IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 7, 2020, accepted August 25, 2020, date of publication August 28, 2020, date of current version September 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3020213

Extracting Petri Modules From Large and Legacy Petri Net Models

REGGIE DAVIDRAJUH, (Senior Member, IEEE)
Department of Electrical Engineering and Computer Science, University of Stavanger, 4036 Stavanger, Norway
e-mail: reggie.davidrajuh@uis.no

ABSTRACT Petri nets, even though very useful for modeling of discrete event systems, suffer from some weaknesses such as huge size, huge state space, and slow in simulation. Due to the huge state space, model checking a Petri net is difficult. Also, due to the slowness in simulation, discrete-timed Petri nets cannot be used for real-time applications. Thus, modular Petri nets are suggested as a way of overcoming these difficulties. In modular Petri nets, modules are designed, developed, and run independently, and the modules communicate with each other via inter-modular connectors. This approach is suggested for developing newer Petri net models. However, there exists a large number of Petri net models of real-life systems, and these legacy models are enormous and non-modular. And, these models cannot be discarded as large amounts of time and money were spent to develop these models. This paper presents a unique algorithm for extracting modules from large and legacy Petri net models. The algorithm extracts modules (known as "Petri modules") that are well-defined for inter-modular collaboration. Also, the extraction method preserves the structural properties. The goal of the paper is to introduce a methodology by which Petri nets can be moved to a new level in which a modular Petri net model can be made of Petri modules. The Petri modules are independent and can be hosted on different computers. These modules communicate via inter-modular components such as TCP/IP sockets. Since Petri modules are compact, also run faster, thus become suitable for supervisory control of real-time systems.

INDEX TERMS Modular Petri nets, module extraction, Petri modules, legacy Petri nets.

I. INTRODUCTION

Petri nets are a highly effective way of modelling discrete event systems. However, Petri net models of real-life discrete event systems are enormous. Due to the huge size, these models lack overview, run slowly during execution, and performing analysis on the model becomes difficult. Sometimes, slicing of a Petri Net is performed so that the size of the Petri Net is reduced so that the state space (*aka* reachability graph) generated from it is smaller as well. However, the existing slicing algorithms are ineffective for real-world discrete event systems [1]; Also, slicing is not applicable to performance evaluation [2]. Thus, there is a need for alternative methodologies for slicing that are effective for Petri net models of real-life large discrete event systems.

This paper proposes Modular Petri Nets. In modular Petri nets, large Petri net models are decomposed into Petri modules. These Petri modules are compact, and the state spaces of these modules are also compact enough to be exhaustively analyzed. Also, these modules need not run on the same computer, as the modules can be run in parallel on different computers. The specific goal of this paper is to introduce a new algorithm that can extract Petri modules from existing large and legacy Petri net models.

Legacy Petri nets are Petri net models that have been around a long time. Legacy Petri nets are large and are complex (difficult to understand). However, legacy Petri nets are still useful as these Petri nets model real-life systems; these models cannot be discarded as they have incurred some cost for the development (modellers' time and resources).

This paper consists of ten sections. Section-II presents a short literature study on modular Petri nets. Section-III presents a new modular Petri net that comprises of a set of Petri modules and inter-modular connectors. The formal definitions to Petri modules and inter-modular connectors are given in section-IV. And, sections V to VII present the new algorithm for extracting Petri modules from Petri nets. An application of the algorithm, as a proof-of-concept,

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

is given in section-VIII. And, section-IX presents an analysis of the preservation of the structural properties. The paper concludes with a discussion in section-X.

II. LITERATURE STUDY

Earlier studies on modular Petri nets can be classified into four generations:

1) First-generation studies on the ease of modeling. (E.g., [3]–[9]).
2) Second-generation studies on the ease of analysis of Petri net with the help of modules (E.g., [10]–[14]).
3) Third-generation studies that provide tools for specific applications (E.g., [15]–[22]).
4) Fourth-generation studies on the developing independent modules that can function as intelligent autonomous agents (E.g., [23]–[27]).

First-generation studies are about non-modular Petri nets. However, these studies laid the foundation. In [3] and [4], the authors present techniques for compressing Petri net segments (rudimentary modules). A clear-cut interface for Petri net segments are introduced in [5]; [6], [7] introduce object-oriented Petri nets. In [8], the authors propose decomposing a large Petri net into segments based on the functionality, and [9] shows how a Petri net model of a flexible manufacturing systems can be decomposed into segments.

Second-generation studies mostly on the ease of analysis of Petri nets, with the use of modularization or segmenting. In [10], the authors discuss how state space analysis can be simplified by the analyses of the segments, and [11] shows how Petri net models of different systems can be quickly developed by recycling modular components. Similarly, [13] emphasizes reconfigurable modules for the use in different models.

Third-generation studies that provide tools for specific applications. E.g., [15] presents a tool (known as ''Exhost-PIPE'') for modeling multi-agents, and [16] for modeling molecular networks. Papers [17], [18] are on modeling national health system, and [19] is for modeling freight terminals. Paper [20] is for modeling traffic signals, and [21] used modular modeling of manufacturing systems. Finally, [22] presents a tool known as General-purpose Petri nets (GPenSIM).

Fourth-generation studies are the ones that treat modules of modular Petri nets as intelligent autonomous agents. E.g., [26] develops a modular Petri net for detecting cyber-security threats; in this model, the modules are autonomous and intelligent. The studies such as [23]–[25] show that models of autonomous manufacturing systems can be made up of modules that are fed by networked sensors.

III. MODULAR PETRI NETS

Based on the literature study, [28] presents a new modular Petri net, in which a Petri model consists of a set of modules (known as Petri modules) and inter-modular connectors (known as IMCs).

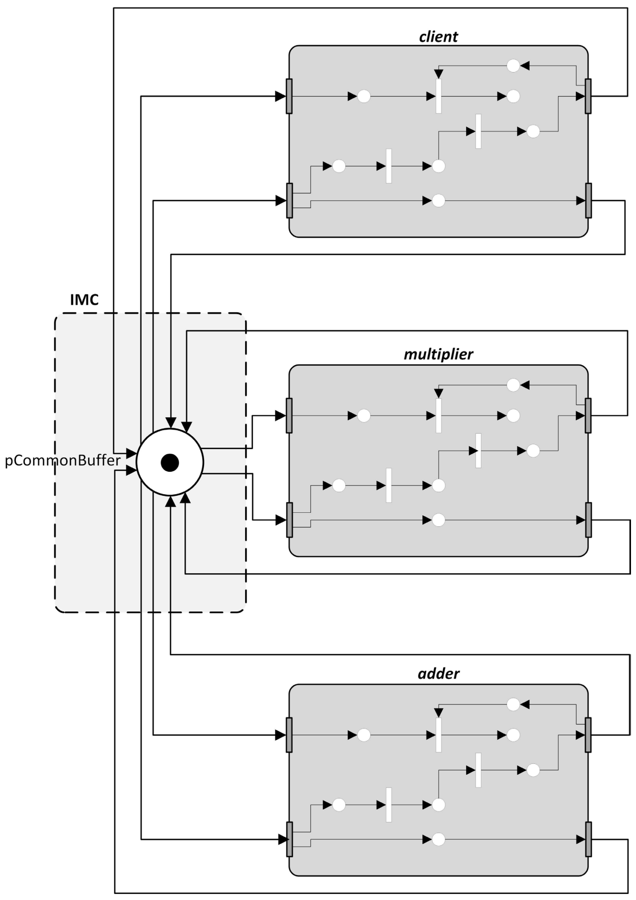


FIGURE 1. A sample modular Petri net for performing the four arithmetic operations (+, −, *, and /). The client creates the jobs and passes them to the multiplier (performs multiplication and division tasks), and the adder (performs addition and subtraction). Place ''pCommonBuffer'' functions as an inter-modular connector.

For example, Fig. 1 shows a sample modular Petri net that is composed of three Petri modules such as client, multiplier, and adder, and only one place (''pCommonBuffer'') that functions as inter-modular connector. This model is to compute the four fundamental arithmetic operations. The client is the one the generate the problem (work load). Adder performs the addition and subtraction operations. And multiplier performs multiplication and division tasks.

In Fig.1, only the place ''pCommonBuffer'' is drawn in a larger size; all other elements are drawn with the same scale. Place ''pCommonBuffer''is drawn in a large size to indicate that it is a special place. Unlike all other elements in Fig.1, the place ''pCommonBuffer'' is an inter-modular connector. Without ''pCommonBuffer'', all other modules can function independently; however, these modules cannot communicate. Hence, ''pCommonBuffer'' is drawn larger to show the importance of it in the communication between the modules.

A. PETRI MODULE

A Petri module has four distinct sets of elements:

- Input ports T_{IP}: Input port transitions function as the input gates of a module. **Only through these transitions (input ports), tokens can be directed into the module**.
- Output ports T_{OP}: Output port transitions function as the output gates of a module. **Only through these transitions (output ports), tokens can be directed away from the module**.
- Local transitions T_L: As the local member (internal element) of a module, a local transition consumes tokens from local input places and deposits tokens into local output places. A local transition cannot have any direct connection with the external places (places outside the modules).
- Local places P_L: As the local member of a module, a local place feeds tokens to either local transitions or input and output ports of the module. A local place gets tokens from either local transitions or input and output ports of the module. A local place cannot have any direct connection with the external transitions.

B. INTER-MODULAR CONNECTORS

A modular Petri net model consists of zero or more Inter-Modular Connectors (IMC). The IMCs are not modules thus don't possess the input and output ports. IMCs possess *IM transitions* and *IM places*.

The algorithm proposed in this is paper for the extraction of Petri modules from legacy models. The extracted Petri modules comply with the definitions given in section-IV. A concise set of formal definitions to the new modular Petri net is given in section-IV.

IV. FORMAL DEFINITIONS

Lets us start with the formal definition for Place-Transition Petri Nets.

A. FORMAL DEFINITION OF P/T PETRI NET

Definition 1 (Place-Transition Petri Net): (P/T Petri Net, for short) is defined as a four-tuple [29]:

$$PTN = (P, T, A, M_0),$$

where,

- Places: P is a finite set of places, $P = \{p_1, p_2, \ldots, p_{n_p}\}$.
- Transitions: T is a finite set of transitions, $T = \{t_1, t_2, \ldots, t_{n_t}\}$. $\mathrm{P} \cap \mathrm{T} = \emptyset$.
- Arcs: A is the set of arcs (from places to transitions and from transitions to places). $A \subseteq (P \times T) \cup (T \times P)$. The default arc weight W of a_{ij} ($a_{ij} \in A$, an arc going from p_i to t_j or from t_i to p_j) is one, unless noted otherwise.
- Marking: m is the row vector of markings (tokens) on the set of places.

$$M = [M(p_1), M(p_2), \ldots, M(p_m)] \in N^{n_p},$$

M_0 is the initial marking. Due to the markings, a $PTN = (P, T, A, M_0)$ is also called a **marked P/T Petri Net**. □

B. FORMAL DEFINITION OF PETRI MODULE

Definition 2 (Petri Module): is defined as a six-tuple [28]:

$$\Phi = (P_{L\Phi}, T_{IP\Phi}, T_{L\Phi}, T_{OP\Phi}, A_\Phi, M_{\Phi 0}),$$

where,

- Input Ports of the module: $T_{IP\Phi} \subseteq T$.
- Local transitions of the module: $T_{L\Phi} \subseteq T$.
- Output Ports of the module: $T_{OP\Phi} \subseteq T$.
- $T_{IP\Phi}$, $T_{L\Phi}$, and $T_{OP\Phi}$, are all mutually exclusive:

$$T_{IP\Phi} \cap T_{L\Phi} = T_{L\Phi} \cap T_{OP\Phi} = T_{OP\Phi} \cap T_{IP\Phi} = \emptyset.$$

