along with the preceding transitions, it follows that all transitions are assigned to at least one of the local CIPN$_i$. To illustrate this, consider, e.g., place P_2 from Fig. 4(a). The place will be assigned along with T_1 and T_9 to the CIPN of the LC that contains output *bp* [as shown in Fig. 4(c), described in the continued Example 1 below].

Example 1 (Continued): Control of the system from Fig. 3 is distributed over three LCs: $\mathbf{N}_1$, $\mathbf{N}_2$, and $\mathbf{N}_3$. The mapping of sensors and actuators, i.e., input and output signals mapping to the LCs, is shown in Fig. 3. Using Procedure 1, from the global CIPN [Fig. 4(a)], local CIPN$_i$s [Fig. 4(b)—(d)] were obtained. □

Therefore, starting from the global CIPN, each local CIPN$_i$, $i \in \{1, \ldots, Q\}$ that is obtained using Procedure 1 can be formally specified by the following six-tuple.

1) $P^i = \{P_1^i, P_2^i, \ldots, P_{m_i}^i\}$ where $P^i \subseteq P$ and $\forall P_j^i$ it holds that A_j^i contains only $f_k(O_k)$ such that $O_k \in \mathbf{O}_i \cup \emptyset$.
2) $T^i = \{T_1^i, T_2^i, \ldots, T_{n_i}^i\}$ where $T^i \subseteq T$ and if $P_j^i = P_l$ and P_j^i contains $A_j^i \in A$, then $\bullet P_l = \bullet P_j^i$ and $\bullet P_l \subseteq T^i$.
3) $A^i = \{A_1^i, A_2^i, \ldots, A_{m_i}^i\}$ where $\forall A_j^i$ the following holds: if $P_j^i = P_l$, then

$$A_j^i = \left(\bigcup_{O_k \in \mathbf{O}_i} f_{l,k}(O_k) \right) \cup \Im_{x,j}^i$$

where $\Im_{x,j}^i$ is the set of transmission commands from the LC $\mathbf{N}_i$ to $\mathbf{N}_f$ in the form $\texttt{Send}(\mathbf{N}_f, I_{i,q})$—see Step 2b.
4) $C^i = \{C_1^i, C_2^i, \ldots, C_{n_i}^i\}$ where $\forall C_j^i$ it holds that if $T_j^i = T_l$, then $C_j^i \subseteq C_l$;
5) $F^i = F_1^i \cup F_2^i \cup F_3^i \cup F_4^i$, where: (i) $F_1^i = \bigcup_{T_f \in \bullet P_l}(T_f, P_l)$, $\forall P_j^i = P_l$, where P_l was extracted in Step 2a; (ii) $F_2^i = \bigcup_{P_l \in T_f \bullet}(T_f, P_l)$, $\forall T_j^i = T_f$, where T_f and P_l were extracted in Step 2b; (iii) $F_3^i = \bigcup_{T_f \in P_l \bullet}(P_l, T_f)$ for all $P_j^i = P_l$ such that $T_f \in T^i$ (Step 3); (iv) $F_4^i = \bigcup(P_j^i, T_l^i)$ such that there exist (P_j^i, T_{x1}), $(T_{x1}, P_{x1}), (P_{x1}, T_{x2}), \ldots, (T_{xn}, P_{xn}), (P_{xn}, T_l^i) \in F$, such that $T_{x1}, T_{x2}, \ldots, T_{xn} \notin T^i$, and $P_{x1}, P_{x2}, \ldots, P_{xn} \notin P^i$; in such case, we refer to P_j^i as a transmitting place with sending command(s) $\Im_{x,j}^i$;
6) $\mathbf{M}_0^i$ is the initial marking in which the place P_1^i is marked.

It is important to note that the arcs from sets F_1^i, F_2^i, and F_3^i are present in the initial CIPN, whereas the arcs from F_4^i are introduced during the derivation of the local CIPN$_i$. Furthermore, the arcs $(T_f, P_l) \in F_2^i$ are present not only in CIPN$_i$ but also in all CIPN$_j$ to which information is sent from P_l. For example, consider CIPN$_1$ presented in Fig. 4(b); here $F_1^1 = \{(T_4, P_5), (T_8, P_9)\}$, $F_2^1 = \{(T_5, P_6), (T_9, P_2)\}$, $F_3^1 = \{(P_5, T_5), (P_9, T_9)\}$, and $F_4^1 = \{(P_1, T_4), (P_6, T_8), (P_2, T_4)\}$, where we use the notation from the global CIPN (i.e., the place/transition names specified in the parentheses). Thus, the CIPN from Fig. 4(a) contains all arcs from F_1^1, F_2^1, and F_3^1, whereas arcs from F_4^1 are added during the derivation of CIPN$_1$. In addition, the arcs from F_2^1 are present in F_1^2 [Fig. 4(c)].

