Performance of Static Slicing Algorithms for Petri Nets

Reggie Davidrajuh [1], Albana Roci [2]

Electrical and Computer Engineering
University of Stavanger, Stavanger, Norway.

[1] Reggie.Davidrajuh@uis.no, [2] Albana.Roci@uis.no.

***Abstract*—This paper focuses on algorithms for static slicing of Petri Nets. This paper presents the implementation of some of these algorithms on the General-purpose Petri Net Simulator (GPenSIM). Also, a new place-invariant based algorithm for static slicing is presented which is more robust to changes in the selection criterion, thus more suitable for model verification stages. However, the experiments presented in the latter part of this paper suggest that all the known static algorithms are impractical for verification of Petri Net models of large real-life systems. Since most of the real-life discrete-event dynamic systems (notably, manufacturing systems) are cyclic (or repetitive), and the Petri Net models of those systems are event graphs. The existing static slicing algorithms are not useful for event graphs as they return the entire Petri Net as the slice. The other type of slicing algorithms – the dynamic algorithms – only works for specific instances of Petri Nets; thus, the applicability of dynamic slicing for model checking is also limited. Therefore, this paper concludes that more work is needed on the static slicing for model verification of large real-life discrete-event systems.**

Keywords- Petri Nets; Static Slicing Algorithms; GPenSIM

I. INTRODUCTION

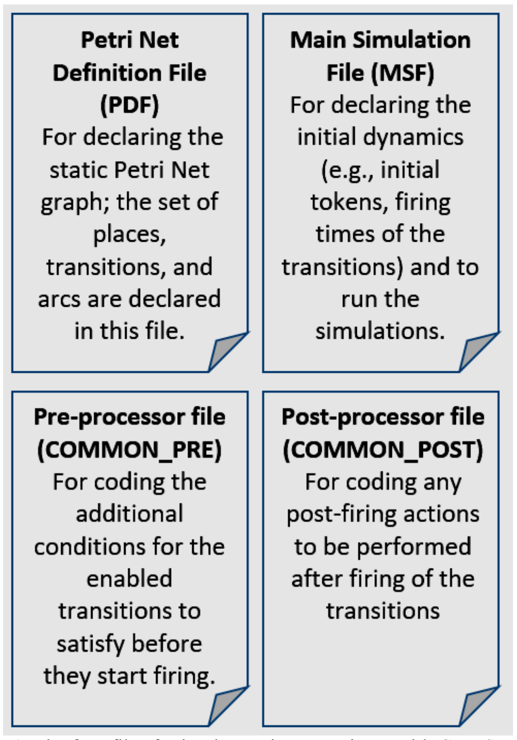
Model checking (model verification of) a Petri Net is a time-consuming task, if not impractical, due to the huge size of the state space (the so-called 'state space explosion problem'). Hence, Petri Net slicing has become active research to find various ways of reducing the size of the Petri Nets and therefore the resulting state space. This paper focuses on one type of slicing, the static slicing.

In this paper: section-II introduces the General-purpose Petri Net Simulator (GPenSIM). Section-III introduces the basics in model checking and Petri Nets. Section-IV presents some of the static slicing algorithms, and a new algorithm is proposed in Section-V. In section-VI, some case studies are done to measure the performances of the algorithms. Finally, the results are discussed in section-VII.

II. GENERAL-PURPOSE PETRI NET SIMULATOR (GPENSIM)

GPenSIM is a simulator for modeling, simulation, and performance analysis of discrete-event systems; it is also used as a model-based controller [1]. GPenSIM is developed by the first author of this paper, and it is freely available for academic and commercial use [2]. GPenSIM runs on MATLAB platform and is being used by some universities around the world, for modeling and simulation large discrete-event systems (e.g., [3, 4]). The reasons for the acceptance being the simplicity of learning and using, and its flexibility to incorporate newer functionality, and its ability to control (model-based control) of external hardware and software [3].

As shown in figure 1, a Petri Net model is implemented with GPenSIM via four MATLAB files (M-files) [1]. With these four simple files, many industrial large-scale discrete problems were modeled and solved in diverse engineering disciplines.


