Article

Not peer-reviewed version

# Enhancing Supervisory Control with GPenSIM

Reggie Davidrajuh *, Shuanglin Tang, Yuming Feng

Posted Date: 20 June 2025

doi: 10.20944/preprints202506.1721.v1

Keywords: supervisory control; petri nets; extended petri nets with enhanced functions; GPenSIM

*Article*

# Enhancing Supervisory Control with GPenSIM

**Reggie Davidrajuh [1,*], Shuanglin Tang [2] and Yuming Feng [3]**

[1] Faculty of Science and Technology, University of Stavanger, 4036 Stavanger, Norway
[2] School of Artificial Intelligence, Chongqing Three Gorges Vocational College, Wanzhou 404155, China
[3] School of Computer Science and Engineering, Chongqing Three Gorges University, Wanzhou 404100, China
[*] Correspondence: reggie.davidrajuh@uis.no (R.D.)

**Abstract**

Supervisory control theory (SCT) based on Petri nets offers a robust framework for modeling and controlling discrete-event systems but faces significant challenges in scalability, expressiveness, and practical implementation. This paper introduces GPenSIM, a MATLAB-based modular Petri net framework, as a novel solution to these limitations. GPenSIM leverages modular decomposition to mitigate state-space explosion, enabling par-allel execution of weakly coupled Petri modules on multi-core systems. Its programmable interfaces (pre-processors and post-processors) extend classical Petri nets' expressiveness by enforcing nonlinear, temporal, and conditional constraints through custom MATLAB scripts, addressing the rigidity of traditional linear constraints. Furthermore, GPenSIM's integration with MATLAB facilitates real-time control synthesis, performance optimization, and seamless interaction with external hardware and software, bridging the gap between theoretical models and industrial applications. Empirical studies demonstrate GPenSIM's efficacy in reconfigurable manufacturing systems, where it reduced downtime by 30%, and in distributed control scenarios, where decentralized modules minimized synchro-nization delays. Grounded in systems theory principles of interconnectedness, GPenSIM emphasizes dynamic relationships between components, offering a scalable, adaptable, and practical tool for supervisory control. This work highlights GPenSIM's potential to overcome longstanding limitations in SCT, providing a versatile platform for both academic research and industrial deployment.

**Keywords:** supervisory control; petri nets; extended petri nets with enhanced functions; GPenSIM

---

## 1. Introduction

Supervisory control theory in the context of Petri nets is a formal framework for modeling, analyzing, and synthesizing controllers for discrete-event systems [1,2]. In this framework, the uncontrolled system ("plant") is represented as a Petri Net, where places represent system states or conditions, transitions represent events (e.g., actions, tasks), and tokens represent the resources or signals [3]. Control specifications for the desired behaviors (such as a maximum of ten tokens in place **p1**; buffer control via monitor places [4]) are formalized as linear constraints (e.g., linear inequalities on markings) [5]. A supervisor is synthesized to enforce specifications by either disabling transitions that would violate constraints or monitoring markings (system states) in real-time [2]. The approach for the synthesis of supervisors is based on Place Invariants (P-invariants), which enforce linear constraints on markings (e.g., via control places added to the plant Petri Net) [5], and maximally permissive control by which all legal behaviors are allowed while illegal ones are prevented [6]. Using Petri Nets for supervisory control offers many advantages, such as a graph- based visual front and linear algebraic-based mathematical modeling and analysis on the back end, handling concurrency and resource sharing naturally, and supporting modular supervisor design [3,7,8]. However, it has several limitations, particularly in scalability, expressiveness, and practical implementation (explained in Section 2). Some of these limi- tations can be overcome with extended Petri Nets such as Colored Petri Nets (for complex systems with data/variable dependencies) [9] and Timed Petri Nets and Stochastic Petri Nets (for performance-oriented real-time control) [10]. However, most of the limitations remain unresolved. This paper provides a novel approach based on GPenSIM to address these gaps. The structure of this paper is as follows: Section 2 presents a compact literature review on the limitations of supervisory control based on Petri nets. Sections 3 and 4 provide formal definitions of Petri nets, as these definitions are expected in works on the modeling of discrete systems. Section 5 introduces GPenSIM, and Section 6 shows how GPenSIM can mitigate some of the limitations of supervisory control. Finally, Section 7 summarizes the findings of this paper.

## 2. Literature Review

While supervisory control theory based on Petri nets is a powerful framework for discrete-event systems [5,11,12], it has several limitations, particularly in scalability, expressiveness, and practical implementation.

### *2.1. State-Space Explosion*

The computational complexity of synthesizing supervisors grows exponentially with the size of the plant (number of places, transitions, and tokens). This so-called "state-space explosion" is a major problem when Petri nets are used, and also for the synthesis of super- visors [13]. Hence, this problem prevents the applicability of Petri nets for controlling large or complex systems such as industrial automation or multi-agent-based communication systems. One way to eliminate the state-explosion problem is to modularize the monolithic Petri net model into multiple "Petri Modules," which can be run on multiple host CPUs. Thus, this modular approach eliminates the state-explosion problem as well as reduces the overall simulation time, as multiple CPUs host these modules [14]. However, the monolithic Petri net model must intrinsically possess laminar flow (no criss-crossing connections) so that it can be decomposed into modules. Also, model abstraction has been proposed as a solution [15]. Here, again, abstraction means loss of information; how much information are we willing to sacrifice in order to reduce the number of states—it is a trade-off between depth of information and number of states.

### *2.2. Scalability in Distributed Systems*

Supervisors synthesized by supervisory control theory are meant for centralized control, as there are no measures for distributed control. However, centralized supervisors may not scale well for decentralized or distributed systems due to communication delays and may not work at all in synchronization issues [8]. Hence, Davidrajuh [8] and Yoo and Lafortune [16] propose decentralized or distributed supervisors with localized modular control and minimal coordination between the modules.

### *2.3. Limited Expressiveness for Nonlinear Constraints*

Classical Petri Net-based supervisory control relies on linear constraints (e.g., place invariants), and it does not support nonlinear constraints. Also, there is no way to enforce temporal, priority-based, or conditional constraints (e.g., "If a fire event occurs, disable the machine for 60 seconds"). Some temporal constraints can be enforced using Timed/S- tochastic Petri nets [10]. Also, Basile et al. [6] suggested the use of hybrid methods (such as integrating Petri nets with Automata or logic programming); however, a formal, robust approach does not exist.

### *2.4. Rigid Enforcement of the Controller*

Supervisors synthesized by supervisory control theory often disable all transitions leading to illegal states, which may be too conservative. This excessive conservativeness reduces system flexibility and efficiency, as sometimes the controller blocks legal behaviors unnecessarily. Ghaffari et al. [17] suggested maximally permissive control using the theory of regions as a way to relax the controller's over-restrictiveness. Yang et al. [18] and Herzig et al. [19] suggested partial observability-aware synthesis to relax the controller.

### *2.5. Partial Observability Challenges*

If some transitions are unobservable, the supervisor cannot always determine the exact system state, and this may lead to incorrect disabling of transitions. Ru and Hadjicostis [20] suggested estimating markings using observers.

### *2.6. Other Difficulties*

There are some other limitations in supervisory control theory based on Petri nets, such as handling unmodeled disturbances. Basile et al. [21] proposed fault-tolerant Petri nets as a remedy and online reconfiguration of supervisors. Also, Wang et al. [10] presented the problem of handling real-time and continuous dynamics in supervisors and suggested an approach based on Timed Petri Nets. Kaid et al. [22] and Al-Ahmari et al. [23] propose a reconfigurable controller to face dynamic changes in the plant. While Petri Net-based supervisory control is a theoretically sound method for synthe sizing controllers for discrete-event systems, its scalability, expressiveness, and adaptability remain challenges. Recent advances (such as hybrid Petri nets and decentralized control) help mitigate these issues, but trade-offs between computational cost and control precision persist.

## 3. Petri Nets

Petri nets have gone through many versions (extensions and subclasses), mainly to incorporate time and to increase their modeling power [7]. In this section, we provide the formal definitions.