- The transitions of the module: $T_\Phi = T_{IP\Phi} \cup T_{L\Phi} \cup T_{OP\Phi}$.
- The local places of the module: $P_{L\Phi} \subseteq P$. Since a module has only local places, $P_\Phi \equiv P_{L\Phi}$.
- $\forall p \in P_{L\Phi}$,
  - -- $\bullet p \in (T_\Phi \cup \emptyset)$. (input transitions of local places are either the transitions of the module or none)
  - -- $p\bullet \in (T_\Phi \cup \emptyset)$. (output transitions of local places are either the transitions of the module or none) This means, local places cannot have direct connections with external transitions.
- $\forall t \in T_{L\Phi}$,
  - -- $\bullet t \in (P_{L\Phi} \cup \emptyset)$. (input places of local transitions are either the local places or none (cold start))
  - -- $t\bullet \in (P_{L\Phi} \cup \emptyset)$. (output places of local transitions either the local places or none (sink))
- $\forall t \in T_{IP\Phi}$
  - -- $\bullet t \in (P_{L\Phi} \cup P_{IM} \cup \emptyset)$. (input places of input ports can be local places or places in inter-modular connectors or can be even an empty set)
  - -- $t\bullet \in (P_{L\Phi} \cup \emptyset)$. (output places of input ports can only be local places, or empty set)
- $\forall t \in T_{OP\Phi}$
  - -- $\bullet t \in (P_{L\Phi} \cup \emptyset)$. (input places of output ports can be local places or an empty set)
  - -- $t\bullet \in (P_{L\Phi} \cup P_{IM} \cup \emptyset)$. (output places of output ports can be local places or places in inter-modular connectors or empty set.
- $A_\Phi \subseteq (P_L \times T_\Phi) \cup (T_\Phi \times P_L)$: where $a_{ij} \in A_\Phi$ is known as the internal arcs of the module.
- $M_{\Phi 0} = [M(p_L)]$ is the initial markings in the local places. □

C. FORMAL DEFINITION OF INTER-MODULAR CONNECTOR

Definition 3. (Inter-Modular Connector (IMC): is defined as a four-tuple [28]:

$$\Psi = (P_\Psi, T_\Psi, A_\Psi, M_{\Psi 0})$$

where,

- $P_\Psi \subseteq P$: P_Ψ is the set of places in the IMC (known as the IM-places). $\forall p \in P_\Psi$,

- -- $\bullet p \in (T_{OP} \cup T_{\Psi} \cup \emptyset)$. (input transitions of IM places are either the output ports of modules, IM transitions of this IMC, or none)
- -- $p\bullet \in (T_{IP} \cup T_{\Psi} \cup \emptyset)$. (output transitions of IM places are either the input ports of modules, IM transitions of this IMC, or none)

  This means, IM places cannot have direct connections with local transitions, or IM transitions of other IMCs.
- $\forall p \in P_{\Psi}, \quad \forall i \;\; p \notin P_{\Phi_i}$ (an IM-place cannot be a local place of any Petri module).
- $T_{\Psi} \subseteq T$: T_{Ψ} is the transitions of the IMC (known as the IM-transitions). $\forall t \in T_{\Phi}$,
  - -- $\bullet t \in (P_{\Psi} \cup \emptyset)$. (input places of IM-transitions are either the IM-places of this IMC, or none (cold start))
  - -- $t\bullet \in (P_{\Psi} \cup \emptyset)$. (output places of IM-transitions either the IM-places of this IMC, or none (sink))
- $\forall t \in T_{\Psi}, \quad \forall i \;\; t \notin T_{\Phi_i}$ (an IM-transition cannot be a transition of any Petri module).
- $A_{\Psi} \subseteq (P_{\Psi} \times (T_{\Psi} \cup T_{IP})) \cup ((T_{\Psi} \cup T_{OP}) \times P_{\Psi})$: where $a_{ij} \in A_{\Psi}$ is known as the connecting arcs of the IMC.
- $M_{\Psi 0} = [M(p_{\Psi})]$ is the initial markings in the IM-places.

$\square$

D. FORMAL DEFINITION OF MODULAR PETRI NET

Definition 4 (Modular Petri Net): is defined as a two-tuple [28]:

$$MPN = (\mathbb{M}, \mathcal{C})$$

where,

- $\mathbb{M} = \sum_{i=0}^{m} \Phi_i$ (zero or more Petri Modules)
- $\mathcal{C} = \sum_{j=0}^{n} \Psi_j$ (zero or more Inter-Modular Connectors)

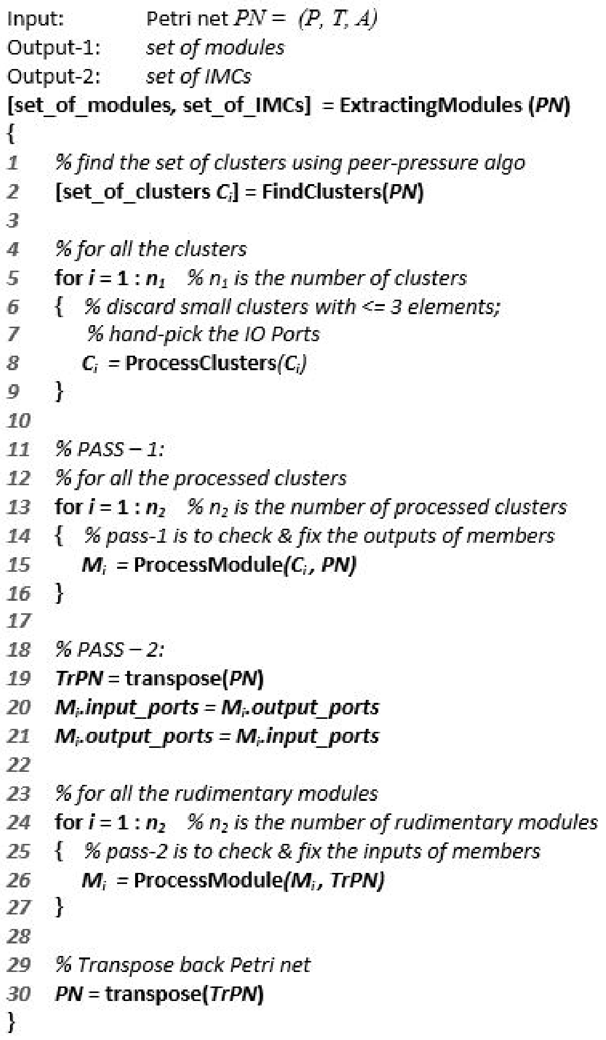
$\square$

V. PETRI MODULES: RULES OF INTERFACING

This section focuses on the module interface (the borders or the input and output ports). For extracting a module from a large Petri net, we can summarize the border rules as follows:

- Inputs to a module: all the inputs to a module can only pass through the input ports of the module. An external element (outsider of a module) is not allowed to input directly to a local member of a module or an output port.
- Outputs from a module: all the outputs from a module can only pass through the output ports of the module. It is not allowed for a local member or an input port to directly output to an external element.
- Internal arcs (connections) of a module: As members of a module, a module consists of three types of transitions (input port, local transition, and output port), and local places. It is allowed that a transition that is a member of a module can be input and out to any local places, and vice versa.

With the rules given above, an algorithm is formulated for extracting the modules and is shown in Fig.2.



```
Input:        Petri net $PN = (P, T, A)$
Output-1:     set of modules
Output-2:     set of IMCs
[set_of_modules, set_of_IMCs] = ExtractingModules (PN)
{
1    % find the set of clusters using peer-pressure algo
2    [set_of_clusters $C_i$] = FindClusters(PN)
3
4    % for all the clusters
5    for $i = 1 : n_1$   % $n_1$ is the number of clusters
6    {  % discard small clusters with <= 3 elements;
7       % hand-pick the IO Ports
8       $C_i$ = ProcessClusters($C_i$)
9    }
10
11   % PASS – 1:
12   % for all the processed clusters
13   for $i = 1 : n_2$   % $n_2$ is the number of processed clusters
14   {  % pass-1 is to check & fix the outputs of members
15      $M_i$ = ProcessModule($C_i$, PN)
16   }
17
18   % PASS – 2:
19   TrPN = transpose(PN)
20   $M_i$.input_ports = $M_i$.output_ports
21   $M_i$.output_ports = $M_i$.input_ports
22
23   % for all the rudimentary modules
24   for $i = 1 : n_2$   % $n_2$ is the number of rudimentary modules
25   {  % pass-2 is to check & fix the inputs of members
26      $M_i$ = ProcessModule($M_i$, TrPN)
27   }
28
29   % Transpose back Petri net
30   PN = transpose(TrPN)
}
```

FIGURE 2. The algorithm for extracting Petri modules from a large Petri net: "FindClusters" detects clusters in the model and "ProcessCluster" (manually done by the modeller) evaluates and pre-processes the clusters into rudimentary modules. "ProcessModules" completes the module formation. During Pass-1, "ProcessModules" fixes the violations in output connections of the members of a module. And during Pass-2, input connections of the members of the module are fixed.

VI. EXTENDED PEER-PRESSURE ALGORITHM

In the extraction algorithm given in the next section, there is a function called "FindClusters" on Line-02 (see Fig.2). This function is to detect the pivotal elements of a large Petri net automatically. This function uses "the extended peer-pressure algorithm" that is presented in the author's earlier work [30]. Hence, in this section, a concise introduction is given to the extended peer-pressure algorithm; for more details, the interested reader is referred to [30].

The extended peer-pressure algorithm is a clustering algorithm that is for finding clusters in a large network. Using this extended peer-pressure algorithm, the user can choose the number of resulting clusters by fine-tuning a parameter known as the "self-loop strength". The extended peer-pressure algorithm not only dissects the network into the expected number of clusters but also the pivotal elements (the important elements) of each cluster will be revealed. In other words, among the internal elements of each cluster, will be one or more pivotal elements (the main elements) and the others will be the secondary elements. Hence, in the extraction algorithm presented in the next section, the function "FindClusters" uses the extended peer-pressure algorithm for automatic detection of the pivotal elements in each cluster.

VII. THE ALGORITHM FOR MODULE EXTRACTION

The algorithm for module extraction (see Fig.2) is a two-pass algorithm:

- In Pass-1: a rudimentary module is formed, by checking and fixing **the outputs** of all the members of the module.
- In Pass-2: all **the inputs** of the members of the module are checked and fixed. For this purpose, the transpose of the Petri Net is used.

In detail, the algorithm consists of the following steps:

- (Line 02) **Finding the clusters**: The extended peer-pressure algorithm is for automatically detecting the main (pivotal) elements of large Petri net; after finding the pivotal elements, clusters (primitive modules) can be extracted surrounding the pivotal elements. The author's earlier work [30] describes automatic detection of the pivotal elements, which is the basis for the function "FindClusters" (line-02).
- Lines 05-09: **Processing the clusters**: Processing the clusters is a **manual process as it has to be done by the modeller**. The modeller performs three specific tasks (again, manually):
  1) Judging whether a cluster is suitable to become a module, due to its small size.
  2) Making sure that an active cycle of operations does not accidentally fall into a set of modules.
  3) Selecting (hand-picking) the input ports and output ports of the modules.

  Task-1: Judging whether a cluster is suitable to become a module: Some of the clusters (again, primitive modules) found by "FindClusters" may possess pivotal elements numbering less than or equal to three. A modeller may appraise that these smaller clusters are not suitable for functioning as modules (e.g., a large model decomposed into too many smaller modules will cause a lot of inter-modular connections). In this case, the elements in these smaller clusters can become IMCs for other modules, or become part of other clusters.

  Task-2: Making sure that an active cycle of operations does not accidentally fall into a set of modules: An active cycle (where a cycle involving a number of elements are repeatedly executed) can be broken into several modules during the modularization, due to the inexperience or oversight of the modeller. Decomposing an active cycle into different modules can cause unnecessarily high inter-modular communication, and high simulation time. Hence, the modeller is supposed to analyze and understand the cycles in the large Petri net, before starting to decompose it. In other words, it could be a good idea to perform a top-down approach. Starting with the large Petri net, analyze it for the cycles, and then start modularizing, while making sure that the active cycles are not dissected and distributed into different modules.

  Task-3: Selecting (hand-picking) the input ports and output ports of the modules: Once a cluster is selected for formation of a module, then both the input ports and the output ports are also determined by the modeller. The following rule of thumb can be used during the selection. The set of transitions that accepts most of the inputs into the cluster becomes the input ports. Similarly, a set of transitions that outwardly connects the cluster to the rest of the model becomes the output ports.