Petri Net Definition File (PDF)
For declaring the static Petri Net graph; the set of places, transitions, and arcs are declared in this file.
Main Simulation File (MSF)
For declaring the initial dynamics (e.g., initial tokens, firing times of the transitions) and to run the simulations.
Pre-processor file (COMMON_PRE)
For coding the additional conditions for the enabled transitions to satisfy before they start firing.
Post-processor file (COMMON_POST)
For coding any post-firing actions to be performed after firing of the transitions
Figure 1. The four files for implementing a Petri Net with GPenSIM.

III. Model Checking and Petri Nets: Formal Definitions

This section is for formal definitions of model checking and Petri Nets. This section is concise. For a detailed study on model checking, the standard textbook such as [5] is suggested. The textbook [6] is recommended for a comprehensive study on Petri Nets.

A. Model Checking

Model checking exhaustively checks all the states of the system, to see whether any state meets a given property specification [5]. Formally, the problem can be stated as follows: $\mathbf{M}, \mathbf{s} \models \boldsymbol{\phi}$, where, $\boldsymbol{\phi}$ is a desired property expressed as a temporal logic formula, and **M** is a mathematical model (e.g., reachability graph of a Petri Net, in the form of a Labeled Transition System). Model checking is to verify whether there exists a state (or a set of states) **s** that satisfies the property specification $\boldsymbol{\phi}$.

B. Petri Nets

Petri Nets is a formalism for modeling and simulation of discrete-event systems. Since its inception, Petri Nets have gone through many versions (extensions and subclasses) mainly to incorporate time and to increase its modeling power [6].

The Place-Transition Petri Net (P-T Petri Net) is the simplest version of Petri Nets. The P-T Petri Net is defined as a four-tuple PTN = (P, T, A, m_0) [6], where,

- P is the set of places, $P = \{p_1, p_2, \dots, p_{n1}\}$.
- T is the set of transitions, $T = \{t_1, t_2, \dots, t_{n2}\}$.
- A is the set of arcs (from places to transitions and from transitions to places). A is a subset of $(P \times T) \cup (T \times P)$.
- m is the row vector of markings (tokens) on the set of places, $m = [m(p_1), m(p_2), \dots, m(p_{n1})] \in N^{n1}$. m_0 is the initial marking.

Figure 2 shows a simple P-T Petri Net that consists of three places and one transition, connected by three arcs. The initial marking m_0 on the Petri Net is four, three, and one token, on the places p_1, p_2, and p_3, respectively.

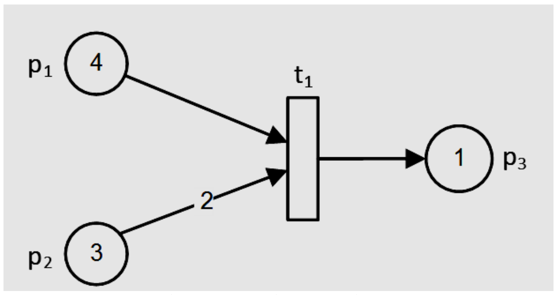


Figure 2. A minute Petri Net.

IV. Static Slicing Algorithms

This section serves as the literature study on the Static Slicing Algorithms for Petri Nets. Slicing of Petri nets is to make a Petri net model thinner by removing the elements (places and transitions) that are not relevant to a particular property. The sliced Petri Net should be simpler and smaller than the original Petri Net. The slice should contain only those elements that may influence the property that is being checked. The slice, being smaller than the original Petri Net, should provide smaller state space for model checking.

The slicing algorithms can be generally grouped into two groups: a) Static slicing algorithms, and b) Dynamic slicing algorithms. The static algorithms work on the static Petri Net, and hence the token game (the movement (flow) of tokens during the run of the Petri Net) is not considered. Whereas, in dynamic algorithms, the flow tokens are taken into consideration. The initial tokens are input to the dynamic algorithms. Hence, a dynamic algorithm works on a specific instance of a Petri Net, whereas a static algorithm on the general case. This means, the advantage of dynamic algorithms is that a dynamic algorithm is better suited to provide thinner slices than its static counterpart, as the dynamic algorithm work on a specific instance; thus, the places and the transitions that are not relevant to the specific instance can be sliced away. However, the disadvantage of dynamic algorithms is that they provide slices that are valid only for the specific instances (the initial tokens).