Procedure 1's computational complexity is $O[(m+n)S]$ for each of the Q LCs, where S is a total number of sensors and actuators. Nevertheless, the procedure is carried out offline, during system design (or reconfiguration) phase. On the other hand, the complexity of each derived CIPN$_i$, which are used to generate control software for each smart device is lower than or equal to the complexity of the global CIPN, thus not imposing additional timing constraints on the system implementation.

A. Properties of Derived Local Controllers

The distribution of control functionality to LCs raises the issue of safeness and liveness of the obtained distributed system. Note that the initial, centralized controller is modeled by a global CIPN, which is 1-bounded (i.e., safe) and deterministic [23]. We also assume that by design, the global CIPN satisfies liveness criterion; thus, the initial system is both safe and live. In what follows, we argue that the behavior of the controlled system imposed by the global CIPN is preserved by the composition of the derived local CIPN$_i$, $i = 1, \ldots, Q$.

From the definition of CIPN and formal representations of all CIPN$_i$, $i = 1, \ldots, Q$, the following directly holds:

$$P = \bigcup_{i=1}^{Q} P^i, \quad T = \bigcup_{i=1}^{Q} T^i, \; C = \bigcup_{i=1}^{Q} C^i, \; A = \bigcup_{i=1}^{Q} A^i.$$

On the other hand, the set of arcs F does not represent the simple union of F^i. Nevertheless, it holds that $\bigcup_{i=1}^{Q} F_1^i = F_T$, while the union of F_3^i represents only a subset of F_P, i.e., generally arcs from F_P are not present in union of arcs from CIPN$_i$. However, by construction (Procedure 1, Step 3) and through the introduction of transmission commands, the sequence of CIPN actions (controlled system inputs) is preserved. Thus, since the controlled system is causal, the sequence of its outputs (CIPN conditions fulfillment) will be also preserved, and the behavior of the plant is the same when control is distributed to CIPN$_i$, as with the centralized control (CIPN).

Due to space constraint, we provide only a proof outline. Let us assume that control with the global CIPN and the composition of the local CIPN$_i$s has resulted in the same sequence of actions until place P_1 in the CIPN is active, and the corresponding places in local CIPN$_i$s. This is satisfied in the initial places of CIPN and CIPN$_i$s, and thus for the control equivalence to hold, it is enough to show that the sequences of control actions (and thus the plant's outputs) for the centralized and distributed control remain identical until a next place P_2 in the CIPN is reached over a transition T_1, as well as the corresponding places in the CIPN$_i$s. To achieve this, it is necessary to consider all possible combinations of the mapping

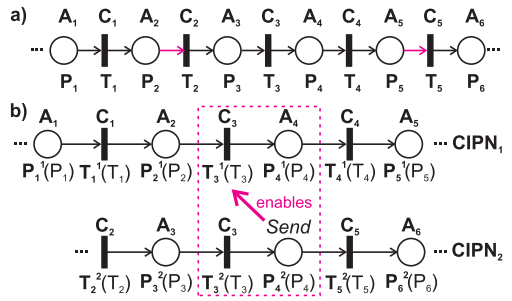

Fig. 5. Composition of local CIPN$_i$. (a) Part of a global CIPN. (b) Places and transitions mapped from the CIPN to LCs CIPN$_1$ and CIPN$_2$ (places and transitions notations in the CIPN are denoted in parentheses).

of places P_1 and P_2, transition T_1, and the corresponding arcs of CIPN into the LCs specified by the CIPN$_i$s, as captured in the cases of Procedure 1.