  After processing (the above mentioned three tasks), the clusters become rudimentary modules, possessing the input ports, the output ports, and the internal elements (local transitions and local places). However, the input connections and output connections of the members of the module may violate the rules of interfacing discussed in section-V. These violations are fixed in the two passes.
- (Lines 13-16) **Pass-1**: the function "ProcessModules" uses an modified "Depth-first-search" traversal. The traversal starts with the input ports, one at a time. While traversing, from an input port or from a local member, if there is an output to an outside node, then this is considered as a violation, and it is fixed. When the traversal hits an output port, it will not continue with outputs to any outside node. However, it will continue if an output port has an output to a local place. The function terminates after checking and fixing all the outputs of all the members of the module (the input ports, the output ports, and the internal elements such as local transitions and local places).
- (Lines 19-27) **Pass-2**: In Pass-1, all the outputs of the members of the module are checked and fixed. Pass-2 is for checking and fixing the inputs of the members, using the same function "ProcessModules". However, to check the inputs, the Petri net is transposed, and the transposed Petri net is input to the function. Also, the input ports and the output ports are exchanged. This means the output ports are fed into the function "ProcessModules" as input ports, and similarly input ports as output ports.

  After pass-2, all the members of the module is thoroughly checked, verifying whether their input and output connections obey the rules of interfacing.
- (Line 30) Completion: Once the Petri net is transposed back to its normal form, there will be a number of Petri modules, clearly identified and encapsulated by the input and output ports. All the transitions and places that reside outside of these modules will become the IMC.

During the two-passes, the algorithm uses fixes for correcting the connections (arcs) that violate the rules of interfacing.

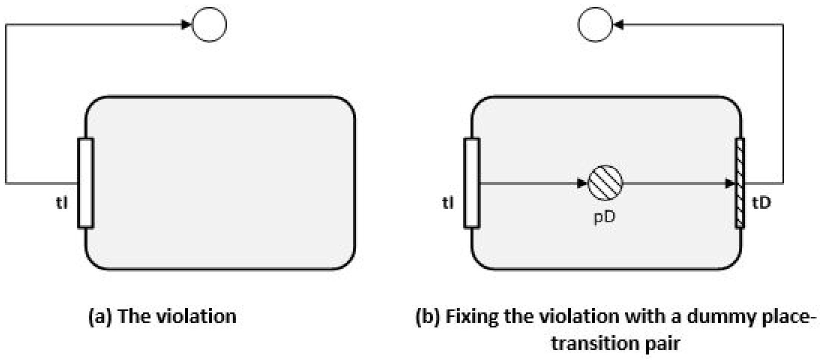


FIGURE 3. Violations in the output connections from an input port.

The process of fixing is explained in the following two sub-sections, section-VII-B and VII-C.

A. RUNNING-TIME OF THE ALGORITHM

As shown in Fig.2, the algorithm mainly consists of three functions:

- FindCluster: it is stated [30] that the algorithm takes $\mathcal{O}(r \cdot V^3)$ time, where r is the number of iterations and V is the number of elements (places and transitions) in the Petri net.
- ProcessClusters: this function cannot be executed automatically, as this function needs the experience and insight of the modeller. The steps described in this function are to be followed by a modeller, e.g., hand-picking input and output ports, moving elements in and out of clusters.
- ProcessModules (in Pass-1 and Pass-2): ProcessModules is a modified depth-first-search (DFS) algorithm. (the running time of DFS is $\mathcal{O}(V + E)$, where V and E are the numbers of vertices and edges, respectively). The DFS will be repeated a number of times, once for each input and output port. Assuming the total number of input and output ports $|T_I + T_O| \ll V$, the running time of the function ProcessModules becomes $\mathcal{O}(V+E)$. Since Petri nets are generally sparsely connected [31], $V \approx E$, we can conclude that the running time of the function ProcessModules in terms of V is $\mathcal{O}(V)$.

Hence, neglecting the function "ProcessClusters", the running time of the algorithm for extraction of modules become $\mathcal{O}(r \cdot V^3) + \mathcal{O}(V) = \mathcal{O}(r \cdot V^3)$.

B. FIXING VIOLATIONS IN PASS-1

In Pass-1, if an **output arc** from an input port or a local member hits a node that is outside the module, it is considered as a violation. Fig.3(a) shows such a situation. When we start the traversal with an input port, there may be an arc from the input port to an outside node. This is not acceptable, as input ports (and local transitions) are only supposed to feed tokens to local places inside a module. In this case, this violation can be fixed by introducing a"dummy" place-transition pair, as shown in Fig.3(b). The dummy transition tD now acts as an output port too.

A similar situation is described in Fig.4(a). In this case, a local transition tL has an output arc to an outside place. This can also be corrected with a dummy place-transition pair, as shown in Fig.4(b). Also, there could be another possibility as shown in Fig.4(c); in this case, the local transition tL can become an output port tO, eliminating the necessity of injecting a dummy place-transition.

Fig.5(a) looks into the problem of a local place having an output to a transition that is residing outside the module. In this case, the solution is to inject a dummy transition-place pair; the dummy transition tD becomes an output port whereas the dummy place pD is placed outside the module meaning it becomes a part of an IMC.

In figures Fig.3-7, 10-14, 17 and 19, the transitions are drawn in different sizes and shapes to indicate that there are different types of transitions. Firstly, the extraction algorithm creates two types of transitions:

1) The process transitions that are part of the original Petri net.
2) The injected "dummy" transitions to fix the violations.

Naturally, to differentiate these two types, these transitions are drawn differently (dummy transitions are represented by thinner rectangles, whereas process transitions by thicker rectangles). Secondly, the Petri modules have input and output ports as well as internal transitions. The transitions that are input and output ports are represented by rectangles with thicker (bolder) lines and the internal transitions with thinner lines.

C. FIXING VIOLATIONS IN PASS-2

The previous section-VII-B looked into the output arcs of the members of a module in Pass-1. In Pass-2, the **input arcs** of the members are checked. Fig.6 and Fig.7 show the three types of violations that can happen, and the fixings are also shown.

Fig.6(a) shows an output port receiving an input from an outside place. A dummy transition-place pair can fix this violation as shown in 6(b); the dummy transition tD becomes an input port too. In Fig.6(c), a local transition receives an

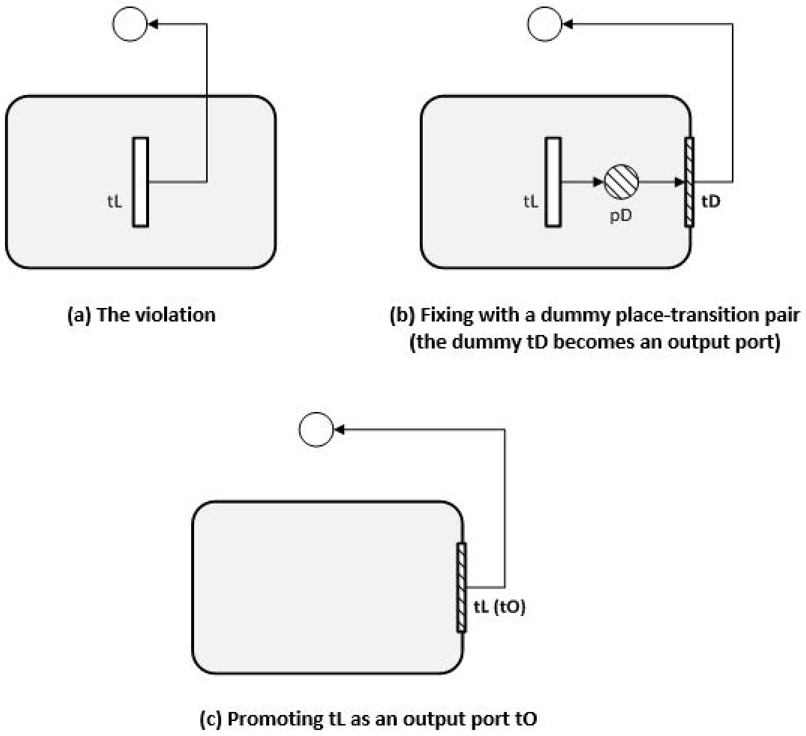

tL
tL
pD
tD
(a) The violation
(b) Fixing with a dummy place-transition pair (the dummy tD becomes an output port)
tL (tO)
(c) Promoting tL as an output port tO

FIGURE 4. Violations in the output connections from a local transition.

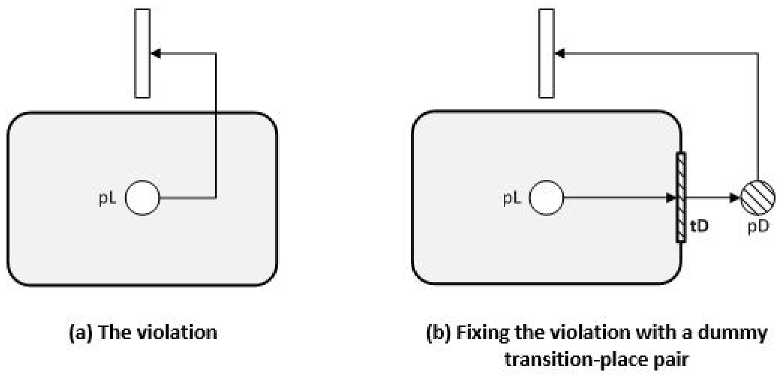


FIGURE 5. Violations in the input connections from a local place.

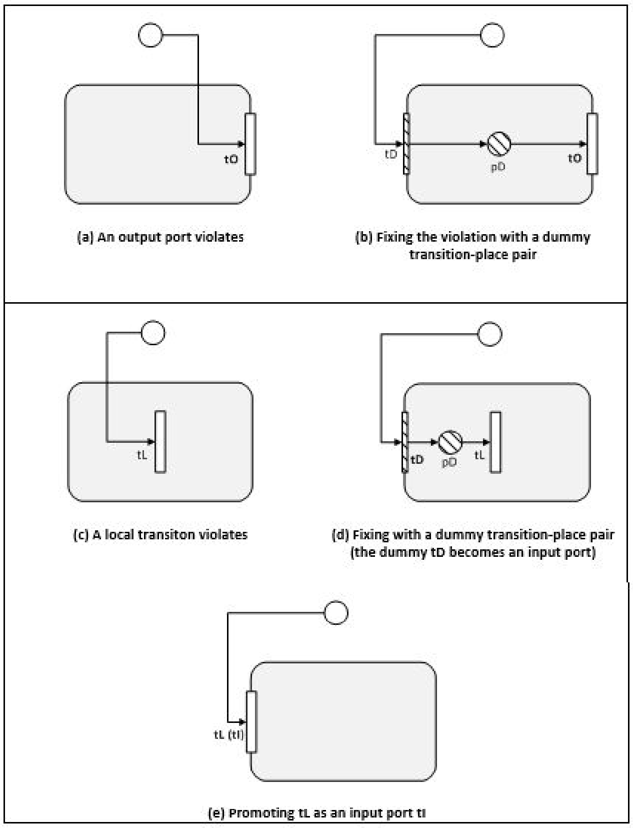
input from an outside place, and the solution with a dummy transition-place pair is shown in 6(d). Another solution is to make the local transition as an input port, as shown in 6(e).

Fig.7 shows the final fixing. In this case, a local place violates the rules by receiving direct inputs from an outside transition (Fig.7(a)). The solution is shown in 7(b). Here again, the dummy place pD becomes an IMC, whereas the dummy transition tD becomes an input port.

VIII. APPLICATION OF THE MODULE EXTRACTION ALGORITHM

A simple Flexible Manufacturing System (FMS) shown in Fig.8 is for making only one type of product. The FMS example is taken from [27] and [32]. The operational specifications of the FMS are as follows:

- The input raw material of type 1 arrives on the conveyor belt C1. Robot R1 picks up the raw material of type 1 and places into the machine M1. Similarly, robot R2 picks up the raw material of type 2 from conveyor belt C2 and places it into the machine M2.
- Machine M1 makes the part P1, and M2 makes the part P2. When the parts are made by the machines M1 and M2, these parts are placed on the assembly station (AS) by the robots R1 and R2, respectively.
- Assembly station AS is used to join the two parts P1 and P2. Robot R2 does the part assembly at AS.



tO
(a) An output port violates
tD
pD
tO
(b) Fixing the violation with a dummy transition-place pair
tL
(c) A local transiton violates
tD
pD
tL
(d) Fixing with a dummy transition-place pair (the dummy tD becomes an input port)
tL (tI)
(e) Promoting tL as an input port tI

FIGURE 6. Violations in the input connections of members of a module.