All the static algorithms that exist today are improved versions of the basic slicing algorithm. In the following subsections, three well-known static slicing algorithms are presented.

```
BasicSlice(PN, SC)
T′ = Ø;
P′ = SC;
Pdone = Ø;
P_Pdone = P′\Pdone;
while not(empty(P_Pdone))
{ p ϵ P_Pdone
    Pdone = Pdone U p
    for every t ϵ (*p U p*)
       { if t not in T′
            T′ = T′ U t
            P′ = P′ U *t
       }
P_Pdone = P′\Pdone;
}
```

Figure 3. The Basic Slicing Algorithm [7].

C. The Basic Slicing Algorithm

The Basic Slicing algorithm is the simplest algorithm [7]. As shown in figure-3, the inputs to the algorithms are the original Petri Net **PN** and the slicing criterion **SC**. The idea behind the algorithm is rather simple: when a place becomes part of the slice, then its input and out transitions are also taken into the slice as these transitions lead the tokens in and out of places. However, when a transition becomes part of the slice, only its input places are of interest, as only these input places can make the transition enabled or not.

The output of the algorithm is the set of selected places (**P′**) and the set of selected transitions (**T′**). The Petri Net slice will be composed of **P′** and **T′** and the all arcs between the elements in these two sets.

D. The Refined Slicing Algorithm

The Refined Slicing algorithm is a simple improvement over the basic algorithm. The improvement is the following [8]: whenever a transition is considered for the slice, it is taken only if it is a *non-reading* transition. A reading transition returns the same number of tokens it takes from a place. Thus, the firing of a reading transition does not effectively play a role in the flow of tokens through the concerned place. If a Petri Net has many reading transitions (in general cases, it is less likely for a Petri Net to have many reading transitions), then the refined algorithm will provide a thinner slice. The refined algorithm is shown in figure 4.

```
RefinedSlice(PN, SC)
T′ = Ø;
P′ = SC;
Pdone = Ø;
P_Pdone = P′\Pdone;
while not(empty(P_Pdone))
{ p ϵ P_Pdone
  Pdone = Pdone U p
  for every t ϵ (*p U p*)
  { if (t is not reading) and
        (t not in T′)
            T′ = T′ U t
            P′ = P′ U *t
  }
  P_Pdone = P′\Pdone;
 }
```

Figure 4. The Refined Slicing Algorithm [8].

E. The Safety Slicing Algorithm

The Safety Slicing algorithm can be considered as a further improvement of the Refined Slicing algorithm. Rakow [8] suggests running the Safety Slicing algorithm as a pre-processor, as it mainly removes “stuttering transitions.” A stuttering transition is a transition that removes fewer tokens from a place than it deposits into the place. Thus, the transition can fire an infinite number of times, keeping the run of the net on its position. The Safety Slicing algorithm (figure 5) will be beneficial only if the Petri Net has many stuttering transitions.

V. Proposing A New Place-Invariant based Algorithm

This is a new algorithm for static slicing of Petri Nets, proposed by the authors of this paper for the first time. The Place-Invariant algorithm is - just like all other static slicing algorithms – is an improvement of the basic slicing algorithm. The new algorithm is based on the fact that the place-invariants are *inseparable* and thus has to be placed as a whole inside the slice. It is noteworthy that if the slicing criterion is not a subset of the place-invariant, then the algorithm will fail to produce a slice.

```
SafetySlice(PN, SC)
T′ = Ø;
P′ = SC;
Pdone = Ø;
P_Pdone = P′\Pdone;
while not(empty(P_Pdone))
{ p ϵ P_Pdone
  Pdone = Pdone U p
  for every t ϵ (*p U p*)
     { if (t is not stuttering) and
          (t not in T′)
          {
             T′ = T′ U t
             P′ = P′ U *t
          }
     }
  P_Pdone = P′\Pdone;
}
```