To illustrate this on an example, let us consider a global CIPN from Fig. 5(a), for which Procedure 1 produces two LCs, specified by CIPN$_1$ and CIPN$_2$ [Fig. 5(b)]. Comparison of the CIPN with CIPN$_1$ and CIPN$_2$ shows that the arcs (P_2, T_2), (P_5, T_5) from CIPN are present neither in CIPN$_1$ nor in CIPN$_2$. Nevertheless, due to the information transmission command in P_4^2, and by the construction of CIPN$_i$, the sequence of actions and transitions firing will be preserved. For example, since the control system is causal, T_2^2 condition C_2 will not be enabled and T_2^2 will not fire before A_2 is performed (P_2^1 is marked). The same holds for P_5^1 and T_5^2. Thus, with distributed control, the sequence of actions and conditions (sensor inputs) will be the same as in the case of CIPN, i.e., $A_1C_1A_2C_2A_3C_3A_4C_4A_5C_5A_6C_6$. Similarly, in Example 1 LCs CIPN$_i$ will impose the sequence defined by (2).

Furthermore, the behavioral equivalence of the global (i.e., centralized) control and the composition of distributed controllers can be also formally verified with the use of model checking techniques and the corresponding tools. However, since the control models feature conditions on sensor values, suitable models of the plant have to be taken into account [24]. For example, the model of a double-acting cylinder from Fig. 3 has two places representing cylinder's limit positions, and two transitions corresponding to cylinder advancing/retracting whose firing is enabled on the corresponding LC commands.

CIPNs can be directly analyzed, e.g., by transforming them to logical models for which properties captured in Linear Temporal Logic or Computation Tree Logic can be verified [25], or through their modeling in a Hardware Description Language and simulation in a suitable tool [26]. Yet, since CIPN and its Grafcet interpretation have exactly the same behavior [23], and Grafcet models can be modeled as time Petri nets (TPN) [27] for which open verification tools are available [28], we have opted to employ TPNs to enable model verification. In TPN, transitions are *timed*, meaning that transition firing takes a time specified for each transition. The interface between the controller and plant models is achieved through the introduction of marking-dependent *guard functions* (specified within the model) which allow transitions in one (e.g., plant) model to be explicitly conditioned on the marking of another (e.g., LC) model. Similarly, communication between LCs is modeled via guard functions [e.g., the receiving $\mathbf{N}_i$ on its receiving transition with condition $I_{j,q} = 1/0$ guards the marking of the place issuing `Send`($\mathbf{N}_i$, $I_{j,q}$) in the transmitting $\mathbf{N}_j$]. On such models, TPN analysis tools support verification of formally specified properties by examining the underlying state space. Due to semantical equivalence, the translation of the CIPN-based to TPN-based LC models is straightforward.

For the models from Example 1, we formally verified relevant behaviors from the process control perspective using Romeo, a TPN model checking tool [28]. For instance, in addition to deadlock absence and safeness, the considered behaviors included requirements that: 1) gripper C is never closed when cylinder B advances toward the pick position; 2) gripper C is always closed while cylinder A advances; and 3) cylinder B is always retracted while cylinder A is active. Specifically, we formally verified that all these properties, which were satisfied when the plant was controlled with the global CIPN, were also satisfied when the distributed controllers specified by CIPN$_i$s were used. Furthermore, we verified that the only possible trace is the trace from (2), ensuring that the desired system behavior is preserved after moving from centralized to the distributed control.

B. Code Generation From Local CIPN$_i$s

There are a number of methods for generation of control code from CIPNs. For example, [29] presents a method for IEC 61131-3 Ladder Diagram generation, [14] proposes a similar procedure for transforming hierarchical Grafcets into programmable logic controller (PLC) code, the work in [24] employs an approach for IEC 61131-3 Instruction List generation, whereas in [25] CIPNs are transformed to the Register Transfer Level that can be used for code synthesis. In this brief, we have opted to employ procedure from [29] that can be easily adapted to another programming language. Due to the intended type of application, we have chosen to generate C code using the following approach:

1) Assign a Boolean variable m_i to each place P_i in CIPN.
2) Introduce initialization by adding variable m_0 that is set if the condition C_k assigned to $\bullet P_{init}$ is true (P_{init} is marked in $\mathbf{M}_0$). Set m_{init} simultaneously with m_0.
3) For each place P_i in CIPN that corresponds to m_i.
   a) If m_i is active, carry out actions A_i associated with P_i including sending commands.
   b) If m_i is active and condition C_l corresponding to a $T_l \in P_i\bullet$ is true, reset m_i and set m_l, for all $P_l \in T_l\bullet$; reset values of all LC external sensors that led to the transition.
   c) If P_i contains command for sending sensor input I_k to LC $\mathbf{N}_j$, associate pm_i to m_i to assure that the sending command is executed only once.