### *3.1. Definition: P/T Petri Nets*

**The Place-Transition Petri net** (P/T Petri net, for short) is defined as a four-tuple [7,24,25]:

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

where, $P$ is a finite set of places, $P = \{p_1, p_2, \ldots, p_{n_p}\}$.

- $T$ is a finite set of transitions, $T = \{t_1, t_2, \ldots, t_{n_t}\}$.  $P \cap T = \varnothing$.
- $A$ 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.
- $M$ is the row vector of markings (tokens) on the set of places. $M = [M(p_1), M(p_2), \ldots, M(p_{n_p})] \in N^{np}$, $M_0$ is the initial marking. Due to the markings, a $PTN = (P, T, A, M)$ is also called a **marked P/T Petri net**. P/T Petri nets do not involve time. Hence, P/T Petri nets are insufficient to model engineering systems, as time is one of the *key performance indicators* (*KPI*). Hence, **Timed Petri net** is an extended Petri net incorporating time.

### *3.2. Definition: Timed Petri Net*

**The Timed Petri net** (TPN, for short) is defined as a five-tuple [26,27]:

$$TPN = (P, T, A, M, FT),$$

where,

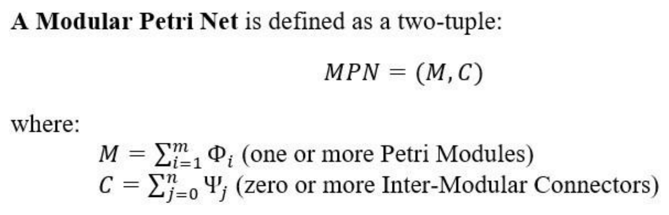
- $(P, T, A, M)$ is a Marked Petri net,
- $FT$ is a mapping of $T$ into $R^+$:  $\forall t \in T$,  $FT(t) \in R^+$,  $FT(t) > 0$
- $FT(t)$ is known as the **firing times** of the transitions and is defined to be greater than zero.

## 4. Modular Petri Nets

The literature review reveals many definitions of Modular Petri Nets [28–33]. Davidrajuh [8,34] proposed a new type of Modular Petri Nets, which is implemented in the software GPenSIM.

*Modular Petri Net and Petri Modules*

A modular Petri net is composed of one or more Petri modules and zero or more Inter-Modular connectors (IMC). Figure 1 shows the definition of modular Petri nets.



**A Modular Petri Net** is defined as a two-tuple:

$$MPN = (M, C)$$

where:

$M = \sum_{i=1}^{m} \Phi_i$ (one or more Petri Modules)

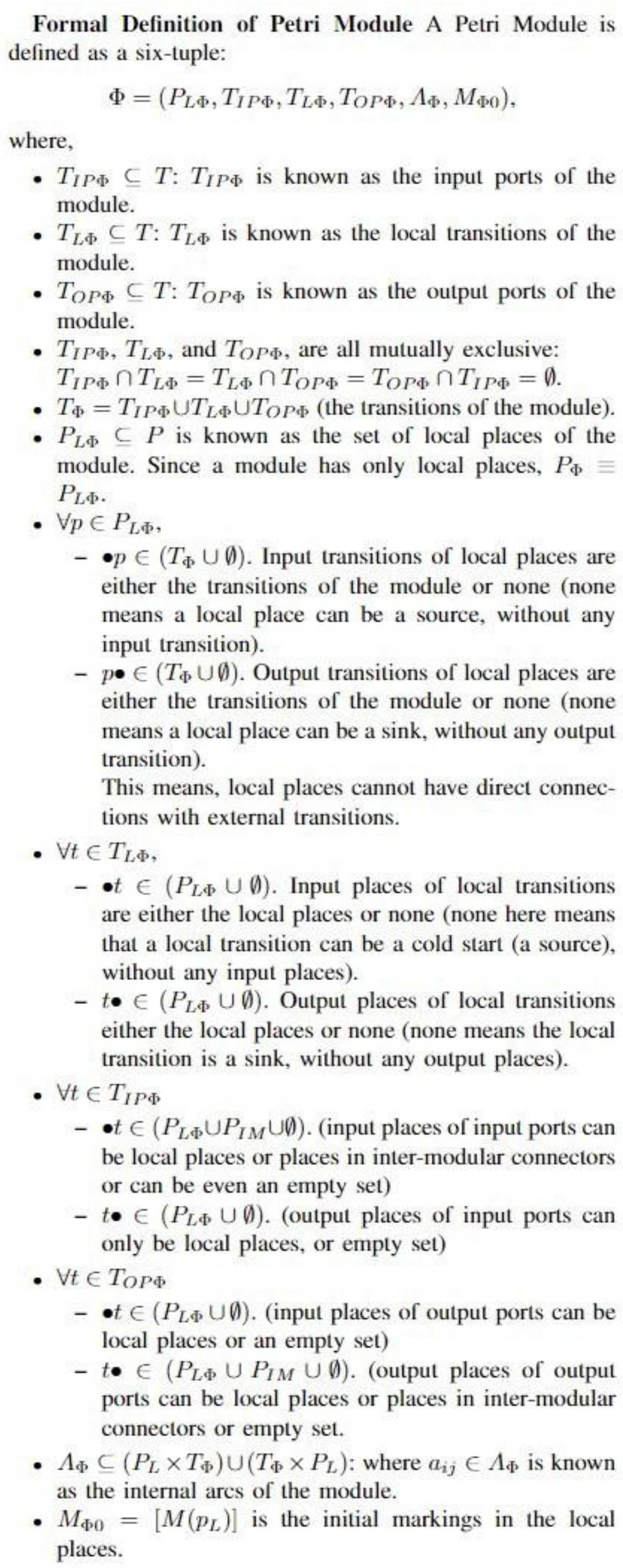
$C = \sum_{j=0}^{n} \Psi_j$ (zero or more Inter-Modular Connectors)

**Figure 1.** Definition of a Modular Petri Net [8].

Figures 2 and 3 show the definitions of a Petri module and IMC, respectively.

## 5. GPenSIM: A Modular Petri Net Framework for Scalable Supervisory Control

**GPenSIM** (General Purpose Petri Net Simulator) is a MATLAB-based Petri net tool developed by the first author of this paper. A *C*++ programming language-based prototype of GPenSIM was initially developed, called PenSIM (Petri Net Simulator). However, the prototype faced significant limitations in managing interdependencies — both among the Petri net elements and with external systems. To address these challenges, GPenSIM was redeveloped as a MATLAB toolbox, offering improved functionality and flexibility. GPenSIM stands out from other Petri net tools due to its **flexibility**, **integration with MATLAB**, and **focus on customization** [35,36]. GPenSIM provides an interface – the **pre-processors** and **post-processors** - that gives a distinctive programming approach to make Petri net models flexible, as explained in the following subsections.