Figure 5. The Safety Slicing Algorithm [8].

```
PIntSlice(PN, SC, PISize)
$P_i$′= Ø
$P_i$′= {(place-invariant)$_i$ |($P_i$′ ⊇ SC)}
if $P_i$′= Ø  % No eligible place-inv found
{  % slicing not possible
   return
}

if PISize = “min” {P′= |$P_i$′|$_{min}$}
else % PISize=“max”% {P′= |$P_i$′|$_{max}$}

T′ = Ø;
Pdone = Ø;
P_Pdone = P′\Pdone;
while not(empty(P_Pdone))
{ p ϵ P_Pdone
   Pdone = Pdone U p
   for every t ϵ (*p U p*)
      { if t not in T′
         T′ = T′ U t
         P′ = P′ U *t
      }
  P_Pdone = P′\Pdone;
}
```

Figure 6. The Place-invariant based Slicing Algorithm.

If the Petri Net possesses many sets of place invariants that include the selection criterion, then one of the place invariant is selected as the initial set of places of the slice. Again, the idea behind this algorithm (shown in figure 6) is to keep the inherently inseparable places that include the selection criterion into the slice.

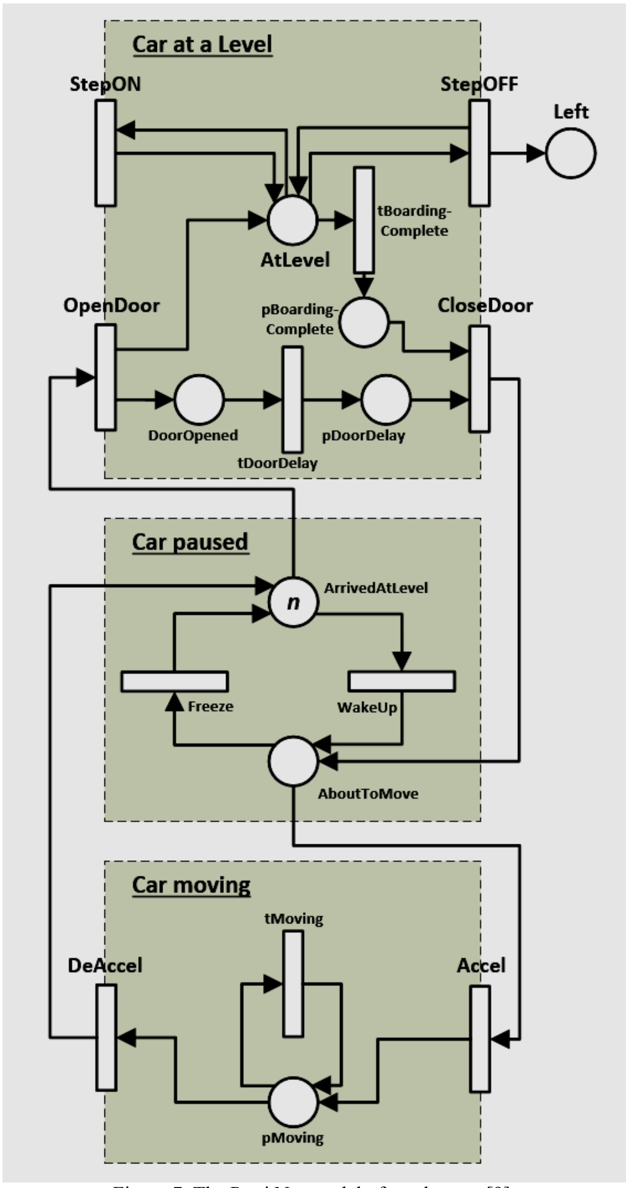


Figure 7. The Petri Net model of an elevator [9].

The advantage of the new algorithm is that it eliminates the need to create new slices should the selection criterion changes. It is noteworthy that all other algorithms produce a slice specific to the selection criterion. If the selection criterion changes even by one place, then the new corresponding slice has to be produced. This is unnecessary and time-consuming, especially during the model verification stages in which it is usual to vary the selection criterion. In the new algorithm, the starting point is the place-invariant that comprises of (is a superset of) the selection criterion. Thus, the selection criterion can be changed as long as the new places in the selection criterion are also members of the place-invariant.