In Step 3b, the values of all LC's external sensors that have led to the transition have to be reset to avoid conflict since the LC will not necessarily receive the change of external sensor value. An example of the code generated for the CIPN$_1$ from Fig. 4(b) for the first three places is given in Fig. 6.

Init	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$
`if(!m0)`	`if (m1)`	`if (m2)`	`if (m3 && pm3)`
`{m1=1;`	`{ap=0;}`	`{ap=1;}`	`{Send(B,a1);`
`m0=1;}`	`if (m1 && b0)`	`if (m2 && a1)`	`pm3=0;}`
	`{m2=1;`	`{m3=1;`	`if (m3 && c0)`
	`m1=0;`	`pm3=1;`	`{m4=1;`
	`b0=0;}`	`m2=0;}`	`m3=0; c0=0;}`

Fig. 6. C code generated for CIPN_1 from Fig. 4(b) for the first three places. Send commands should be based on the employed communication API.

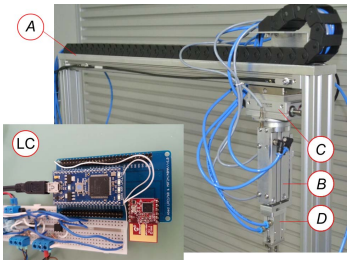

Fig. 7. Case study 1. A photograph of pneumatic manipulator with three degrees of freedom (A—translation; C—rotation; and B—translation) and gripper D; LC is MCU based with IEEE 802.15.4 compatible transceivers.

IV. Case Studies

We have tested the performances of our method using two real-world case studies. The first focuses on a pneumatic manipulator with a similar structure but higher complexity than the manipulator described in Example 1. The second case study considers the prototype of a manipulator whose control requires the parallel execution of tasks. In our experiments, for control of system components and for distribution of control tasks we used a microcontroller (MCU)-based wireless nodes with IEEE 802.15.4 compatible transceivers.

A. Case Study 1: Pneumatic Manipulator

We consider a pneumatic manipulator (Fig. 7) that immerses parts into liquid. The manipulator has three degrees of freedom in the configuration: translation—rotation—translation and a pneumatic gripper D. Translational degrees of freedom are carried out using two double-acting cylinders (A and B) and for the rotational degree of freedom, rotary cylinder C is used.

All cylinders are controlled by bistable dual control valves 5/2 and they are equipped with proximity sensors—one for each limit position ($x0$, $x1$ for cylinder X, where $x \in \{a, b, c\}$). The gripper is controlled by a monostable dual control valve 5/2, while gripping/releasing of the part before return stroke of cylinder B is ensured by a timer $tim1$. In the desired system operation, the manipulator takes the part from the starting position to the position above the container with liquid, immerses the part, takes it out, shakes off liquid excess by rotational motion, and returns the part into the starting position. The first cycle of manipulation is initiated by the start switch (st). The work cycle of the manipulator is described by

$$B + D + B - A + B + B - C + C - A - B + D - B - . \tag{5}$$

Each actuator (three cylinders and gripper) represents a module (smart device) with its own LC, and with assigned sensors and actuators as given in Table I. Hence, the control system is implemented using four wireless nodes (LCs) based on low-cost ARM Cortex-M3 MCU boards running at 96 MHz (Fig. 7.) The nodes communicate via IEEE 802.15.4-compliant wireless transceivers, with the generated control code directly using the employed communication application program interface (API).

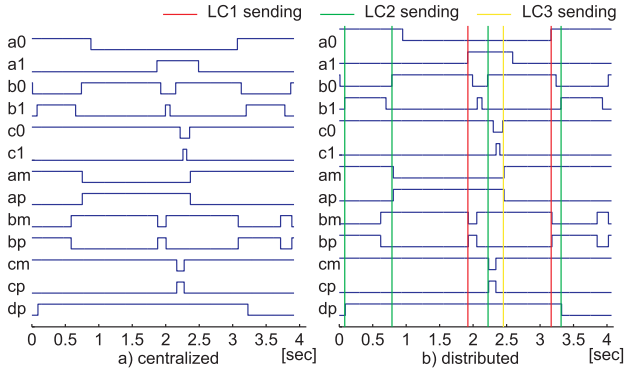

Fig. 8. Case study 1: comparison of the time diagrams capturing changes in the input and output signals for centralized and distributed control during one cycle of the manipulator operation.