**Formal Definition of Petri Module** A Petri Module is defined as a six-tuple:

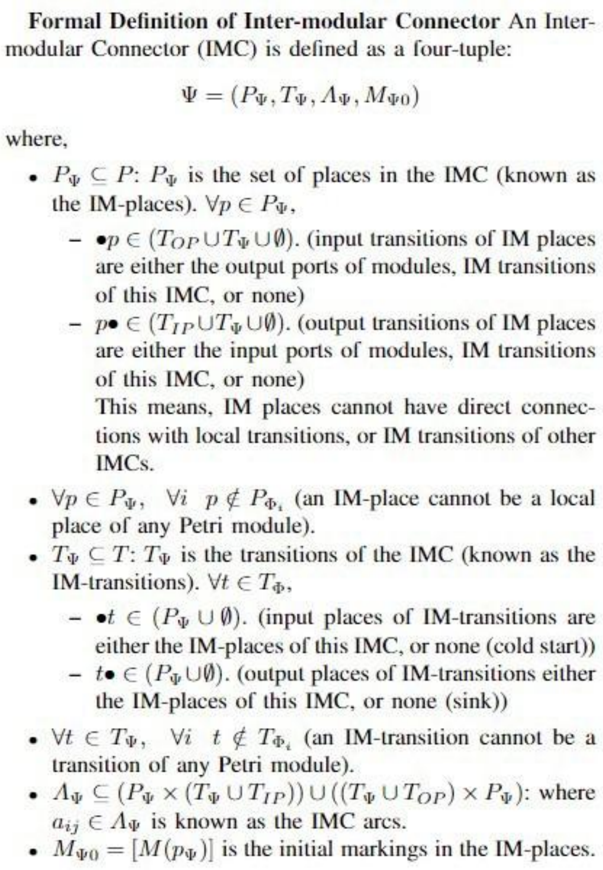
$$\Phi = (P_{L\Phi}, T_{IP\Phi}, T_{L\Phi}, T_{OP\Phi}, \Lambda_{\Phi}, M_{\Phi 0}),$$

where,

- $T_{IP\Phi} \subseteq T$: $T_{IP\Phi}$ is known as the input ports of the module.
- $T_{L\Phi} \subseteq T$: $T_{L\Phi}$ is known as the local transitions of the module.
- $T_{OP\Phi} \subseteq T$: $T_{OP\Phi}$ is known as the output ports of the module.
- $T_{IP\Phi}$, $T_{L\Phi}$, and $T_{OP\Phi}$, are all mutually exclusive: $T_{IP\Phi} \cap T_{L\Phi} = T_{L\Phi} \cap T_{OP\Phi} = T_{OP\Phi} \cap T_{IP\Phi} = \emptyset$.
- $T_{\Phi} = T_{IP\Phi} \cup T_{L\Phi} \cup T_{OP\Phi}$ (the transitions of the module).
- $P_{L\Phi} \subseteq P$ is known as the set of local places of the module. 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 (none means a local place can be a source, without any input transition).
  - $p \bullet \in (T_{\Phi} \cup \emptyset)$. Output transitions of local places are either the transitions of the module or none (none means a local place can be a sink, without any output transition).

    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 (none here means that a local transition can be a cold start (a source), without any input places).
  - $t \bullet \in (P_{L\Phi} \cup \emptyset)$. Output places of local transitions either the local places or none (none means the local transition is a sink, without any output places).
- $\forall t \in T_{IP\Phi}$
  - $\bullet t \in (P_{L\Phi} \cup P_{IM} \cup \emptyset)$. (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.
- $\Lambda_{\Phi} \subseteq (P_L \times T_{\Phi}) \cup (T_{\Phi} \times P_L)$: where $a_{ij} \in \Lambda_{\Phi}$ is known as the internal arcs of the module.
- $M_{\Phi 0} = [M(p_L)]$ is the initial markings in the local places.

**Figure 2.** Definition: A Petri Module [8].



**Formal Definition of Inter-modular Connector** An Inter-modular Connector (IMC) is defined as a four-tuple:

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

where,

- $P_\Psi \subseteq P$: $P_\Psi$ 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.
- $\forall p \in P_\Psi, \quad \forall i \quad p \notin P_{\Phi_i}$ (an IM-place cannot be a local place of any Petri module).
- $T_\Psi \subseteq T$: $T_\Psi$ 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))
- $\forall t \in T_\Psi, \quad \forall i \quad t \notin T_{\Phi_i}$ (an IM-transition cannot be a transition of any Petri module).
- $\Lambda_\Psi \subseteq (P_\Psi \times (T_\Psi \cup T_{IP})) \cup ((T_\Psi \cup T_{OP}) \times P_\Psi)$: where $a_{ij} \in \Lambda_\Psi$ is known as the IMC arcs.
- $M_{\Psi 0} = [M(p_\Psi)]$ is the initial markings in the IM-places.

**Figure 3.** Definition: An Inter-Modular Connector [8].

### 5.1. *"The Theory of Connection" Based Design*

GPenSIM was designed based on the **Theory of Connection**. The theory of connection is a concept that appears in different fields (e.g., psychology, sociology, network theory, etc.) with varying interpretations. However, in the context of **systems theory**, the theory of connection (*a.k.a.* the principle of interconnectedness) refers to the fundamental idea that **elements of a system are interdependent**, and their relationships determine the system's behavior. According to **the theory of connection in systems theory**, systems are not merely col- lections of parts but **networks of dynamic relationships**. The connections (flows, feedback loops, dependencies) between elements are **as critical as** the elements themselves. The concept is **interdependence** - no element operates in isolation as changes in one element ripple through the system. Also, the connections (**loops**) create reinforcing, balancing, or stabilizing effects. Additionally, the theory of connection emphasizes **Network Structure**, meaning the pattern of connections (centralized, decentralized, modular) determines adaptability and that systems are defined by their connections to/from the environment (their **Boundaries and Openness**). The theoretical foundations for the theory of connection were laid by Ludwig von Bertalanffy [37,38], Jay Forrester [39,40], and Fritjof Capra [41,42]. However, GPenSIM was designed following the guidelines provided by the legendary Norwegian professor Øyvind Bjørke in his book on "Manufacturing Systems Theory" [43], and in some follow-up works [44–46].

### 5.2. *GPenSIM Processors*

Since GPenSIM runs on the MATLAB platform, it allows seamless use of MATLAB's computational power (matrix operations, advanced statistics, control systems, AI/ML/DL toolboxes) to be integrated with Petri nets. Since MATLAB provides hundreds of functions in virtually any branch of engineering, making Petri net models adapted to different domains becomes easy. Also, after

Petri net simulations, modelers can leverage MATLAB's plotting and data analysis tools for results. At a minimum, implementing a Petri net model with GPenSIM involves four M-files [47,48]:

1. Petri Net Definition File (PDF): This file defines the static Petri net graph (places, transitions, and the arcs). This file will be executed only once, for creating a MATLAB structure representing the topological information of this file.
2. The Main Simulation File (MSF): This file declares the initial dynamics (e.g., initial tokens, firing times of transitions) and starts the simulations. Once the simulation is started, the GPenSIM compiler takes over to run the simulation. When the simulations are complete, control is passed back to this file so that the simulation results can be analyzed and plotted.
3. Pre-processors: Whenever transitions are enabled, pre-processors are executed. Only if the corresponding pre-processors return logical one, the transition will be allowed to start firing.
4. Post-processors: Whenever transitions complete firing, post-processors are executed. This means that whenever a transition starts firing and completes firing, there will be a processor file executed by the compiler. Hence, it is the processor files (pre-processors and post-processors) that work as the interface, both intra-model (connecting the elements of the model) and between the model and the external world.

### *5.3. Integration with MATLAB and the External World*

Figure 4 shows a small monolithic (non-modular) Petri net, consisting of four places (**p1** to **p4**) and three transitions (**t1** to **t3**). In GPenSIM, the interface for **monolithic Petri net** models is different from that of **modular Petri nets**. For a monolithic Petri net, there are two groups of processors: **specific processors** and **common processors**. For example, whenever **t1** is enabled, its specific pre-processor t1_pre will be executed if it exists. Also, the common pre-processor COMMON_PRE will be executed if it exists too. Figure 5 shows the processors for the monolithic Petri net shown in Figure 4.