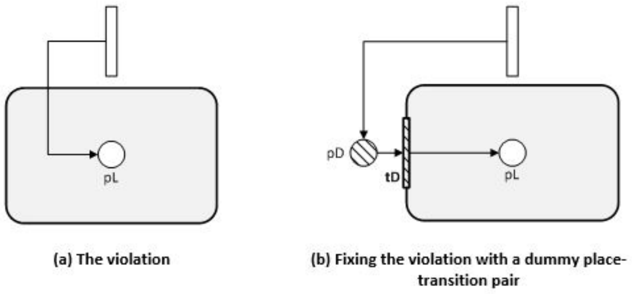


FIGURE 7. Violations in input connections of local place.

- Robot R3 picks the product from the assembly station and places it on the painting station PS.
- Robot R4 performs the surface polishing and painting.
- Once the painting is completed, robot R3 picks up the completed product from the painting station PS and packs it into the cartridge OB.

A. THE PETRI NET MODEL

In the Petri net model shown in Fig.9, the following activities represent the FMS operations ('t' stands for transition):

- tC1: conveyor belt C1 brings the input material of type 1 into the FMS.
- tC2: conveyor belt C2 brings the input material of type 2 into the FMS.
- tC1M1: robot R1 moves raw material from conveyor belt C1 and places it on M1.
- tC2M2: robot R2 moves raw material from conveyor belt C2 and places it on M2.
- tM1: machining of part 1 at machine M1.
- tM2: machining of part 2 at machine M2.
- tM1AS: robot R1 moves part 1 from M1 to the Assembly Station AS.
- tM2AS: robot R2 moves part 2 from M2 to the Assembly Station AS.
- tAS: robot R2 assembles parts P1 and P2 together at the assembly station AS.
- tAP: robot R3 picks the product from the assembly station and places on the painting station PS.
- tPS: robot R4 performs surface polishing and painting on the product.
- tPCK: when the painting job is finished, R3 packs the product into the output cartridge.

The Petri net model of the FMS is shown in Fig.9 is obtained by connecting the activities listed above, one after the other. In the Petri net model, the input buffers IB1 and IB2 are represented by the places pIB1 and pIB2, and the output buffer by the place pOB. It is an assumption that the three places have no capacity restraints.

In Fig.9, transitions tM1, tM2, tAS, and tPS are shown in shaded color, as these transitions represent the pivotal elements (the most important machines) of the Petri net model. Also, the numbers written inside the transitions are the firing times (time took by the relevant activity), given in some time units.

B. STATE SPACE OF THE PETRI NET MODEL

In Fig.9, the initial state is assumed as one token each in the following places. Input places pIB1, pIB2, pR1 to pR4 (showing the availability of the robots R1 to R4), pC1, pC2 (showing the availability of the conveyor belts), and po1AS, po2AS. The state space of the FMS is shown in Fig.10.

It is important to note that there are 45 states in the state space shown in Fig.10. **The state space shown in Fig.10 is incomprehensible due to overlapping of the 45 states**. Out of the 45 states, only 29 are unique states (the 16 duplicate states are highlighted in yellow color). However, this state space is only for the initial markings of one token each in pIB1 and pIB2. If the initial markings on the pIB1 and pIB2 are increased (while keeping the same one initial token in pR1 to pR4, pC1, pC2, and po1AS, po2AS) then the number states in the state space increases linearly.

Table 1 presents the number of states in the state space when the initial token in pIB1 and pIB2 are increased.

Table 1 shows the enormous size of the state space (e.g., 184 states when there are just two tokens each in the input buffers pIB1 and pIB2).

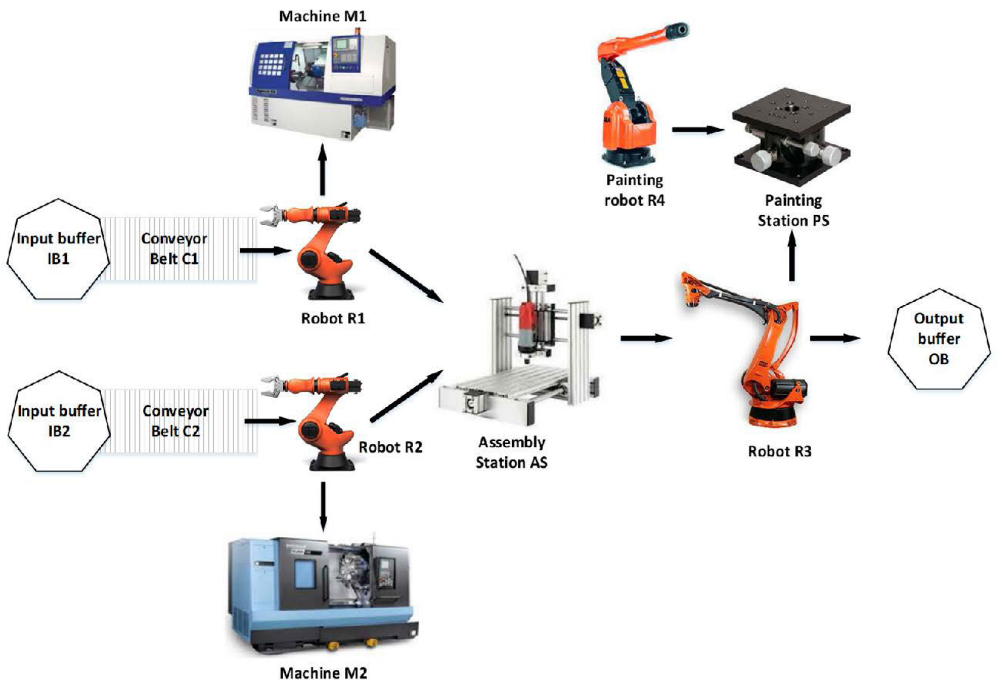


FIGURE 8. A simple Flexible Manufacturing System (FMS); adapted from [27], [32].

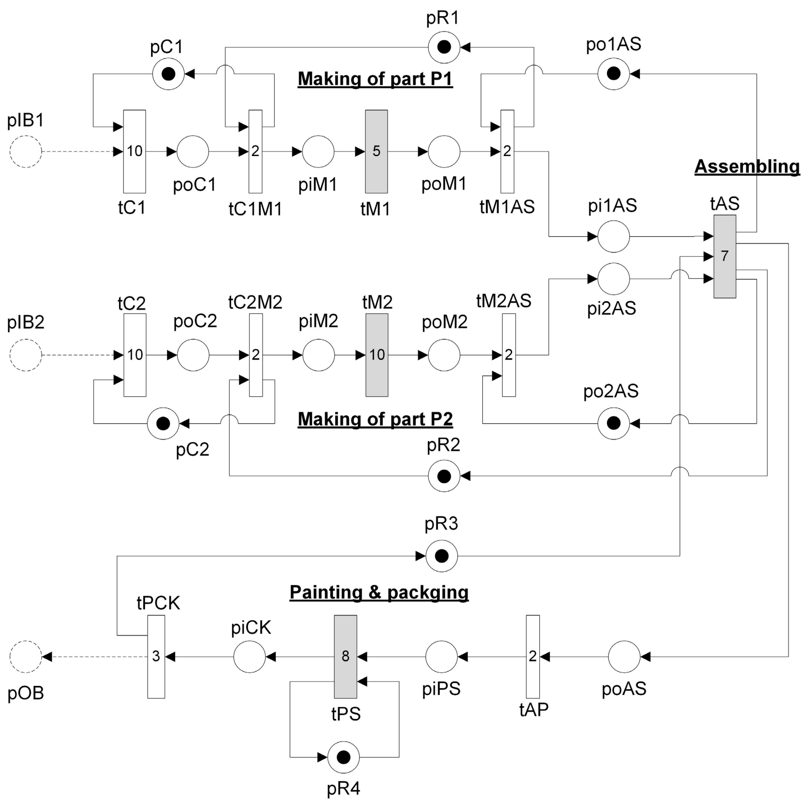


FIGURE 9. The Petri Net model of a flexible manufacturing system.

C. IDENTIFYING THE PRIMITIVE MODULES

Let us apply the extended peer-pressure algorithm to detect the clusters (primitive modules) in the Petri net model shown in Fig.9. The following nine clusters were detected by the function "FindCluster" which is based on the extended peer-pressure algorithm (see section-VI).:

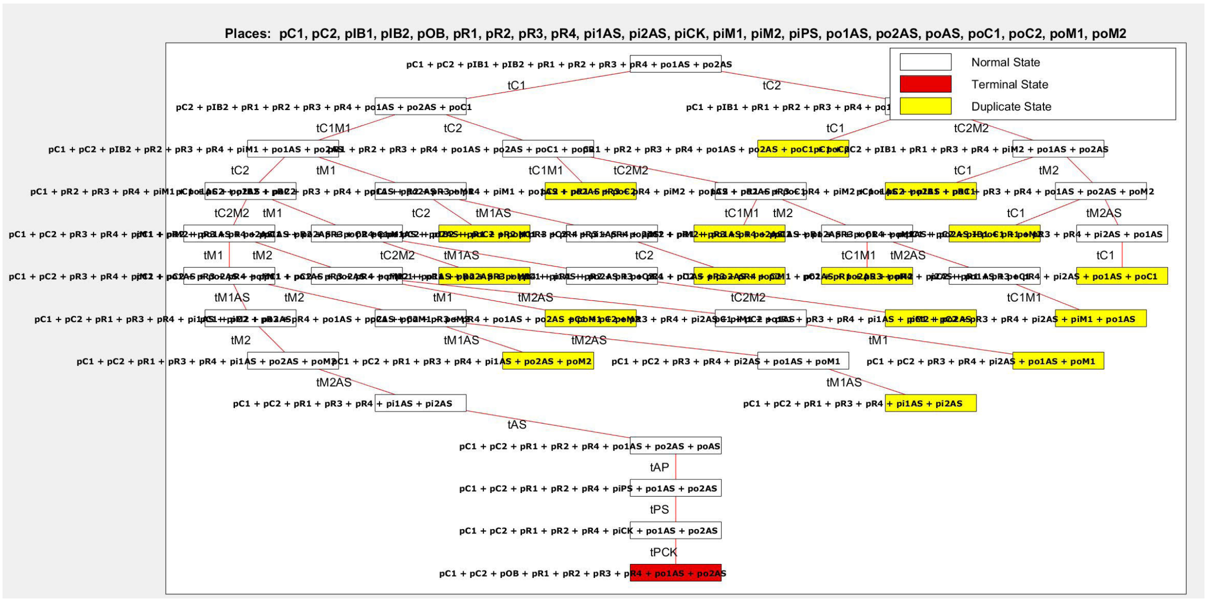

Places: pC1, pC2, pIB1, pIB2, pOB, pR1, pR2, pR3, pR4, pi1AS, pi2AS, piCK, piM1, piM2, piPS, po1AS, po2AS, poAS, poC1, poC2, poM1, poM2
Normal State
Terminal State
Duplicate State
pC1 + pC2 + pIB1 + pIB2 + pR1 + pR2 + pR3 + pR4 + po1AS + po2AS
tC1
tC2
tAS
tAP
tPS
tPCK
pC1 + pC2 + pR1 + pR2 + pR4 + po1AS + po2AS + poAS
pC1 + pC2 + pR1 + pR2 + pR4 + piPS + po1AS + po2AS
pC1 + pC2 + pR1 + pR2 + pR4 + piCK + po1AS + po2AS
pC1 + pC2 + pOB + pR1 + pR2 + pR3 + pR4 + po1AS + po2AS

FIGURE 10. The state space of the flexible manufacturing system. *The state space is incomprehensible due to overlapping of the 45 states.*

TABLE 1. Number of states in the state space versus number of initial token in the input buffers.