TABLE I

Case Study 1: Mapping of Inputs and Outputs to the Nodes

Node	Sensor (input) signals	Output signals	Other
$\mathbf{N}_1$	$a0$, $a1$	ap $(A+)$, am $(A-)$	
$\mathbf{N}_2$	$b0$, $b1$, st	bp $(B+)$, bm $(B-)$	$tim1$
$\mathbf{N}_3$	$c0$, $c1$	cp $(C+)$, cm $(C-)$	
$\mathbf{N}_4$		dp $(D+)$	

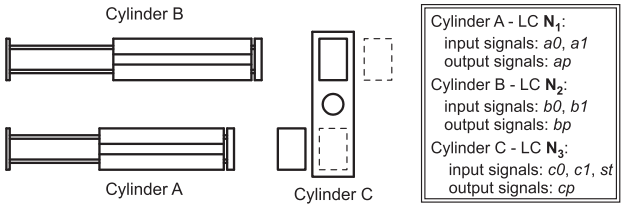

Fig. 9. Case study 2: configuration of the manipulator with parallel processes.

The structure of the control system for this manipulator is similar (although more complex in terms of the number of LCs and output signals per LC) to the structure of manipulator from Example 1. Thus, the obtained CIPN, CIPN_is, and code are similar to those from Figs. 4 and 6, and are thus not shown.

To test the performance of the obtained distributed control system, we have also developed a centralized control system where all sensors and actuators from all modules were assigned to a single MCU implementing the global CIPN, i.e., with wired connections between sensors/actuators and the MCU. Fig. 8 presents the time diagrams of sensor and actuator signals from Table I for one manipulator cycle for both centralized and distributed control. It can be observed that the sequence of input ($x0$ and $x1$) and output (xp and xm) signals is the same for both control modes. Furthermore, Fig. 8(b) presents points of communication between LCs in colored lines. For example, red lines present sending information from LC_1 to LC_2 about $a0$ and $a1$ inputs (assigned to $\mathbf{N}_1$) changes that invoke changes in bm and bp outputs (assigned to $\mathbf{N}_2$).

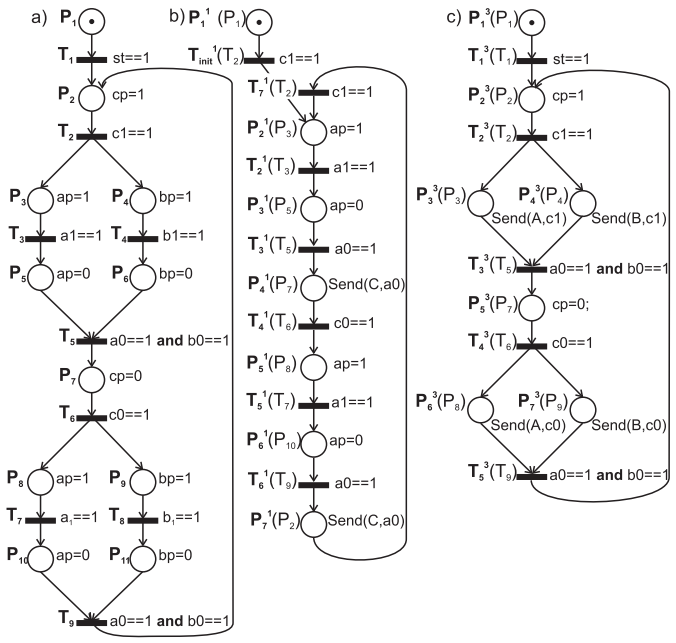

Fig. 10. Case study 2 (manipulator with parallel processes). (a) Global CIPN. (b) CIPN_1 (CIPN_2 has equivalent representation). (c) CIPN_3.

As expected, the communication between nodes introduced some latency, which resulted in a longer duration of the cycle in the case of a distributed system, compared to the centralized. However, it is important to highlight that this delay is mainly caused by the use of low-bandwidth wireless links (802.15.4 compatible) and not due to the control distribution. When, instead of physical wiring, wireless links between the centralized controller and sensors/actuators are used, the same latency levels are also observed in the centralized control setup.