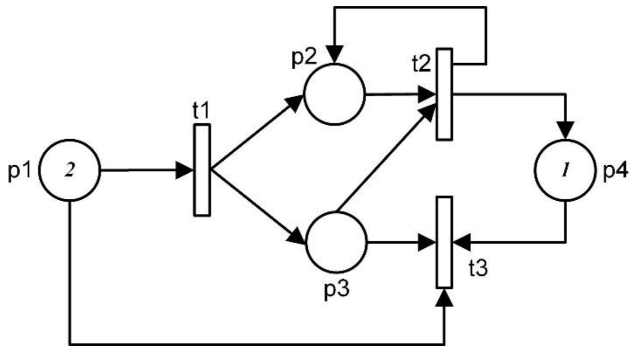


**Figure 4.** A monolithic Petri net.

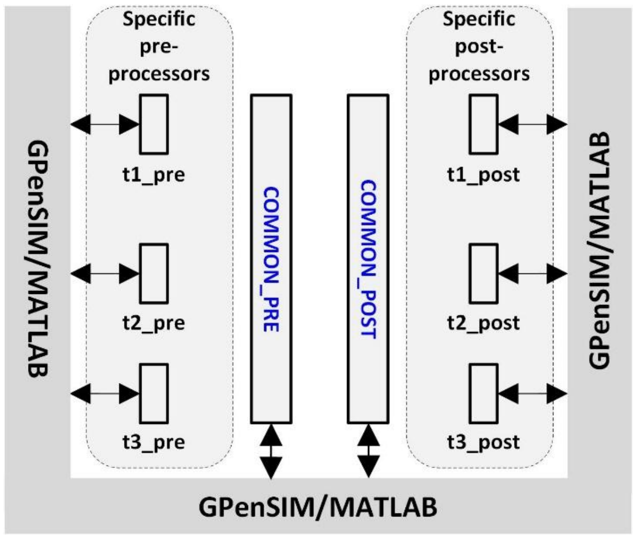


**Figure 5.** The interface for a monolithic Petri net.

Figure 6 shows a small modular Petri net, consisting of two “Petri Modules,” Module-Alfa 212 and Module-Beta, and two “Inter-Modular Connectors,” IMC-X and IMC-Y. For modular Petri nets, there are three groups of processors - in addition to specific processors and common processors, there are also **modular processors**. For example, for the modular Petri net shown in Figure 6, whenever **tAI1** (an Input Port), **tAO1** (an Output Port), or **tAL1** (a local transition) is enabled, its modular pre-processor MOD_Alfa_PRE will be executed if it exists. Figure 7 shows the processors for the modular Petri net shown in Figure 6.

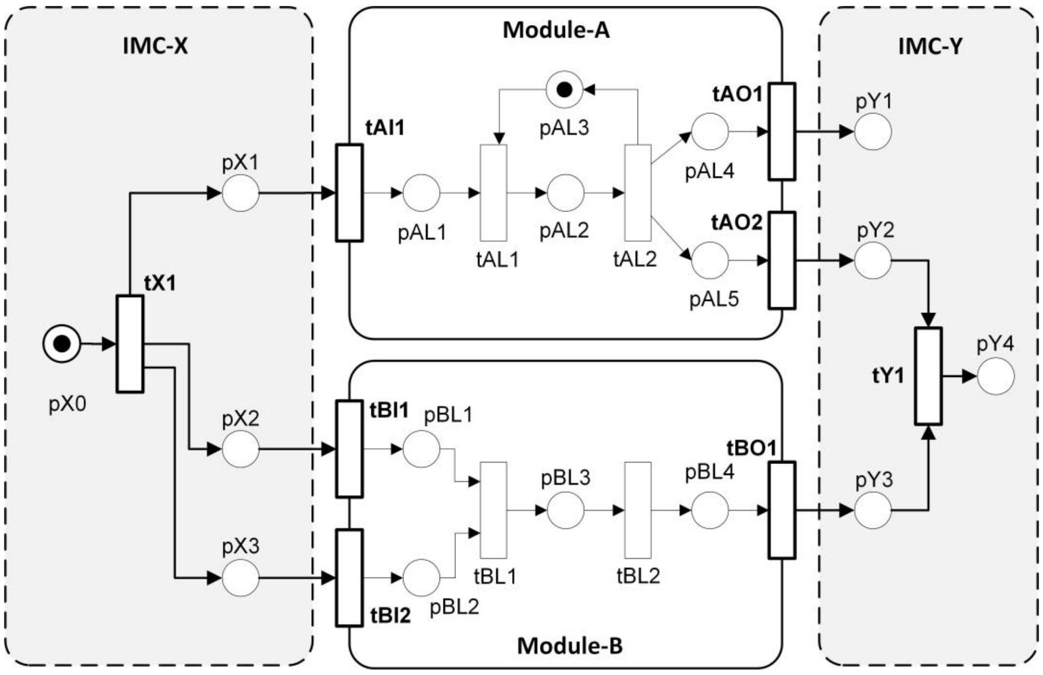


**Figure 6.** A modular Petri net.

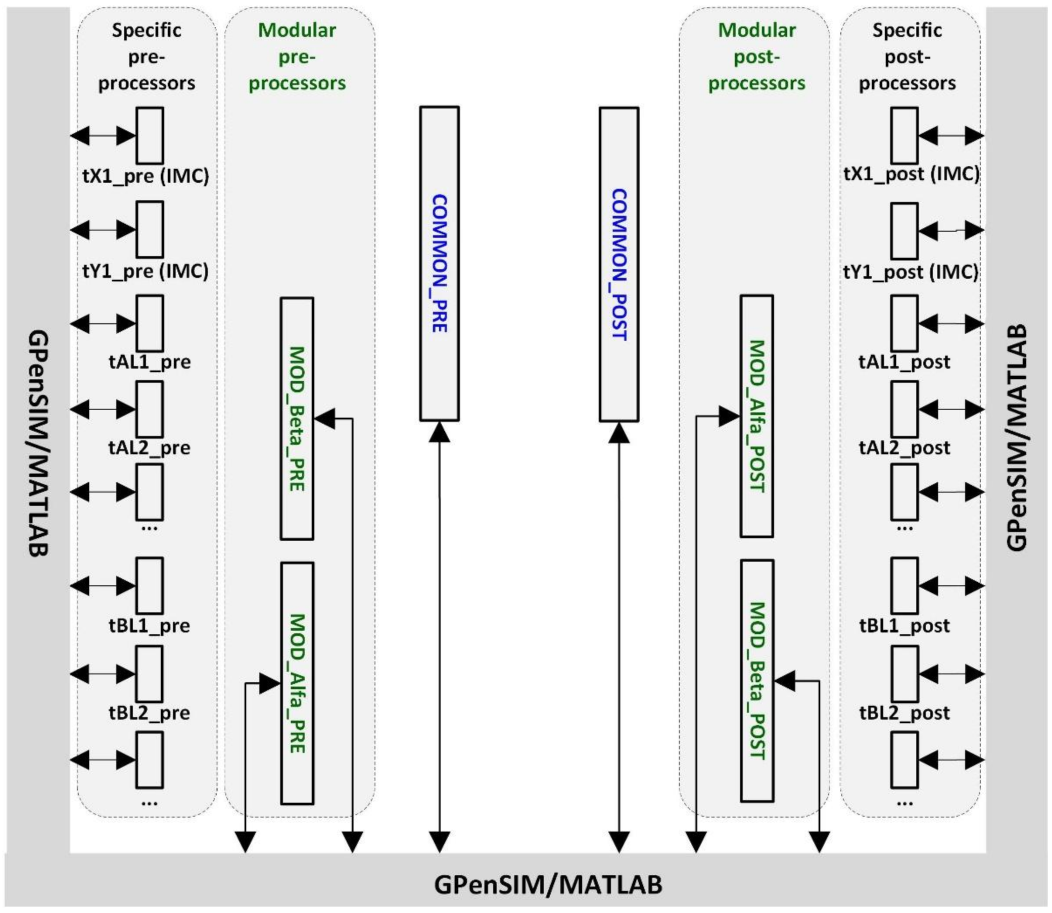


**Figure 7.** The interface for a modular Petri net.

In addition to functioning as a standalone offline Petri net simulator, GPenSIM is also designed to function as a real-time Petri net model-based controller. Literature study reveals that some works have used GPenSIM not only as a Petri net simulator but also as a controller [22,35,49]. Figure 8 shows the use of processors as the interface when Petri net models are used to interact with and control external software and hardware.