Number of initial tokens in pIB1 and pIB2	Number of unique states in the state space
(1, 1)	29
(2, 2)	184
(3, 3)	520
(4, 4)	904
(5, 5)	1288
(6, 6)	1672
(7, 7)	2056
(8, 8)	2440
(9, 9)	2812
(10, 10)	3208

* Cluster-1: tAS, tPCK, tPS
* Cluster-2: tC1,tC1M1,tM1,pC1,pIB1,pOC1,piM1,poM1
* Cluster-3: tC2,tC2M2,tM2,pC2,pIB2,pOC2,piM2,poM2
* Cluster-4: pOB
* Cluster-5: pR2
* Cluster-6: pR3, pR4, piCK
* Cluster-7: tM1AS, pR1, pi1AS, po1AS
* Cluster-8: tM2AS, pi2AS, po2AS
* Cluster-9: tAP, piPS, poAS

Some of these clusters contains only places (clusters 4-6). Thus, these places become IMCs. The other five clusters are destined to become modules. Table 2 shows the clusters and the hand-picked input and output ports of the modules. Selecting (hand-picking) the input ports and output ports of a module is described as task-3 of the modeller during the execution of the function "ProcessClusters" (in section-VII). The following rule of thumb can be used during the selection: The set of transitions that accepts most of the inputs into the cluster becomes the input ports; a set of transitions that passes most of the outputs of the cluster to the rest of the system becomes the output ports.

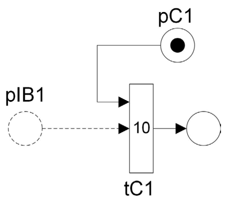


FIGURE 11. Conveyor Belt-1: the segment.

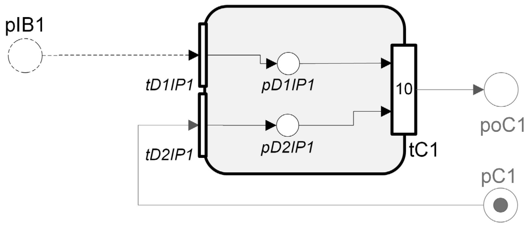


FIGURE 12. Conveyor Belt-1: the module.

The inter-modular components are the elements that do not fall into any of the five modules. The IMCs are the places pC1, pC2, poC1, poC2, pR2, pi1AS, pi2AS, po1AS, po2AS, pR3, poAS, and pOB, and the transition tAS.

Let us go through these five modules in the following subsections.

D. INPUT MODULES

Two input modules deal with the arrival of input (raw) material into the production line and transporting the material on the conveyor belts. The two input modules are "Conveyor Belt-1" and "Conveyor Belt-2". Since these two modules are similar, let us focus on "Conveyor Belt-1".

TABLE 2. **The clusters for modularization.**

Module	Pivots	Input Ports	Output Ports
Conveyor Belt-1	pIB1, **tC1**	-	**tC1**
Conveyor Belt-2	pIB2, **tC2**	-	**tC2**
Machining-1	**tC1M1**, piM1, pR1, tM1, poM1, **tM1AS**	**tC1M1**	**tM1AS**
Machining-2	**tC2M2**, piM2, tM2, poM2, **tM2AS**	**tC2M2**	**tM2AS**
Product Finishing	**tAP**, piPS, tPS, pR4, piCK, **tPCK**	**tAP**	**tPCK**

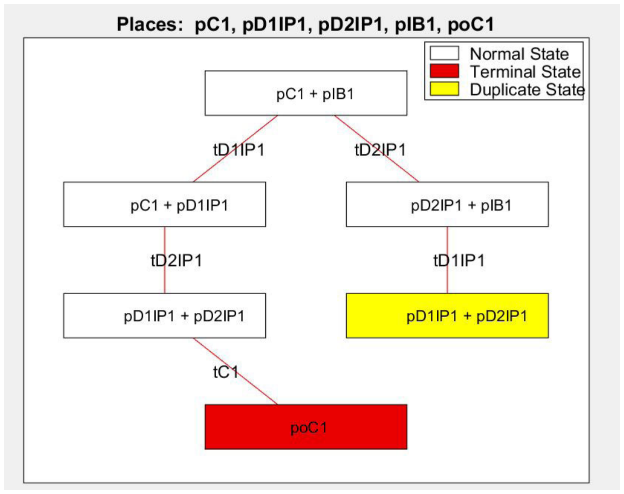


FIGURE 13. **The state space of the module "Conveyor Belt-1".** ***The state space consists of five unique states.***

Fig.11 shows the segment of Petri net that represents the arrival of input material-1. Fig.12 shows the modular version. pIB1 that represents the continuous arrival input material is kept outside the module. In the segment (Fig.11), pC1 and pIB1 are the two inputs to tC1. In the module (Fig.12), since tC1 functions as an output port, it cannot receive any input from the external places. Thus, in Pass-2, a dummy transition-place pairs (tD1IP1 & pD1IP1 and tD2IP1 & pD2IP1) are introduced so that pC1 and pIB1 can still input tokens into tC1 via tD1IP1 and tD2IP1.

Fig.13 shows the state space of the module "Conveyor Belt-1", for the initial tokens of one token each in pC1 and pIB1. The state space of this module consists of five unique states.

E. MACHINING MODULES

There are two machining modules,"Machining-1" and "Machining-2". Since these two modules are similar too, let us focus on the module "Machining-1" only.

Fig.14 shows the segment in which tC1M1 outputs to an outside element pC1. During the Pass-1, this violation is fixed with a dummy place-transition pair pD2M1-tD2M1. Also, the output port tM1AS receives tokens from the outside place po1AS, which will be fixed in Pass-2. Fig.15 shows the resulting module.

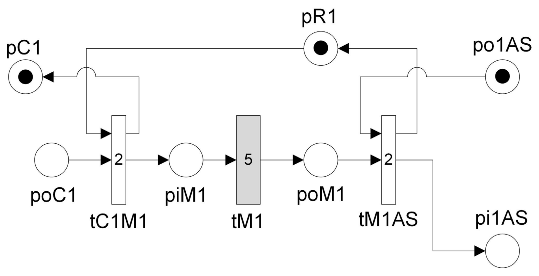


FIGURE 14. **Machining-1: the segment.**

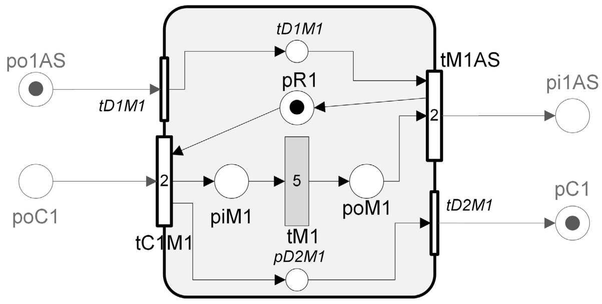


FIGURE 15. **Machining-1: the module.**

Fig.16 shows the state space of the module "Machining-1", for the initial tokens of one token each in pC1 and pIB1. The state space of this module consists of 12 unique states (and seven duplicate states).

F. FINISHING AND OUTPUT MODULE

Fig.17 shows the segment "Finishing-1". The connections of the input port tAP, the local members pPS, tPS, pR4, and pPCK, and the output port tPCK do not cause any violation. Thus, during the two passes, there was no need to fix any connections. Fig.18 shows the resulting module.

Fig.19 shows the state space of the module "Finishing-1", for the initial tokens of one token each in poAS and pR4. The state space of this module consists of four unique states.

G. INTER-MODULAR CONNECTOR

All those elements that are not a member of any modules become IMC. The following places that function as buffers between the modules are the IMC: pIB1, pIB2, pC1, pC2, poC1, poC2, pR2, pi1AS, pi2AS, po1AS, po2AS, pR3, poAS, and pOB. Note that pIB1, pIB2, and pOB in a way special IMCs, as these are the sources and the sink of the model. Also, tAS becomes the only IMC transition.

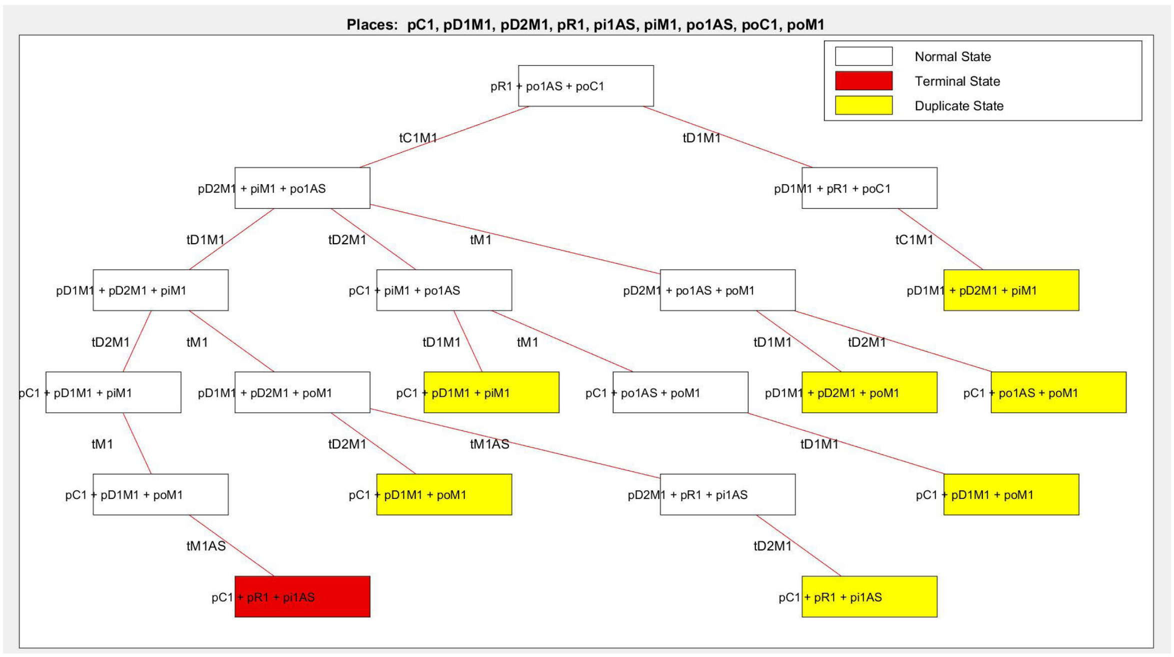


FIGURE 16. The state space of the module "Machining-1". *The state space consists of 12 unique states.*

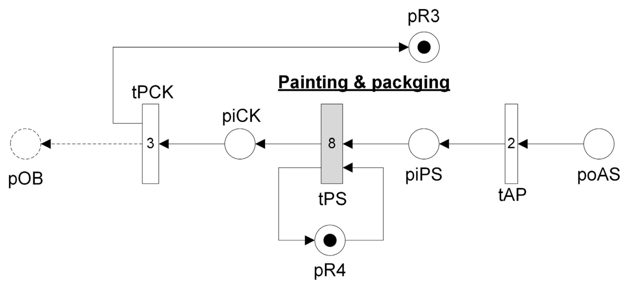


FIGURE 17. Finishing: the segment.

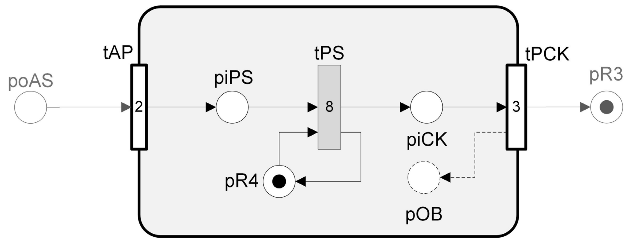


FIGURE 18. Finishing: the module.