The new algorithm also takes an optional third parameter ***`PISize`***. The third parameter allows the choice of starting with the largest place-invariant or the smallest. If the largest invariant is preferred, then it will enable large variations in the selection criterion, but also create a larger slice. On the contrary, the smallest invariant limits changes in the selection criterion, but provide a smaller slice.

VI. Implementing and Testing the Slicing Algorithms in GPenSIM

All the four Slicing algorithms discussed in the previous sections are implemented in GPenSIM. Due to space limitation, the GPenSIM program codes for the algorithms are not shown in this paper. However, the reader can download the code from the website [2] and reproduce the results shown in this paper.

For case studies, three Petri Net models of real-life discrete-event systems are considered. All three models are taken from the first author's earlier works on modeling of large real-life discrete-event systems [9-11]. The models are purposely tailored to be compact to fit the page and size limitations of this paper. The tables in each subsection summarize the sizes of the generated slices by the four slicing algorithms. In the tables, n_P, n_T, and n_A stands for the number of places, transitions, and arcs, respectively.

F. Case Study-I: Modeling Elevator Operations

Figure 7 shows the Petri Net model of an elevator. The Petri Net model is composed of three modules such as:

- "Car at a level" that captures the activities that are involved when the car arrives at a level (e.g., open the door, let passengers leave and enter, and close door).
- "Car paused" that deals with freezing the elevator when no passengers are traveling or waiting at any levels.
- "Car moving" that takes the car between levels.

More details of the model are given in [9]. Let us assume that we are concerned about the safeness issue, e.g., anybody who enters the elevator at a level, will eventually leave the elevator. Thus, the selection criterion becomes the following two places: {'AtLevel', 'Left'}.

TABLE I. Sizes of the Slices of the Elevator Model.

Slicing algorithm	n_P	n_T	n_A
Basic Slicing	8	11	25
Refined Slicing	8	9	21
Safety Slicing	7	9	20
Place-invariant based	8	11	25

The table-I shows that the safety algorithm and the refined algorithm perform slightly better. This is because the Petri Net model for the elevator has two "stutters" or diversions that do not involve any of the places in the selection criterion. For example, the transitions **StepON** and **tMoving** are self-loops (stutters). Hence, they are taken away in the slice. However, if the stutters do involve a selection criterion, then the stutters cannot be sliced away. Thus, the benefit the refined algorithm will vanish.

G. Case Study-II: Petri Net model of a Soda Vending Machine

Figure 8 shows the Petri Net model of a soda vending machine. The machine has the following properties [10]:

- It can hold a limited number of drinks after refilling.
- A user can insert only one coin (e.g., a pound coin) to get a drink.
- After inserting the coin, the user can push the buttons for either "Deliver Product" or "Money Back."
- If the "Deliver Product" (or "Money Back") button is pressed, then the drink will be delivered (resp. the inserted coin will be returned).
- The process of refilling is not covered in this problem.

Let us assume that we are interested in verifying the following fairness property: after returning the coin (if "MoneyBack" button is pressed) the system resets to the Ready state. Thus, the selection criterion becomes {'MoneyBack', 'Ready'}.

TABLE II. SIZES OF THE SLICES OF SODA VENDING MACHINE

Slicing algorithm	n_P	n_T	n_A
Basic Slicing	7	7	17
Refined Slicing	7	7	17
Safety Slicing	7	7	17
Place-invariant based	7	7	17

Table-II shows the sizes of the slices generated by the four algorithms. It is clear that all four algorithms return the whole Petri Net model as the slices; not a single place or transition is removed in the slice. Since there are no self-loops or stutters in the Petri Net model, the Refined algorithm and the Safety algorithm failed to make a difference in the slices.

H. Case Study-III: Petri Net model of a Flexible Manufacturing System

Figure 9 shows the Petri Net model of a flexible manufacturing system (FMS) that makes only one type of product [11]. In the FMS:

- Raw materials (type-1 or type-2) arrives on the conveyor belt C1 (resp. C2) that is moved into the CNC machine M1 (resp. M2) by the robot R1 (resp. R2).
- M1 (and M2) makes the part P1 (and P2, resp.). Robots R1 (and R2) moved the part P1 (P2, resp.) into the assembly station AS.
- Robot R2 assembles P1 and P2 into the semi-product PX.
- Robot R3 picks the semi-product PX from the assembly station AS and moves it to the painting station PS. Also, R3 performs the finishing operations (painting and polishing) on the painting station PS. Finally, R3 moves the completed product into the output buffer OB.