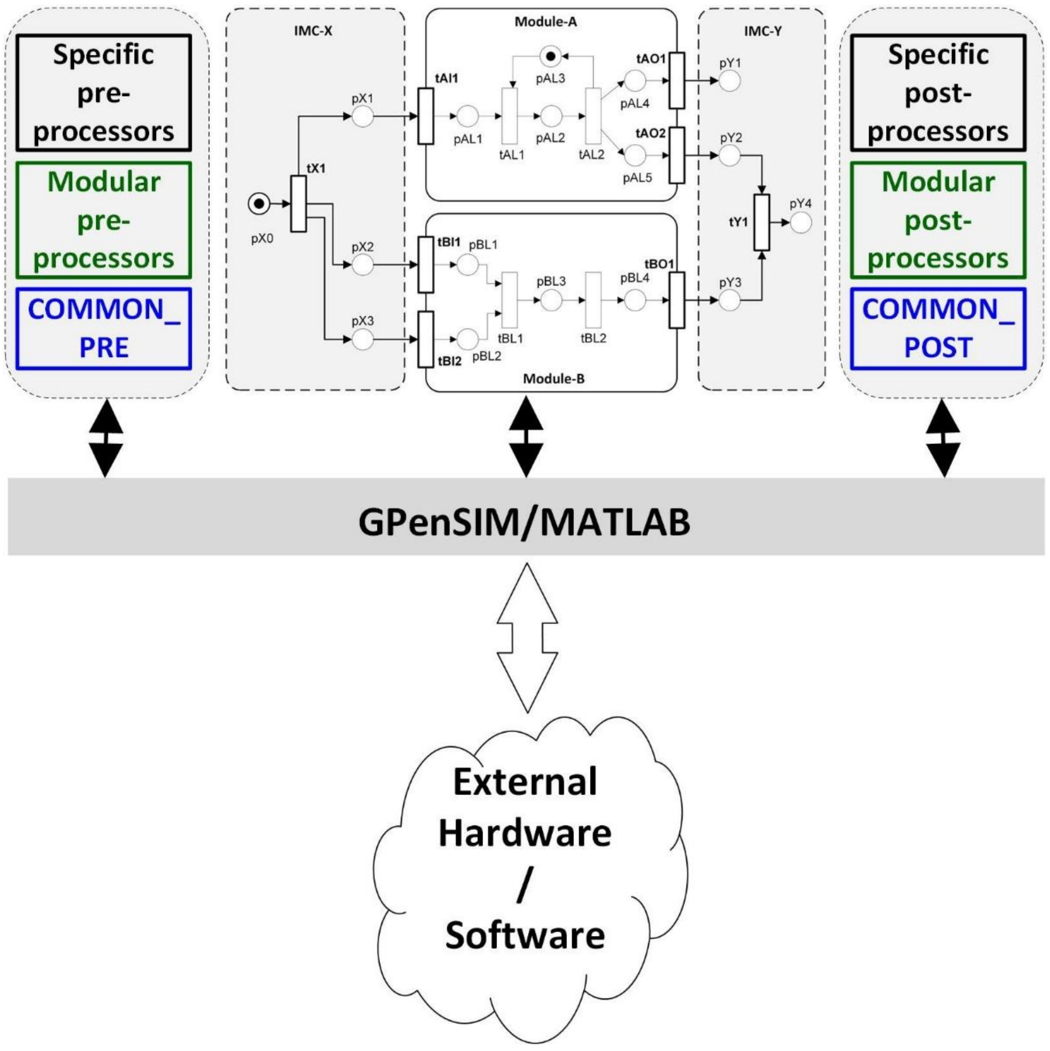


**Figure 8.** Processors as interface for Petri net model-based control of external software and hardware.

## 6. Discussion: Enhancing Supervisory Control with GPenSIM

GPenSIM is grounded in the systems theory principle of interconnectedness, which posits that system behavior emerges from dynamic relationships between components rather than from isolated elements. This principle, rooted in the works of Bertalanffy [37], Forrester [50], and Capra [41], emphasizes: **Interdependence**: Transitions, places, and tokens interact via feedback loops and dependencies, mirroring real-world discrete-event systems.

- **Network Structure**: Modular design patterns (centralized vs. decentralized) deter- mine adaptability, aligning with supervisory control challenges in distributed systems (Section 2.2).
- **Boundary Dynamics**: GPenSIM's interface - the processors (*pre/post-processors*) - mod- 234 els interactions between the Petri net and its environment, addressing partial observ- 235 ability (Section 2.5) and unmodeled disturbances (Section 2.6). In the following subsection, we shall see how GPenSIM can help solve some of the difficulties we face in supervisory control theory.

### 6.1. Reducing State-Space Explosion: Moving Control Logic from Controller to Processors

GPenSIM offers a unique solution to mitigate the state-space explosion problem. As we know, the number of states increases exponentially with the number of places, transitions, and tokens. Hence, embedding a supervisor (a controller with multiple control places and their tokens) on top of a plant to create a controlled system will dramatically increase the total number of states. To address this, GPenSIM provides a unique solution: mapping the controller logic into processors, so that the controlled system consists of the plant and the processors without additional state overhead.

For example, Figure 9 shows a simple plant with three places, four transitions, and no initial tokens. Consider the following three constraints:

1. The number of tokens in **p1** must always be less than or equal to 10.
2. The number of tokens in **p2** must always be less than or equal to 5.
3. The difference between the tokens in **p1** and **p2** must always be less than or equal to 2.

Figure 10 shows the controlled system synthesized using supervisory control theory, assuming all transitions are observable and controllable. The resulting supervisor includes three control places and a total of 17 tokens, which will inevitably cause the state-space of the controlled system to explode.

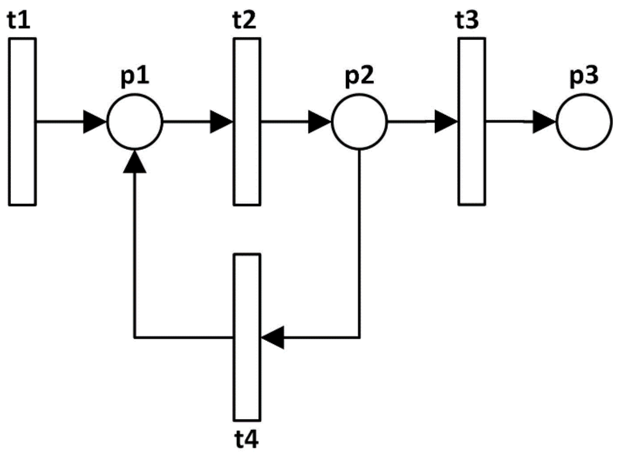


**Figure 9.** The Plant.

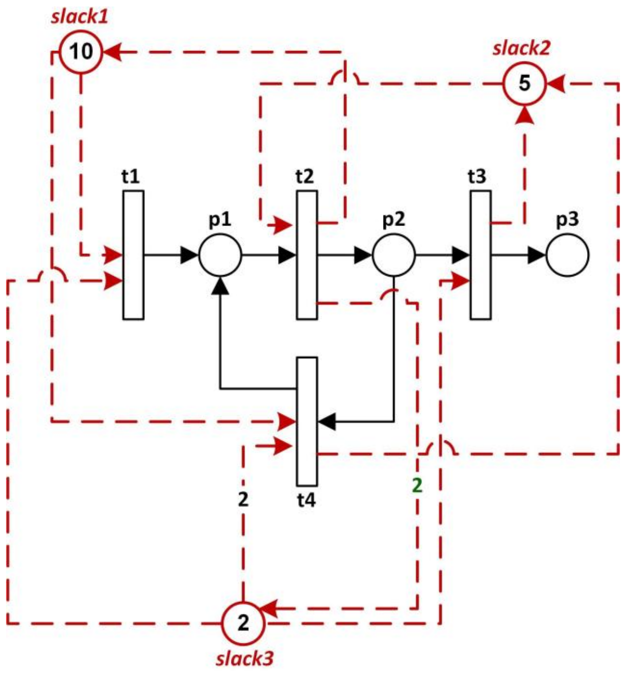


**Figure 10.** The Controlled System.

GPenSIM’s built-in function control2processors maps the supervisor into program code, automatically generating pre-processors and post-processors. This moves the super- visor out of the controlled system, leaving the controlled system as the original Petri net epresenting the plant. Figure 11 shows the controlled system with processors.