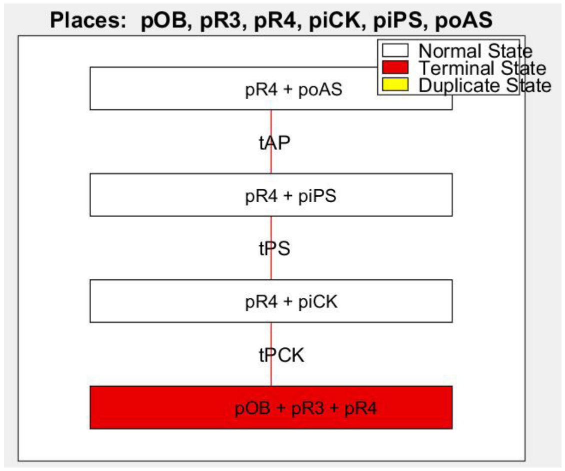


FIGURE 19. The State Space of the module "Finishing-1". The state space consists of four unique states.

H. THE COMPLETE MODULAR PETRI NET

Fig.20 shows the modular Petri net that is composed of the five modules.

IX. ANALYZING THE FIXES

The algorithm for module extraction applies fixes in the formation of modules; in Pass-1 and Pass-2, the fixes are injection of dummy place-transition pairs. In this section, we study the impact of the fixes. By studying the structural properties place-invariant and transition-invariant, this section shows that the injection of dummy places do not cause any changes. There are some algorithms available for computing place-invariants and transition-invariants, e.g., [33], [34].

At first, we start with a general analysis of the impact of the dummy elements.

A. GENERAL ANALYSIS OF THE FIXES

1) DUMMY PLACES

Dummy places are virtual places as these do not represent any passive elements in the real-life system (e.g., pD does not represent a real buffer). Thus, because of the virtual existence, the dummy places do not malfunction anytime.

In a fix, [real transition tX → dummy place pD → dummy transition tD], whenever pD receive a token from tX, the token will be immediately snatched away by tD. Similarly,

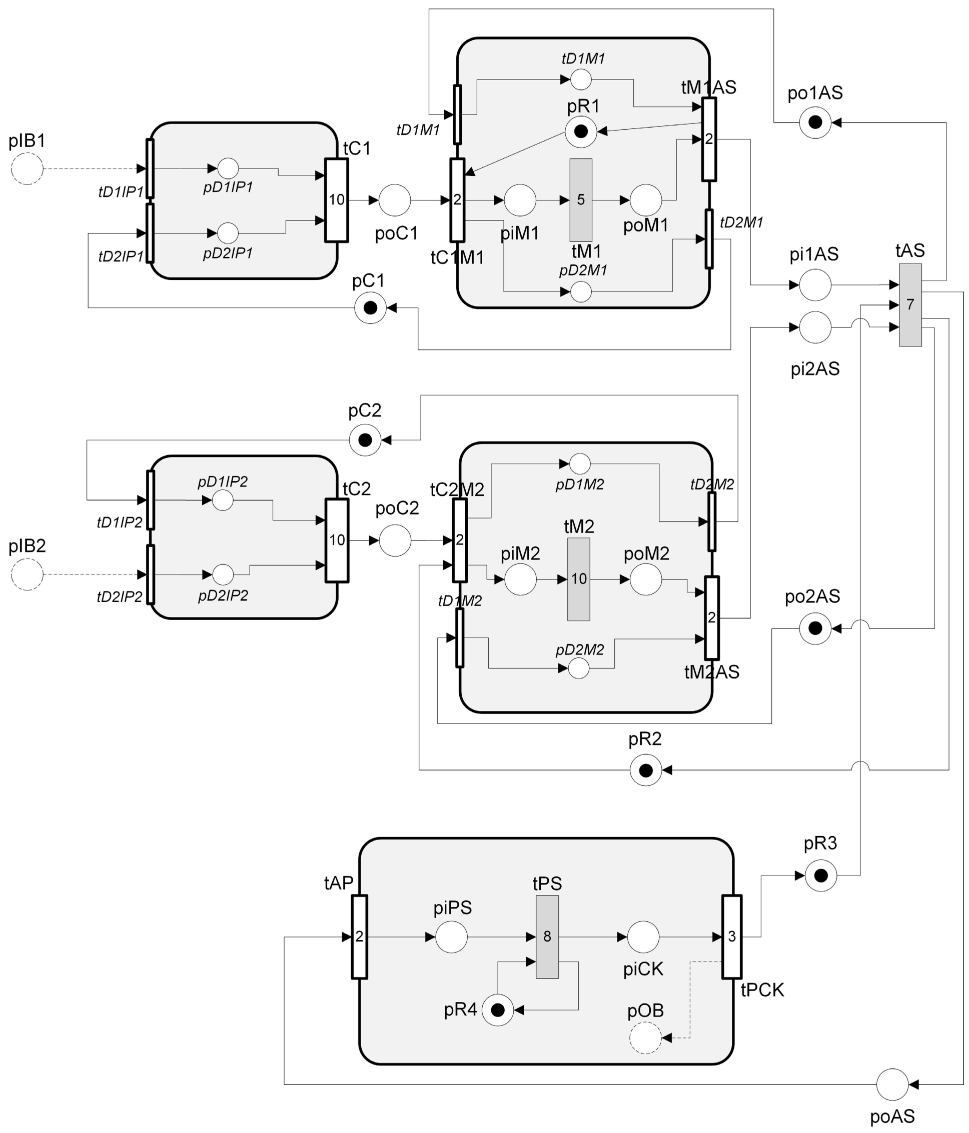


FIGURE 20. Modular Petri Net model of FMS.

in [dummy transition tD $\rightarrow$ dummy place pD $\rightarrow$ real transition tX], whenever pD receive a token from tD, the token will be immediately snatched away by tX. Thus, in both cases, pD holds the token only momentarily. Therefore, in both cases, it can be assumed that always, $m(pD) = 0$.

2) DUMMY TRANSITIONS

Like dummy places, dummy transitions are also virtual as these do not represent any active elements in the real-life system (e.g., tD does not represent a real machine). Thus, because of the virtual existence, the dummy transitions do not malfunction anytime. Also, a dummy transition tD fire immediately.

For untimed Petri net, tD is a primitive transition that do not possess a firing time (in other words, $ft(tD) = 0$). For timed Petri net, all transition must have non-zero firing time (at least, in GPenSIM environment). While $t_i \in T$ takes non-zero

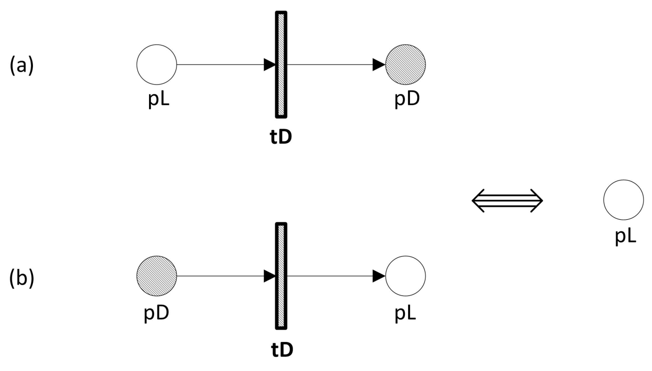


FIGURE 21. Fix-1: Original place pL and a dummy transition-place pair.

firing time, the dummy transitions $tD_i \in TD$ are suppose to fire immediately. Hence, firing time of tD_i is assigned the minimum time interval that is possible, $ft(tD) = \Delta T$. ΔT is the absolute minimum time in GPenSIM realization that is

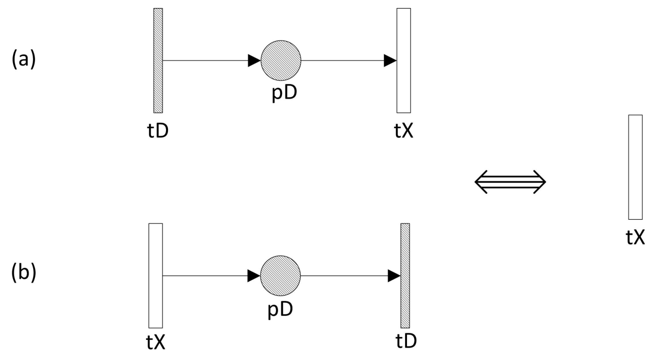


FIGURE 22. Fix-2: Original transition tX and a dummy place-transition pair.

not zero (known as "DELTA_TIME" in GPenSIM reference manual [35]. Since the firing time of tD is negligible when the activities in real-systems are considered, even for timed systems, it can be safely assumed that $ft(tD) = 0$.

Fig.3-7 show six cases of fixes. All these fixes can be grouped into three types:

1) Input or output connection of a local place is fixed, as shown in Fig.21. As shown in Fig.21(a), [original place pL → dummy transition tD → dummy place pD] is equivalent the original place pL. This is because, pD falls outside the module and becomes an IMC (see also Fig.5(b)), thus will not appear in the state space of the module. tD, as it always fires immediately, will make the leaves of the state space. Similarly, [dummy place pD → dummy transition tD → original place pL] (Fig.21(b)) is equivalent the original place pL. In this case, pD again falls outside the modules as an IMC that becomes the input to the module (see also Fig.7(b)).
2) Input or output connection of a transition is fixed, as shown in Fig.22. Again, by the proof shown in [36] ("fusion of series transitions"), [dummy transition tD + dummy place pD + original transition tX] (Fig.22(a)) is equivalent the original transition tX; tX can be a local transition tL or an output port tO. Similarly, [original transition tX → dummy place pD → dummy transition tD] (Fig.22(b)) is equivalent the original transition tX; in this case, tX can be a local transition tL or an input port tI.
3) A local transition is moved to either an input port or an output port. In this case, there is no change in structural or behavioral properties.

B. P-INVARIANTS

Consider the Resource Allocation System (RAS) shown in Fig.23; this example is taken from [10]. Description of the RAS:

- The RAS has three common resources such as Rx, Ry, and Rz.
- The resources Rx, Ry, and Rz have three, two, and one instances (copies), respectively. The number of instances is shown as the initial tokens in Fig.23.
- The RAS is made up of two cyclic processes A and B.
- Process A possesses four tasks, A1 to A4. Process B owns five tasks, B1 to B5.
- A and B use the three resources during different tasks, as shown in Fig.23. For example, task A1 needs two instances of Rx, whereas task B1 needs one instance each of Rx and Rz. Fig.23 also shows that the tasks release the resources after usage.
- Though processes A and B can run in parallel, simultaneous execution of the tasks are not possible since the tasks of A and B requires more than the available resources instances. For example, A1 and B2 cannot run in parallel. As A1 and B2 each need two instances of Rx (A1 directly takes two instances of Rx, whereas B2 takes one instance directly, and one from B1). However, there are only three instances of Rx.

The RAS has the following p-invariants:

1) PI_{pA41} is for preserving the initial tokens in pA41.
2) PI_{pB51} is for preserving the initial tokens in pB51.
3) PI_{R_X} is for preserving the initial tokens in R_X.
4) PI_{R_Y} is for preserving the initial tokens in R_Y.
5) PI_{R_Z} is for preserving the initial tokens in R_Z.

The p-invariants of RAS model:

$$\begin{aligned}
PI_{pA41} &: m(pA12) + m(pA23) + m(pA34) + m(pA41) = 2\\
PI_{pB51} &: m(pB12) + m(pB23) + m(pB34) + m(pB45)\\
&\quad +m(pB51) = 3\\
PI_{R_X} &: m(pR_X) + m(pB12) + 2 \times (m(pA12) + m(pA23)\\
&\quad + m(pA34) + m(pB23) + m(pB34) + m(pB45)) = 3.\\
PI_{R_Y} &: m(pR_Y) + m(pA23) + m(pB45) + 2 \times m(pA34) = 2.\\
PI_{R_Z} &: m(pR_Z) + m(pB12) + m(pB23) = 1.
\end{aligned}$$

Let us remodel the RAS problem as a modular Petri net. The modular RAS is shown in Fig.24.