B. Case Study 2: Manipulator With Parallel Processes

Second case study refers to another pneumatic manipulator in which parallel processes are present. This manipulator consists of two double-acting cylinders and a rotary cylinder controlled by monostable dual control valves 5/2. Double-acting cylinders A and B carry out simultaneous loading and unloading of parts to and from the platform placed on rotary cylinder C (Fig. 9.) All cylinders are equipped with proximity sensors, one for each limit position ($x0$, $x1$ for cylinder X, where $x \in \{a, b, c\}$). The system is equipped with a start switch. Finally, the manipulator's work cycle is described by

$$C + \left\{ \begin{array}{l} A+ \\ B+ \end{array} \right\} \left\{ \begin{array}{l} A- \\ B- \end{array} \right\} C - \left\{ \begin{array}{l} A+ \\ B+ \end{array} \right\} \left\{ \begin{array}{l} A- \\ B- \end{array} \right\}. \qquad (6)$$

In this case study, there exist parallel processes (advancing and retracting of cylinders A and B) and they are represented by branches in the global CIPN shown in Fig. 10(a). Distributed control system consists of three LCs—one for each cylinder, and the input-output signals assignment to LCs is shown in Fig. 9. The obtained CIPN_i for LCs $\mathbf{N}_i$ are presented in Fig. 10(b) and (c) (CIPN_2 is equivalent to CIPN_1 due to similar work cycle of the cylinders). As in the first case study, the performance of the distributed system matched the centralized one during testing, meaning that the method is able to adequately distribute control tasks in systems with parallel processes.

V. Conclusion

We have presented a method for automatic generation of distributed controllers for sequential discrete event dynamic systems used in manufacturing automation. Our approach is based on the representation of the control task using the CIPN formalism. Once the control sequence of the overall system as if centrally controlled is defined, and sensors and actuators are mapped to LCs, our method is able to automatically distribute control tasks to LCs as well as generate code for such LCs.

The presented design framework is especially advantageous when the control distribution to LCs is carried out such that intensive communication between LCs, along with a relatively small number of commands between communications is necessary to accomplish the desired functionality. Nevertheless, the description of systems in which a large sequence of commands is executed on one LC (i.e., one module of reconfigurable resource) without communication to other LCs can lead to an unnecessarily complex global CIPN. In such case, a hierarchical structure introduced through a higher level of abstraction in generating actions (such as macro-steps in Grafcet [14]) can reduce the complexity of the global CIPN.

Our approach is independent of the employed communication APIs. Communication induced latencies, erroneous communication and attacks by different adversaries depend on the applied communication protocol, they are beyond the scope of this brief and present an avenue for future efforts.

References

[1] H. ElMaraghy *et al.*, "Product variety management," *CIRP Ann.*, vol. 62, no. 2, pp. 629–652, 2013.

[2] H. Kagermann, W. Wahlster, and J. Helbig. (2013). *Recommendations for Implementing Strategic Initiative INDUSTRIE 4.0*. [Online]. Available: http://www.acatech.de

[3] Y. Koren, X. Gu, and W. Guo, "Reconfigurable manufacturing systems: Principles, design, and future trends," *Frontiers Mech. Eng.*, vol. 13, no. 2, pp. 121–136, 2018.

[4] V. Lesi, Z. Jakovljevic, and M. Pajic, "Towards plug-n-play numerical control for reconfigurable manufacturing systems," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat. (ETFA)*, Sep. 2016, pp. 1–8.

[5] J. Wan, B. Yin, D. Li, A. Celesti, F. Tao, and Q. Hua, "An ontology-based resource reconfiguration method for manufacturing cyber-physical systems," *IEEE/ASME Trans. Mechatron.*, vol. 23, no. 6, pp. 2537–2546, Dec. 2018.

[6] J. Wan *et al.*, "Reconfigurable smart factory for drug packing in healthcare industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 15, no. 1, pp. 507–516, Jan. 2019.

[7] H. Hu and M. Zhou, "A Petri net-based discrete-event control of automated manufacturing systems with assembly operations," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 2, pp. 513–524, Mar. 2015.