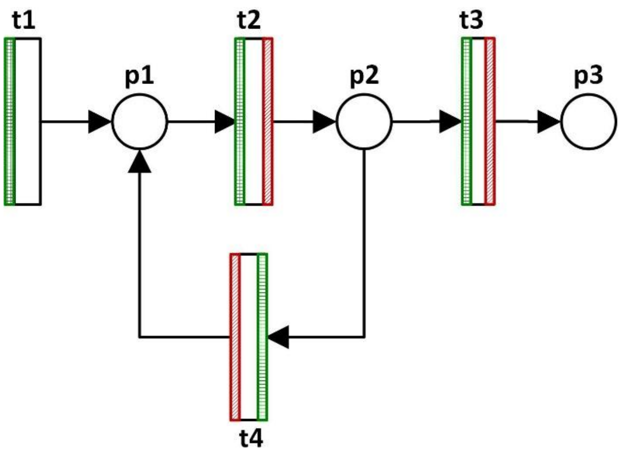


**Figure 11.** The Controlled System with Processors.

Listing 1 shows one of the pre-processors, t2_pre, automatically generated by GPen- SIM's control2processors function. Listing 2 shows one of the post-processors, t3_post.

**Listing 1:** Specific pre-processor "t2_pre".

```
function [fire, transition] = t2_pre(transition)
% Pre-processor for transition "t2"
%
% this file ("t2_pre.m") is automatically generated
%    by the function "writePreProcessors"


global global_info  % slacks are attached to global_info

fire = 1; % initial assumption
fire = and(fire, global_info.slack2);

% if the transition is allowed to fire
% then reduce the slack variables
%    (remove "tokens" from the slack "places")
if fire
     global_info.slack2 = global_info.slack2 - 1;
end
```

**Listing 2:** Specific post-processor "t3_post".

```
function [] = t3_post(transition)
% Post-processor for transition "t3"
%
% this file ("t3_post.m") is automatically generated
%   by the function "writePostProcessors"

global global_info  % slacks are attached to global_info

% after firing of the transition
% increase the slack variables
%    (add "tokens" to the slack "places")
global_info.slack2 = global_info.slack2 + 1;
```

In addition to avoiding state-space explosion, GPenSIM's approach of moving the supervisor's control logic into processor files offers another advantage. As shown in Figure 12, the controlled system can be implemented in real life as a real-time controller for a physical (real-world) plant. Thus, GPenSIM provides a practical solution for developing industrial supervisory controllers.

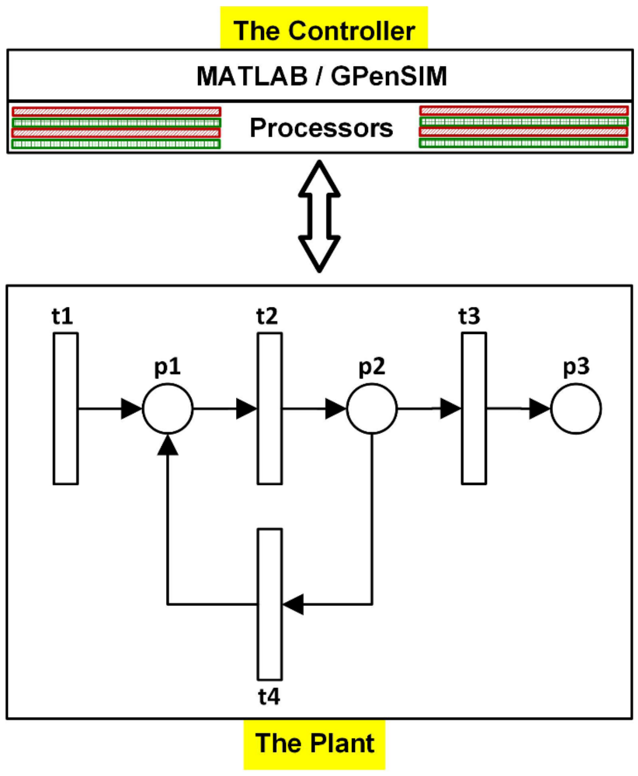


**Figure 12.** GPenSIM Approach for realizing Industrial Controllers.

## *6.2. Increasing Scalability in Distributed Systems*

GPenSIM supports modular Petri nets, as defined in [8]. This enables the development of multiple supervisors, each controlling a decentralized plant. These supervisors are implemented as "Petri Modules" that communicate over a shared bus. While operating independently, the distributed supervisors can also collaborate to make global decisions, requiring only minimal coordination between them. Figure 13 illustrates this architecture, showing a set of supervisors managing a decentralized plant.

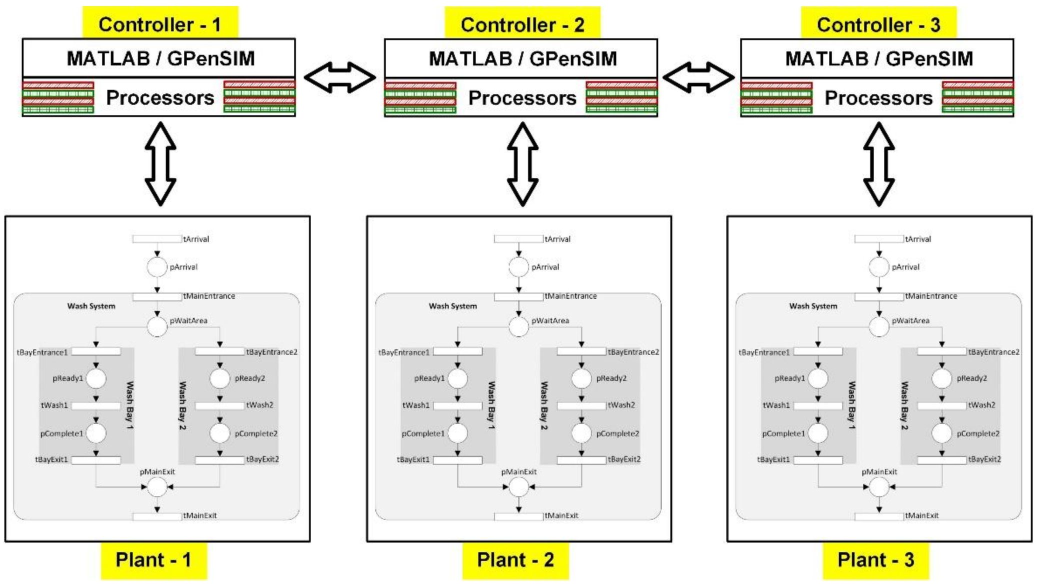


**Figure 13.** GPenSIM's Approach for Realizing Distributed Controllers.

GPenSIM's modular architecture directly tackles the state-space explosion problem (Section 2.1) by:

- Decomposition: Partitioning monolithic Petri nets into weakly coupled Petri Modules (Figure 6), enabling parallel execution on multi-core systems. This reduces computational complexity from O($n^k$) to O(n) for laminar-flow systems [8].
- Hierarchical Abstraction: Modules can operate at varying granularity levels, trading off detail for scalability—a key consideration in industrial automation [15].

### *6.3. Enhanced Expressiveness via Programmable Interfaces*

Classical Petri nets lack support for nonlinear or temporal constraints (Section 2.3). Classical Petri nets lack support for nonlinear or temporal constraints (Section 2.3). GPenSIM addresses this through programmable processors, enabling:

- Nonlinear constraints (e.g., quadratic, exponential) via MATLAB expressions.
- Temporal/conditional logic (e.g., disabling transitions dynamically).

Example: Enforcing Nonlinear and Temporal Constraints Consider a plant with places p1 and p2. We enforce:

1. Quadratic constraint: ($p1^2 + p2 \leq 100$).
2. Temporal constraint: Transition t1 is disabled for 60 seconds if p1 exceeds a threshold.