Let us study the p-invariants of the two modules A and B. While performing the invariant analysis, we need to include the complete instances of the resources (in other words, the initial tokens) in the drivers as shown in Fig.24. The invariants of the module-A:

$$\begin{aligned}
PI_{pA41} &: m(pA12) + m(pA23) + m(pA34) + m(pA41) = 2\\
PI_{R_{X_A}} &: m(pR_{X_A}) + 2 \times (m(pA12) + m(pA23)\\
&\quad +m(pA34)) = n_1\\
PI_{R_{Y_A}} &: m(pR_{Y_A}) + m(pD1) + m(pA23) + m(pD2)\\
&\quad +2 \times m(pA34) = n_2.
\end{aligned}$$

The invariants of the module-B:

$$\begin{aligned}
PI_{pB51} &: m(pB12) + m(pB23) + m(pB34) + m(pB45)\\
&\quad +m(pB51) = 3\\
PI_{R_{X_B}} &: m(pR_{X_B}) + m(pD3) + m(pB12) + 2 \times (m(pB23)\\
&\quad +m(pB34) + m(pB45)) = n_3\\
PI_{R_{Y_B}} &: m(pR_{Y_B}) + m(pD4) + m(pB45) = n_4.\\
PI_{R_Z} &: m(pR_Z) + m(pB12) + m(pB23) = 1.
\end{aligned}$$

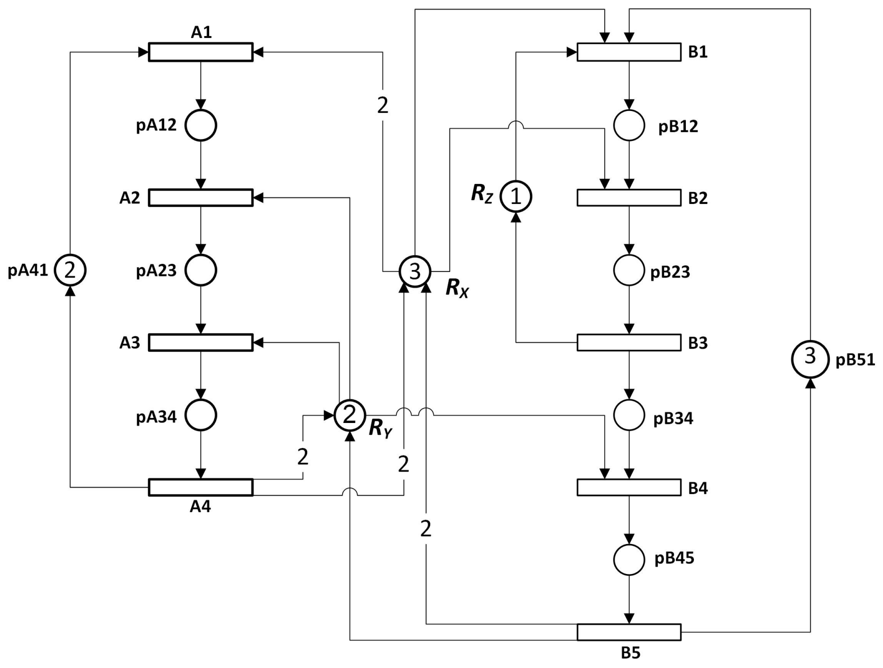


FIGURE 23. Resource Allocation System (RAS) [10].

By setting all $m(pDn) = 0$, we can see that,

- $PI_{R_X} = PI_{R_{X_A}} + PI_{R_{X_B}}$, as $n_1 + n_3 = 3$ (the initial tokens in R_X).
- $PI_{R_Y} = PI_{R_{Y_A}} + PI_{R_{Y_B}}$, as $n_2 + n_4 = 2$ (the initial tokens in R_Y),

Thus, the presence of the dummy places do not disturb the p-invariants.

C. T-INVARIANTS

To study the impact of fixing on t-invariants, let us take a simple Petri net that is shown in Fig.25. In this Petri net, there is one t-invariant: {t1, t2, t3}. If we put a token into p4, then the firings of t1, t2, and t3, will bring back the same state we started (that is one token in p4). Hence, t-invariant {t1, t2, t3} is justified.

Fig.26 shows one way of modularizing the Petri net in Fig.25. In the modular Petri net, transitions t1 and t3, and places p2 and p4 become members of module-A. Module-A is free from any fixing as all the connections of its members (t1, t3, p2, p4) do not violate the rules of interfacing.

The rest of the places in Fig.25, p1 and p2, become the members of module-B. Since p1 cannot have direct input from outside transition, and p3 output to outside transition, these two connections are fixed. The fixing makes use of the dummy transitions tD1 and tD2 as the input and output port of module-B. Also, due to the fixing, dummy places pD1 and pD2 become IMCs.

Studying the modular Petri net reveals that the modular model posses a t-invariant too, which is {t1, t2, t3, tD1, tD2}. For example, if we put one token in p4, and let the transitions t1, tD1, t2, tD2, and t3 to fire in that order, we then will go back to the original state of one token in p4. Since it is already shown in subsection-IX-A "General analysis of the Fixes" that tDi can be neglected in comparison with the real transitions, the t-invariant of the modular Petri net becomes the same as its monolithic (non-modular) version.

X. DISCUSSION

In the example shown above, the non-modular (monolithic) Petri net posses a t-invariant consisting of the transitions t1, t2, and t3. The t-invariant could be a cycle that might frequently occur. In the modular version, the cycle is broken into two groups of transitions, and one group (t1 and t3) become members of one module (module-A) and the other group (t2) a member of the other module (module-B). Technically seen, these two versions, monolithic and modular, are the same and provide the results (e.g., same t-invariants as shown above). However, there is a big difference between these two versions.

If the t-invariant (consisting of the transitions t1, t2, and t3) represent a frequently occurring cycle of a real system, then splitting the cycle into two or more modules is a bad idea. This is because of the cycle that span several modules will pass tokens from module to module that may incur additional communication delays, assuming that the modules are hosted on different computers.

In this paper, a small example (the flexible manufacturing systems, shown in Fig.8) is taken as the case study as we are trying to explain how the definitions for the modular Petri net and the extraction algorithm actually work in practice. The case study should help the readers to visualize the mechanisms behind the algorithm in action. The case study

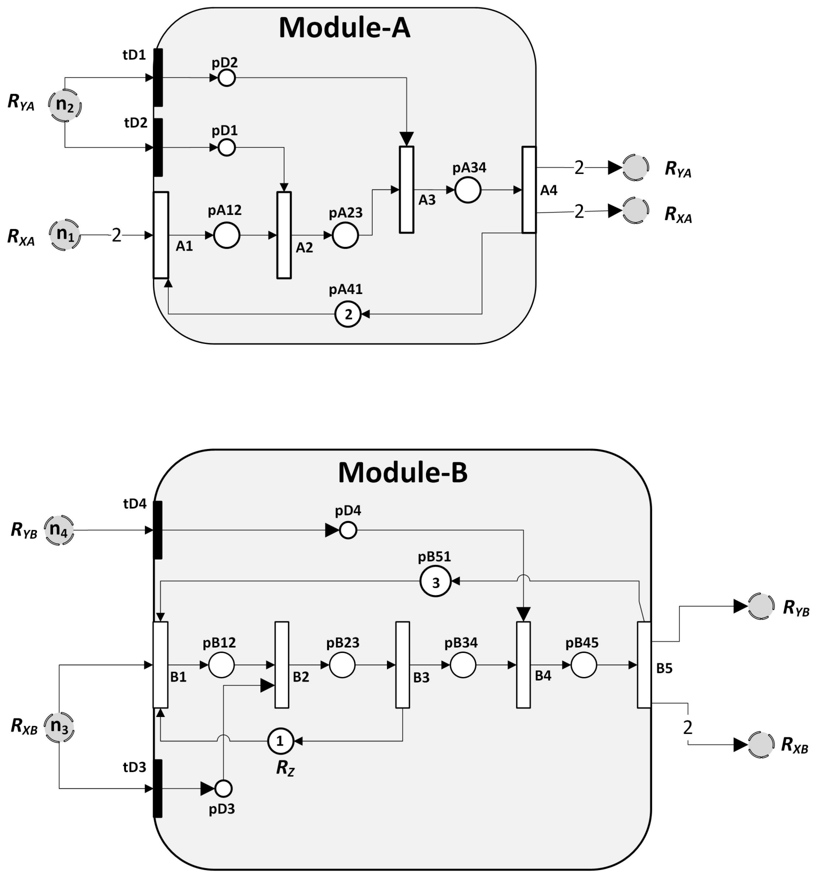


FIGURE 24. Modular Petri net model of RAS.

is purposely chosen to be small so that it will not result in too many modules. Also, due to the smaller size, it will be easy to focus on what we are supposed to focus, namely, the extraction algorithm. Even though the problem (the Petri net model shown in Fig.9) is small, the resulting state space is large (shown in Fig.8). However, for a Petri net model of a real-life discrete-event system, the resulting state space will be huge.

In comparison to the state space of the whole model, the state spaces of the modules are compact (Fig.13, 16, and 19). The modules are smaller in size in terms of the number of places, transitions, and arcs. Hence, the execution time for individual modules also becomes small compared to the execution time of the whole model.

Limitations of the Proposed Methodology: This paper does not present the experimental evaluation of the extraction algorithm. This because the algorithm can not be fully implemented as an executable software as part of the algorithm (lines 05-09, the function "ProcessCluster") has to be executed manually by the modeller. The rest of the algorithm (functions "FindCluster" and the two passes involving the function "ProcessModules") are implemented as software using the GPenSIM tool [37] on the MATLAB platform.

Since the algorithm, on the whole, cannot fully be implemented as an executable software, experimental evaluation is not possible. However, the running time of the parts of the algorithm that can be implemented as an executable software is presented in section-VII-A.

There is no guarantee that the extraction algorithm presented in this paper will always work. The algorithm will not work if a Petri net model is "highly connected", where most the elements (places and transitions) are connected with each other, and the connections are crisscrossing the model. In this case, modularizing the Petri net with the algorithm presented in this paper will not be possible. However, in reality, Petri nets are streamlined meaning a Petri net can be partitioned into segments due to the different functionalities; in this case, these segments can be extracted as Petri modules using the algorithm provided in this paper.

In engineering, we start with the developing subsystems and finally assemble these subsystems together to create the overall model [38]–[40]. However, the focus of this paper is entirely different. In this paper, as the title clearly states, we start with existing Petri net models (legacy Petri nets); this paper is about how to decompose large and legacy Petri nets into modules so that modular Petri net benefits can be achieved.

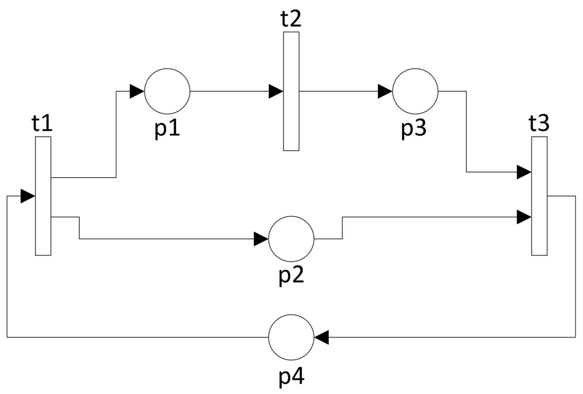


FIGURE 25. T-invariants of a Petri net.

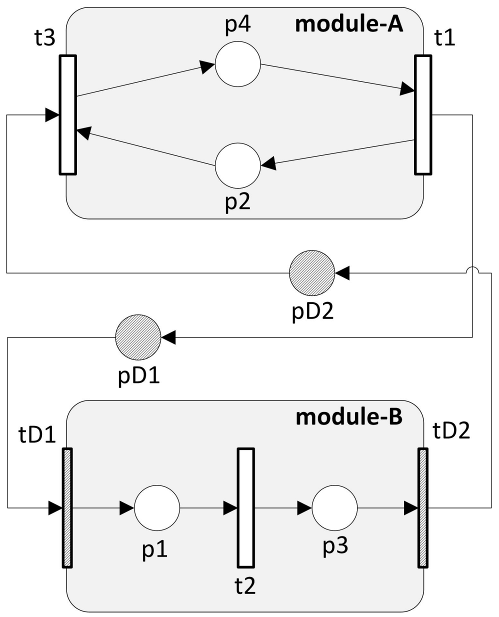


FIGURE 26. T-invariants of a modular Petri net.