The complete details of the Petri Net model are given in [11]. Let us say that we are interested in verifying whether the following property holds: a product P2 that is being produced by M2 will eventually get painted at the painting station PS. Thus, we are focusing on the selection criterion {'pOM2', 'piPS'}.

TABLE III. SIZES OF THE SLICES OF FMS

Slicing algorithm	n_P	n_T	n_A
Basic Slicing	21	12	40
Refined Slicing	21	12	40
Safety Slicing	21	12	40
Place-invariant based	21	12	40

Table III presents the sizes of the slices of the FMS model. As in the previous case study-II, the table clearly shows that the four algorithms returned the whole Petri Net as the slice. Here again, due to the lack of stutters or self-loops, the Refined and Safety algorithms failed to produce any optimization.

VII. DISCUSSION

The previous section experimented with four static slicing algorithm for the slicing of Petri Nets. Three of these algorithms are well-known, and the fourth one is proposed in this paper as a new algorithm. Although literature reveals some works that claim the efficiency of these algorithms, none of the works show any experimentation with real-world examples. This paper, to the best of our knowledge, is the first paper that applies the static slicing algorithms on the Petri Net models of real-life discrete-event systems.

As shown in the previous section on the experimentation with the four algorithms, the existing slicing algorithms perform very poorly on the real-life models. Although the refined and the safety algorithms show some improvement when the models have self-loops and stutters, there is no need (or less likely) for the real-life Petri Net models to have these components. The fourth algorithm is proposed in this paper only as a convenience for model verification, as this algorithm does not necessitate the generation of newer slices for varying slicing criterion. Thus, it is clear that further research is needed to create newer static slicing algorithms that can produce slices with substantial savings.

This paper doesn't focus on dynamic slicing. The dynamic slices are only applicable to specific instances. Specific instance means the initial markings are known, and the initial markings are certain to lead to particular states that involve the selection criterion. Thus, slices obtained by dynamic slicing algorithms are bounded with distinct initial markings, and should the initial markings change (in addition to the selection criterion) new slice has to be produced.

REFERENCES

[1] R. Davidrajuh, Modeling Discrete-Event Systems with GPenSIM: An Introduction. Springer International Publishing. 2018.

[2] General-purpose Petri Net Simulator (GPenSIM): http://www.davidrajuh.net/gpensim/ (2019).

[3] A. Cameron, M. Stumptner, N. Nandagopal, W. Mayer, & T. "Rule-based peer-to-peer framework for decentralised realtime service oriented architectures." Science of Computer Programming, 97, 202-234, 2015.

[4] U. Mutarraf, K. Barkaoui, Z. Li, N. Wu, and T. Qu. "Transformation of Business Process Model and Notation models onto Petri nets and their analysis." Advances in Mechanical Engineering, 10(12), 2018. Doi: 1687814018808170.

[5] C. Baier, and JP. Katoen. Principles of model checking. MIT press, 2008.

[6] J. L. Peterson. Petri net theory and the modeling of systems. 1981.