**GPenSIM Implementation (version 11):**

1. Pre-processor for t1 ('t1_pre.m') enforces the quadratic constraint and temporal block-ing:

```
function [fire, transition] = t1_pre(transition)
global global_info  % Access Petri net markings
```

```
% Nonlinear constraint: p1^2 + p2 ≤ 100
marking_p1 = current_marking('p1');  % Get current tokens in p1
marking_p2 = current_marking('p2');
fire = le(marking p1^2 + marking p2, 100);  % Block if violated

% Temporal constraint: Disable t1 for 60s if p1 > 15
if gt(marking_p1, 15)
    global_info.t1_blocked_until = current_time() + 60;  % Set block end time
    fire = 0;  % Force-disable transition
elseif isfield(global_info, 't1_blocked_until') && ...
       lt(current_time(), global_info.t1_blocked_until)
    fire = 0;  % Keep disabled until time elapses
end
```

2. Post-processor for t1 (‘t1_post.m‘) logs violations:

```
function [] = t1_post(transition)
global global_info
marking_p1 = current_marking('p1');
marking_p2 = current_marking('p2');

if gt(marking_p1^2 + marking_p2, 100)
    disp(['Warning: Nonlinear constraint violated at t=', ...
        num2str(current_time())]);
end
```

Key features demonstrated in the above code:

- **Nonlinear arithmetic**: Direct use of MATLAB’s math operators (‘^‘, ‘+‘).
- **Dynamic state checks**: GPenSIM’s built-in function‘current_marking’ queries real time token counts.
- **Temporal control**: GPenSIM’s Global variables (‘global_info‘) track blocking dura-tions.
- **Integration**: Leverages MATLAB’s native functions (e.g., ‘disp’) and GPenSIM’s built-in functions (e.g., ‘current_time‘).

GPenSIM’s approach extends classical Petri nets to hybrid systems without modifying the net structure, addressing the rigidity noted in Section 2.3.

### *6.4. Mitigating Rigid Control via Adaptive Processors*

As noted in Section 2.4, classical supervisory controllers often disable **all** transitions leading to illegal states, even when some paths could safely proceed. GPenSIM’s processors enable **adaptive control** by:

1. **Conditional enabling** of transitions based on real-time system state.
2. **Partial permissions** (e.g., allowing transitions only under specific token distributions).

Example: Relaxing Overly Conservative Control Consider a manufacturing system where:

1. **Plant**: Places **p1** (raw materials), **p2** (workspace), **p3** (finished goods).

2. **Constraint**: Avoid workspace overflow ('**p2** ≤ 5'), but allow temporary violations if **p3** is nearly empty (to maintain throughput).

**GPenSIM Implementation (version 11):**

1. **Pre-processor for t_work ('**t_work_pre.m**')** dynamically adjusts enforcement:

```
function [fire, transition] = t_work_pre(transition)
global global_info
```

```
marking_p2 = current_marking('p2');
marking_p3 = current_marking('p3');

% Default constraint: p2 ≤ 5
if le(marking_p2, 5)
    fire = 1;  % Always allow if workspace is safe
else
    % Relax constraint if finished goods are critically low (p3 < 2)
    fire = lt(marking_p3, 2);  % Allow temporary overflow only if p3 is low
    if fire
        disp(['Warning: Allowing workspace overflow (p2=', ...
            num2str(marking_p2), ...
              ') to maintain production at t=', num2str(current_time())]);
    end
end
```

2. Post-processor for t_work ('t_work_post.m') logs relaxations:

```
function [] = t_work_post(transition)
global global_info
marking_p2 = current_marking('p2');
if gt(marking_p2, 5)
    log_file('relaxations.log', ...  % GPenSIM's logging function
             ['p2 overflow tolerated at t=', num2str(current_time()), ...
              ' (p2=', num2str(marking_p2), ', p3=', ...
                   num2str(current_marking('p3')), ')']);
end
```

Key features demonstrated in the code above:

- **Context-aware firing**: Transitions are blocked only when strictly necessary (e.g., '**p2** > 5' and '**p3** ≥ 2').
- **Real-time state checks**: Uses GPenSIM's built-in function'current_marking' to evalu-
- ate tokens dynamically.
- **Logging**: Trades off rigor for throughput while maintaining auditability.
- **GPenSIM integration**: Leverages GPenSIM's built-in functions like 'log_file' for traceability.
- **Theoretical Alignment**: This approach aligns with Ghaffari et al.'s *maximally permissive control* [17] by minimizing unnecessary restrictions, while GPenSIM's programmability avoids the complexity of formal region theory.

## 7. Conclusions

Supervisory control theory based on Petri nets provides a robust framework for discrete-event systems but faces persistent challenges in scalability, expressiveness, and practical implementation. This paper presented **GPenSIM**, a MATLAB-based modular Petri net tool, as a versatile solution to these limitations; GPenSIM is developed by the first author of this paper. By leveraging **modular decomposition**, *programmable interfaces* known as the **processors**, and **integration with MATLAB**, GPenSIM addresses three core challenges identified in Section 2, such as **State-Space Explosion** (Section 2.1), **Rigid Enforcement** (Section 2.4), and **Limited Expressiveness** (Section 2.3). Broader Implications of this paper:

- **Industrial Applicability**: GPenSIM bridges the gap between theoretical models and real-world deployment. Its processors interface with external hardware/software (Figure 8), enabling real-time control synthesis (e.g., Kaid et al.'s work [22]).
- **Scalability**: *Petri Modules* (Section 4) distribute control across decentralized systems (Section 6.2), minimizing synchronization delays via *Inter-Modular Connectors* (Figure 1).
- **Theoretical Grounding**: GPenSIM embodies the *Theory of Connection* (Section 5.1), emphasizing dynamic interdependencies between system components - a principle rooted in Bertalanffy's systems theory [37].

**Limitations and Future Work**: While GPenSIM advances supervisory control theory, challenges remain:

- **Abstraction Trade-offs**: Modular decomposition requires laminar-flow models (Section 2.1), limiting applicability to highly interconnected systems.
- **Verification**: GPenSIM lacks native formal verification tools, though bridges to model checkers like NuSMV [51] offer partial solutions.

Future research could explore:

- **Hybrid Systems**: Integrating continuous dynamics (e.g., via MATLAB's Control Toolbox).
- **AI/ML Enhancements**: Adaptive processors trained on historical data to optimize constraints dynamically.

**Final Remarks**: GPenSIM redefines supervisory control by prioritizing **practicality**, **scalability**, and **expressiveness**. Its modular architecture and MATLAB integration provide a unified platform for academia and industry, as demonstrated in manufacturing and distributed control case studies (Section 6.4). By addressing the limitations of classical supervisory control theory, GPenSIM not only extends the theoretical boundaries of Petri nets but also delivers a *practical toolkit* for next-generation discrete-event systems.

**Author Contributions:** "Conceptualization, R.D., S.T, and Y.F.; methodology, R.D.; software, R.D.; validation, R.D., S.T. and Y.F.; formal analysis, R.D., S.T. and Y.F.; investigation, S.T.; resources, Y.F.; writing—original draft preparation, R.D., S.T. and Y.F.; writing—review and editing, R.D., S.T. and Y.F.; project administration, Y.F.; funding acquisition, Y.F. All authors have read and agreed to the published version of the manuscript."

**Funding:** This work is sponsored by the Natural Science Foundation of Chongqing, P.R. China (2024NSCQ-LZX0121, 2024NSCQ-LZX0120, 2023TIAD-ZXX0017, CSTB2023NSCQ-LZX0135), the Scientific and Technological Research Program of Chongqing Municipal Education Commission, P.R China (KJZD-K202301023), the Scientific and Technological Research Program of Wanzhou District, P.R. China (WZSTC-20230309), the National Natural Science Foundation of China (12201086), and Program of Chongqing Municipal Education Commission, P.R. China (KJQN202201209, 233356).