Finally, the algorithm is semi-automated meaning the algorithm and the resulting processes heavily dependent on the modeller. The modeller determines the grain size of the extracted model. It is also clearly stated that in the paper that the inexperience or oversight of the modeller can cause problems. For example, an active cycle can be accidentally broken into different modules causing unnecessary delays. The example that is shown in Fig. 25 and 26 is to prove this point.

XI. CONCLUSION

Ever since its inception in 1962, Petri Nets have been used for modeling of discrete event systems. Petri net is still an active research area; research papers on newer applications (e.g., [41], [42]), and the analysis (e.g., [43]–[45]) continue to appear. However, its huge size, and its state-space, and the slow execution (and simulation) prevents its use for modern large real-life applications, e.g., Industry 4.0. The research on Petri nets needs to upgrade Petri nets into a higher level, in which Petri net models (modules) become smaller and faster, distributed and communicating agents.

This paper presents an algorithm by which large and legacy Petri net models can be decomposed into Petri modules. These modules are smaller thus runs (can be executed) faster. Also, these modules can be run on different computers, and the communication between the modules happen via well-defined input and output ports, passing messages (tokens) between them (e.g., in the form of TCP/IP packets).

REFERENCES

[1] R. Davidrajuh, "Experimenting with the static slicing of Petri nets," in *Proc. IEEE 24th Int. Conf. Intell. Eng. Syst. (INES)*, Jul. 2020, pp. 25–30.

[2] R. Davidrajuh and A. Roci, "Performance of static slicing algorithms for Petri nets," *Int. J. Simul., Syst., Sci. Technol.*, vol. 20, p. 15, Mar. 2019.

[3] V. M. Savi and X. Xie, "Liveness and boundedness analysis for Petri nets with event graph modules," in *Proc. Int. Conf. Appl. Theory Petri Nets*. Berlin, Germany: Springer, 1992, pp. 328–347.

[4] J. F. Claver, G. Harhalakis, J. M. Proth, V. M. Savi, and X. L. Xie, "A stepwise specification of a manufacturing system using Petri nets," in *Proc. Conf. Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 1991, pp. 373–378.

[5] G. G. de Jong and B. Lin, "A communicating Petri net model for the design of concurrent asynchronous modules," in *Proc. 31st Annu. Conf. Design Autom. Conf. DAC*, Jun. 1994, pp. 49–55.

[6] L.-C. Wang, "Object-oriented Petri nets for modelling and analysis of automated manufacturing systems," *Comput. Integr. Manuf. Syst.*, vol. 9, no. 2, pp. 111–125, May 1996.

[7] L.-C. Wang and S.-Y. Wu, "Modeling with colored timed object-oriented Petri nets for automated manufacturing systems," *Comput. Ind. Eng.*, vol. 34, no. 2, pp. 463–480, Apr. 1998.

[8] W. J. Lee, S. D. Cha, and Y. R. Kwon, "Integration and analysis of use cases using modular Petri nets in requirements engineering," *IEEE Trans. Softw. Eng.*, vol. 24, no. 12, pp. 1115–1130, Dec. 1998.

[9] Y. Xue, R. M. Kieckhafer, and F. F. Choobineh, "Automated construction of GSPN models for flexible manufacturing systems," *Comput. Ind.*, vol. 37, no. 1, pp. 17–25, Jun. 1998.

[10] S. Christensen, "Modular analysis of Petri nets," *Comput. J.*, vol. 43, no. 3, pp. 224–242, Mar. 2000.

[11] G. J. Tsinarakis, N. C. Tsourveloudis, and K. P. Valavanis, "Modular Petri net based modeling, analysis, synthesis and performance evaluation of random topology dedicated production systems," *J. Intell. Manuf.*, vol. 16, no. 1, pp. 67–92, Feb. 2005.

[12] N. Tsourveloudis, "Fuzzy work-in-process inventory control of unreliable manufacturing systems," *Inf. Sci.*, vol. 127, nos. 1–2, pp. 69–83, Aug. 2000.

[13] H. Lee and A. Banerjee, "A modular Petri net based architecture to model manufacturing systems exhibiting resource and timing uncertainties," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Aug. 2009, pp. 525–530.

[14] J. I. Latorre-Biel, E. Jiménez-Macías, J. L. García-Alcaraz, J. C. S.-D. Muro, J. Blanco-Fernandez, and M. P. D. L. Parte, "Modular construction of compact Petri net models," *Int. J. Simul. Process Model.*, vol. 12, no. 6, pp. 515–524, 2017.

[15] O. Bonnet-Torrès, P. Domenech, C. Lesire, and C. Tessier, "E xhost-pipe: Pipe extended for two classes of monitoring Petri nets," in *Proc. Int. Conf. Appl. Theory Petri Nets*. Berlin, Germany: Springer, 2006, pp. 391–400.

[16] M. A. Blätke, S. Meyer, and W. Marwan, "Pain signaling-a case study of the modular Petri net modeling concept with prospect to a protein-oriented modeling platform," in *Proc. 2nd Int. Workshop Biol. Processes Petri Nets (BioPPN)*, Newcastle Upon Tyne, U.K., Jun. 2011, pp. 1–19.

[17] C. Mahulea, J.-M. Garcia-Soriano, and J.-M. Colom, "Modular Petri net modeling of the spanish health system," in *Proc. IEEE 17th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2012, pp. 1–8.

[18] C. Mahulea, L. Mahulea, J. M. García Soriano, and J. M. Colom, "Modular Petri net modeling of healthcare systems," *Flexible Services Manuf. J.*, vol. 30, nos. 1–2, pp. 329–357, Jun. 2018.

[19] M. Dotoli, N. Epicoco, M. Falagario, and G. Cavone, "A timed Petri nets model for performance evaluation of intermodal freight transport terminals," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 842–857, Apr. 2016.

[20] M. dos Santos Soares and J. Vrancken, "A modular Petri net to modeling and scenario analysis of a network of road traffic signals," *Control Eng. Pract.*, vol. 20, no. 11, pp. 1183–1194, Nov. 2012.
[21] A. Słota, J. Zając, and M. Uthayakumar, "Synthesis of Petri net based model of a discrete event manufacturing system for nonlinear process plan," *Manage. Prod. Eng. Rev.*, vol. 7, no. 2, pp. 62–72, Jun. 2016.
[22] R. Davidrajuh, "Distributed workflow based approach for eliminating redundancy in virtual enterprising," *J. Supercomput.*, vol. 63, no. 1, pp. 107–125, Jan. 2013.
[23] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manuf. Lett.*, vol. 3, pp. 18–23, Jan. 2015.
[24] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg, "How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective," *Int. J. Mech. Ind. Sci. Eng.*, vol. 8, no. 1, pp. 37–44, 2014.
[25] J. Davis, T. Edgar, J. Porter, J. Bernaden, and M. Sarli, "Smart manufacturing, manufacturing intelligence and demand-dynamic performance," *Comput. Chem. Eng.*, vol. 47, pp. 145–156, Dec. 2012.
[26] S. Berger, M. Bogenreuther, B. Häckel, and O. Niesel, "Modeling availability risks of IT threats in smart factory networks—A modular Petri net approach," in *Proc. 27th Eur. Conf. Inf. Syst. (ECIS)*, Stockholm-Uppsala, Sweden: Association for Information Systems, AIS Electronic Library (AISeL), 2019, pp. 1–17.
[27] R. Davidrajuh, B. Skolud, and D. Krenczyk, "Performance evaluation of discrete event systems with gpensim," *Computers*, vol. 7, no. 1, pp. 1–8, 2018.
[28] R. Davidrajuh, "A new modular Petri net for modeling large discrete-event systems: A proposal based on the literature study," *Computers*, vol. 8, no. 4, p. 83, Nov. 2019.
[29] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 1981.
[30] R. Davidrajuh and C. Rong, "Finding the pivotal elements for modularization of Petri nets," in *Proc. Int. Conf. Adv. Mech. Syst. (ICAMechS)*, Aug. 2019, pp. 92–97.
[31] J. Jensen and W. S. Kendall, *Networks and Chaos-Statistical and Probabilistic Aspects*, vol. 50. Boca Raton, FL, USA: CRC Press, 1993.
[32] R. Davidrajuh, B. Skolud, and D. Krenczyk, "Gpensim for performance evaluation of event graphs," in *Advances in Manufacturing* (Lecture Notes in Mechanical Engineering), vol. 201519. Cham, Switzerland: Springer, 2018, pp. 289–299.
[33] J. Martínez and M. Silva, "A simple and fast algorithm to obtain all invariants of a generalised Petri net," in *Application and Theory of Petri Nets*. Berlin, Germany: Springer, 1982, pp. 301–310.
[34] G.-J. Liu and C.-J. Jiang, "Incidence matrix based methods for computing repetitive vectors and siphons of Petri net," *J. Inf. Sci. & Eng.*, vol. 25, no. 1, pp. 1–16, 2009.
[35] R. Davidrajuh, *Modeling Discrete-Event Systems With GPenSIM*. Cham, Switzerland: Springer, 2018.
[36] F. DiCesare, G. Harhalakis, J.-M. Proth, M. Silva, and F. Vernadat, *Practice of Petri Nets in Manufacturing*. Dordrecht, The Netherlands: Springer, 1993.
[37] GPenSIM. (2019). *General-Purpose Petri Net Simulator*. Accessed: Jul. 20, 2020. [Online]. Available: http://www.davidrajuh.net/gpensim
[38] M. Zhou, F. DiCesare, and A. A. Desrochers, "A hybrid methodology for synthesis of Petri net models for manufacturing systems," *IEEE Trans. Robot. Autom.*, vol. 8, no. 3, pp. 350–361, Jun. 1992.
[39] G. J. Liu, C. J. Jiang, Z. H. Wu, and L. J. Chen, "A live subclass of Petri nets and their application in modeling flexible manufacturing systems," *Int. J. Adv. Manuf. Technol.*, vol. 41, nos. 1–2, pp. 66–74, Mar. 2009.
[40] C. Xia and C. Li, "Property preservation of Petri synthesis net based representation for embedded systems," *IEEE/CAA J. Automatica Sinica*, early access, Jan. 16, 2020, doi: 10.1109/JAS.2020.1003003.
[41] D. A. Zaitsev, T. R. Shmeleva, and J. F. Groote, "Verification of hypertorus communication grids by infinite Petri nets and process algebra," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 3, pp. 733–742, May 2019.
[42] J. Zhou, J. Wang, and J. Wang, "A simulation engine for stochastic timed Petri nets and application to emergency healthcare systems," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 4, pp. 969–980, Jul. 2019.
[43] S. Wang, W. Duo, X. Guo, X. Jiang, D. You, K. Barkaoui, and M. Zhou, "Computation of an emptiable minimal siphon in a subclass of Petri nets using mixed-integer programming," *IEEE/CAA J. Automatica Sinica*, early access, Jun. 2, 2020, doi: 10.1109/JAS.2020.1003210.
[44] M. Agarwal, S. Biswas, and S. Nandi, "Discrete event system framework for fault diagnosis with measurement inconsistency: Case study of rogue DHCP attack," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 3, pp. 789–806, May 2019.
[45] B. Huang, M. Zhou, P. Zhang, and J. Yang, "Speedup techniques for multiobjective integer programs in designing optimal and structurally simple supervisors of AMS," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 1, pp. 77–88, Jan. 2018.

REGGIE DAVIDRAJUH (Senior Member, IEEE) received the master's degree in control systems engineering and the Ph.D. degree in industrial engineering from the Norwegian University of Science and Technology (NTNU), in 1993 and 2001, respectively, and the D.Sc. (Dr.Hab.) degree in informatics from the AGH University of Science and Technology, Poland, in 2016. He is currently a Professor of informatics with the Department of Electrical Engineering and Computer Science, University of Stavanger, Norway. He is also a Visiting Professor with the Silesian University of Technology, Poland. His current research interests include discrete-event systems, Petri nets, and graph algorithms. He is an Editor of the *International Journal of Business and Systems Research* (Inderscience) and an Associate Editor of *Expert Systems with Applications* (Elsevier). He developed the software General-purpose Petri Net Simulator (GPenSIM). Some universities around the world use GPenSIM for modeling and simulation of discrete-event systems.