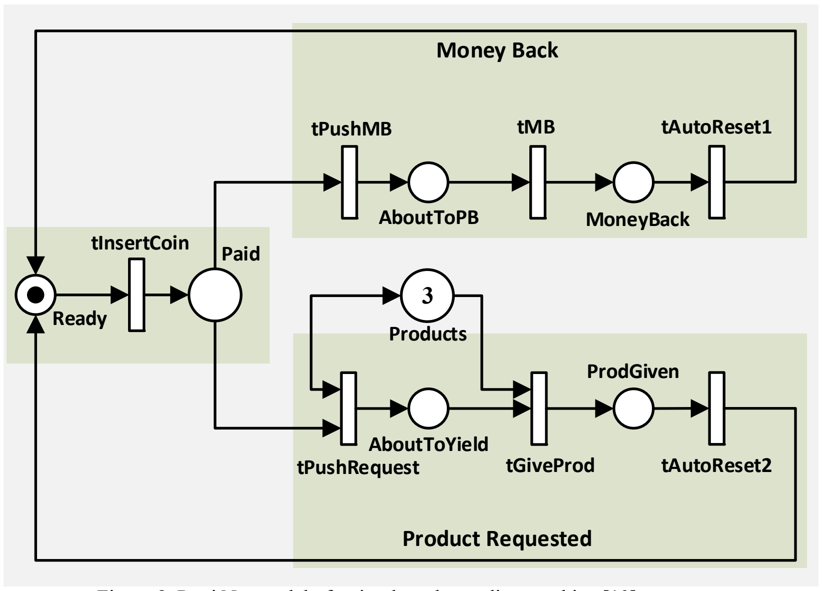
[7] M. Weiser. "Program slicing." Proc. 5th international conference on Software engineering. IEEE Press, 1981.

[8] A. Rakow. "Safety slicing petri nets." Int C. on Application and Theory of Petri Nets and Concurrency. Springer, Berlin, Heidelberg, 2012.

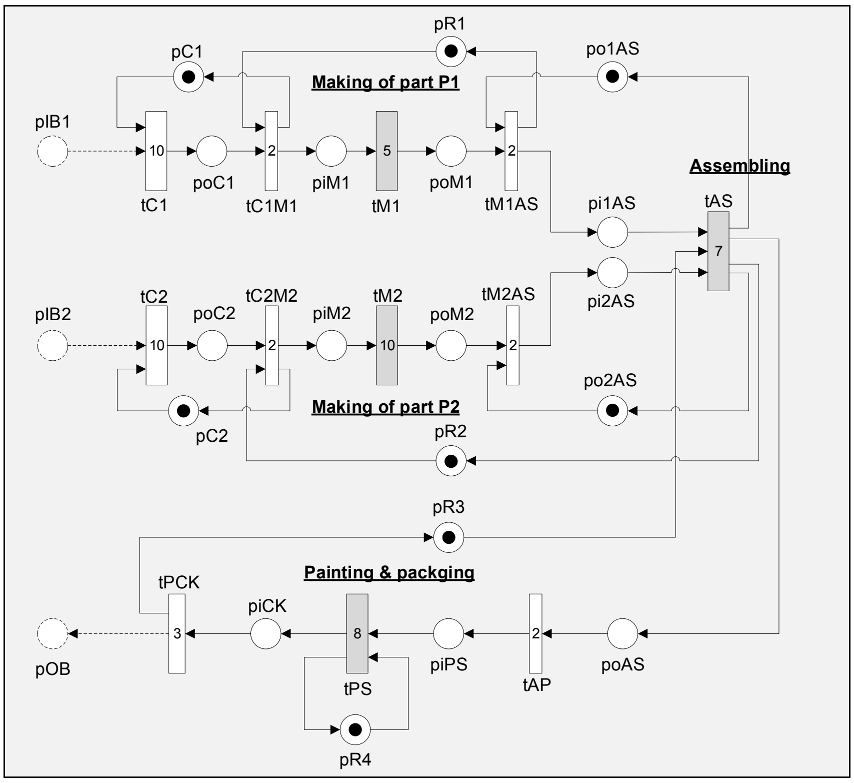
[9] R. Davidrajuh, and D. Krenczyk. "Extending GPenSIM for Model Checking on Petri Nets." International Journal of Simulation--Systems, Science & Technology 20.1 (2019).

[10] R. Davidrajuh. "Developing a Toolbox for Modeling and Simulation of Elevators". In 2019 UKSim-AMSS 21st International Conference on Computer Modelling and Simulation (UKSim). IEEE.

[11] R. Davidrajuh, B. Skolud, and D. Krenczyk. "Performance Evaluation of Discrete Event Systems with GPenSIM." Computers 7.1 (2018): 8.



–Figure 8. Petri Net model of a simple soda vending machine [10].



–Figure 9. Petri Net model of a Flexible Manufacturing System [11].