[8] Y. Yang, H. Hu, and Y. Liu, "A Petri net-based distributed control of automated manufacturing systems with assembly operations," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Aug. 2015, pp. 1090–1097.

[9] J. Otto, B. Vogel-Heuser, and O. Niggemann, "Automatic parameter estimation for reusable software components of modular and reconfigurable cyber-physical production systems in the domain of discrete manufacturing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 1, pp. 275–282, Jan. 2018.

[10] R. Vrabič, D. Kozjek, A. Malus, V. Zaletelj, and P. Butala, "Distributed control with rationally bounded agents in cyber-physical production systems," *CIRP Ann.*, vol. 67, no. 1, pp. 507–510, 2018.
[11] D. A. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Inst.*, vol. 257, no. 3, pp. 161–190, 1954.
[12] D. A. Huffman, "The synthesis of sequential switching circuits: Part II," *J. Franklin Inst.*, vol. 257, no. 4, pp. 275–303, 1954.
[13] C. Yen and W.-J. Li, "Web-based learning and instruction support system for pneumatics," *Comput. Educ.*, vol. 41, no. 2, pp. 107–120, 2003.
[14] R. Julius, M. Schürenberg, F. Schumacher, and A. Fay, "Transformation of GRAFCET to PLC code including hierarchical structures," *Control Eng. Pract.*, vol. 64, pp. 173–194, Jul. 2017.
[15] *Function Blocks—Part 1: Architecture*, IEC Standard 61499-1:2012, International Electrotechnical Commission Standard, 2012.
[16] Z. Jakovljevic, S. Mitrovic, and M. Pajic, "Cyber physical production systems—An IEC 61499 perspective," in *Proc. 5th Int. Conf. Adv. Manuf. Eng. Technol.*, in Lecture Notes in Mechanical Engineering, 2017, pp. 27–39.
[17] Y. Qamsane, A. Tajer, and A. Philippot, "A synthesis approach to distributed supervisory control design for manufacturing systems with Grafcet implementation," *Int. J. Prod. Res.*, vol. 55, no. 15, pp. 4283–4303, 2017.
[18] L. Wang, C. Mahulea, and M. Silva, "Distributed model predictive control of timed continuous Petri nets," in *Proc. IEEE Conf. Decis. Control*, Dec. 2013, pp. 6317–6322.
[19] J. Zhang, M. Khalgui, Z. Li, G. Frey, O. Mosbahi, and H. Ben Salah, "Reconfigurable coordination of distributed discrete event control systems," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 1, pp. 323–330, Jan. 2015.
[20] J. W. Park, M. Shin, and D. Y. Kim, "An extended agent communication framework for rapid reconfiguration of distributed manufacturing systems," *IEEE Trans. Ind. Informat.*, to be published.
[21] U. D. Atmojo, Z. Salcic, and K. I.-K. Wang, "Dynamic reconfiguration and adaptation of manufacturing systems using sosj framework," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2353–2363, Jun. 2018.
[22] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[23] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*, 2nd ed. Berlin, Germany: Springer, 2010.
[24] G. Frey and L. Litz, "Verification and validation of control algorithms by coupling of interpreted Petri nets," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 1998, pp. 7–12.
[25] I. Grobelna and M. Adamski, "Model checking of control interpreted Petri nets," in *Proc. 18th Int. Conf. Mixed Design Integr. Circuits Syst.*, Jun. 2011, pp. 621–626.
[26] R. Wiśniewski, A. Karatkevich, M. Adamski, A. Costa, and L. Gomes, "Prototyping of concurrent control systems with application of Petri nets and comparability graphs," *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 2, pp. 575–586, Mar. 2018.
[27] M. Sogbohossou and A. Vianou, "Formal modeling of grafcets with time Petri nets," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 5, pp. 1978–1985, Sep. 2015.
[28] G. Gardey, D. Lime, M. Magnin, and O. Roux, "Romeo: A tool for analyzing time Petri nets," in *Proc. Int. Conf. Comput. Aided Verification*. Berlin, Germany: Springer, 2005, pp. 418–423.
[29] J.-S. Lee and P.-L. Hsu, "A systematic approach for the sequence controller design in manufacturing systems," *Int. J. Adv. Manuf. Technol.*, vol. 25, nos. 7–8, pp. 754–760, 2005.