**Conflicts of Interest:** The authors declare no conflicts of interest. Also, the funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

## References

1. Ramadge, P.J.; Wonham, W.M. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization* **1987**, *25*, 206–230.

2. Giua, A.; DiCesare, F. Petri nets and supervisory control. *Discrete Event Dynamic Systems* **1993**, *3*, 151–168.
3. Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* **1989**, *77*, 541–580.
4. Zhou, M.; DiCesare, F. Petri net synthesis for discrete event control of manufacturing systems. *IEEE Transactions on Robotics and Automation* **1993**, *9*, 62–75.
5. Moody, J.O.; Antsaklis, P.J. *Supervisory control of discrete event systems using Petri nets*; Springer, 1998.
6. Basile, F.; Chiacchio, P.; De Tommasi, G. Supervisory control of Petri nets: A survey. *Annual Reviews in Control* **2020**, *49*, 241–257.
7. Peterson, J.L. *Petri net theory and the modeling of systems*; Prentice Hall PTR, 1981.
8. Davidrajuh, R. *Petri Nets for Modeling of Large Discrete Systems*; Springer, 2021.
9. Jensen, K. *Coloured Petri nets: Basic concepts, analysis methods and practical use*; Vol. 1, Springer, 1997.
10. Wang, H.; Zhou, M. Timed Petri nets for performance analysis of automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **2021**, *51*, 1309–1320.
11. Sheridan, T.b. *Monitoring behavior and supervisory control*; Vol. 1, Springer Science & Business Media, 2013.
12. Iordache, M.; Antsaklis, P.J. *Supervisory control of concurrent systems: a Petri net structural approach*; Springer Science & Business Media, 2007.
13. Kaid, H.; Al-Ahmari, A.; Li, Z.; Davidrajuh, R. processes MDPI. *Optimization for Control, Observation and Safety* **2020**, p. 387.
14. Davidrajuh, R. Literature Review on Modular Petri Nets. *Petri Nets for Modeling of Large Discrete Systems* **2021**, pp. 77–85.
15. Choi, J.Y.; Reveliotis, S.A. A generalized stochastic Petri net model for performance analysis and control of capacitated reentrant lines. *Robotics & Automation IEEE Transactions on* **2003**, *19*, 474–480.
16. Yoo, T.S.; Lafortune, S. Decentralized supervisory control with conditional decisions: Supervisor existence. *IEEE Transactions on Automatic Control* **2002**, *47*, 1886–1890.
17. Ghaffari, A.; Rezg, N.; Xie, X. Design of a live and maximally permissive Petri net controller using the theory of regions. *IEEE Transactions on Robotics and Automation* **2003**, *19*, 137–142.
18. Yang, J.; Reed, S.; Yang, M.H.; Lee, H. Weakly-supervised Disentangling with Recurrent Transformations for 3D View Synthesis. In Proceedings of the International Conference on Neural Information Processing Systems-volume, 2016.
19. Herzig, A.; Lang, J.; Longin, D.; Polacsek, T. A Logic for Planning under Partial Observability. In Proceedings of the Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA., 2000.
20. Ru, Y.; Hadjicostis, C.N. Obfuscation-based partial observability in supervisory control. *IEEE Transactions on Automatic Control* **2009**, *54*, 1551–1565.
21. Basile, F.; Chiacchio, P.; De Tommasi, G. Fault-tolerant supervisory control of Petri nets. *IEEE Transactions on Automatic Control* **2015**, *60*, 810–815. 504
22. Kaid, H.; Al-Ahmari, A.; Li, Z.; Davidrajuh, R. Automatic supervisory controller for deadlock control in reconfigurable manufacturing systems with dynamic changes. *Applied Sciences* **2020**, *10*, 5270.
23. Al-Ahmari, A.; Kaid, H.; Li, Z.; Davidrajuh, R. Strict minimal siphon-based colored Petri net supervisor synthesis for automated manufacturing systems with unreliable resources. *IEEE Access* **2020**, *8*, 22411–22424.
24. Reisig, W. *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*; Springer: Berlin, 2013. Modern treatment with formal methods, industrial examples, and tool support., https://doi.org/10.1007/978-3-642-33278-4.
25. Desrochers, A.A.; Al-Jaar, R.Y. *Applications of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis*; IEEE Press: New York, 1995. Focus on manufacturing, automation, and real-time systems.
26. Desel, J.; Reisig, W. The concepts of Petri nets. *Software & Systems Modeling* **2015**, *14*, 669–683.
27. Liu, G. Time Petri Nets and Time-Soundness. In *Petri Nets: Theoretical Models and Analysis Methods for Concurrent Systems*; Springer, 2022; pp. 203–236.

28. Kindler, E.; Petrucci, L. Towards a standard for modular Petri nets: A formalisation. In Proceedings of the Applications and Theory of Petri Nets: 30th International Conference, PETRI NETS 2009, Paris, France, June 22-26, 2009. Proceedings 30. Springer, 2009, pp. 43–62.
29. Righini, G. Modular Petri nets for simulation of flexible production systems. *International Journal of Production Research* **1993**, *31*, 2463–2477. 520
30. Lee, W.J.; Cha, S.D.; Kwon, Y.R. Integration and analysis of use cases using modular Petri nets in requirements engineering. *IEEE Transactions on software engineering* **1998**, *24*, 1115–1130.
31. Latorre-Biel, J.I.; Jiménez-Macías, E.; García-Alcaraz, J.L.; Muro, J.C.S.D.; Blanco-Fernandez, J.; Parte, M.P.D.L. Modular construction of compact Petri net models. *International Journal of Simulation and Process Modelling* **2017**, *12*, 515–524.
32. Wang, C.H.; Wang, F.J. An object-oriented modular Petri nets for modeling service oriented applications. In Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007). IEEE, 2007, Vol. 2, pp. 479–486.
33. Lakos, C.; Petrucci, L. Modular state spaces for prioritised Petri nets. In Proceedings of the Monterey Workshop. Springer, 2010, pp. 136–156.
34. Davidrajuh, R. A New Modular Petri Net for Modeling Large Discrete-Event Systems: A Proposal Based on the Literature Study. *Computers* **2019**, *8*, 83.
35. Cameron, A.; Stumptner, M.; Nandagopal, N.; Mayer, W.; Mansell, T. Rule-based peer-to-peer framework for decentralized real-time service oriented architectures. *Science of Computer Programming* **2015**, *97*, 202 – 234. https://doi.org/10.1016/j.scico.2014.06.005.
36. Mutarraf, U.; Barkaoui, K.; Li, Z.; Wu, N.; Qu, T. Transformation of Business Process Model and Notation models onto Petri nets and their analysis. *Advances in Mechanical Engineering* **2018**, *10*, 1 – 21. https://doi.org/10.1177/1687814018808170.
37. Bertalanffy, L.v. *General System Theory: Foundations, Development, Applications*; George Braziller: New York, 1968.
38. Bertalanffy, L.v. The Theory of Open Systems in Physics and Biology. *Science* **1950**, *111*, 23–29. https://doi.org/10.1126/science.111.2872.23.
39. Forrester, J.W. *Industrial Dynamics*; MIT Press: Cambridge, MA, 1961.
40. Forrester, J.W. World Dynamics. *Studies in the Life Sciences* **1971**.
41. Capra, F. *The Web of Life: A New Scientific Understanding of Living Systems*; Anchor Books: New York, 1996.
42. Capra, F. *The Turning Point: Science, Society, and the Rising Culture*; Simon & Schuster, 1982.
43. Bjorke, O. Manufacturing Systems Theory. *Tapir Academic Press* **1995**.
44. Davidrajuh, R. Planning e-government start-up: a case study on e-Sri Lanka. *Electronic Government, an International Journal* **2004**, *1*, 92–106.
45. Davidrajuh, R. Automating supplier selection procedures. *PhD Thesis* **2001**.
46. Davidrajuh, R. Array-based logic for realising inference engine in mobile applications. *International Journal of Mobile Learning and Organisation* **2007**, *1*, 41–57.
47. Davidrajuh, R. *Modeling Discrete-Event Systems with GPenSIM*; Springer International Publishing: Cham, 2018. https://doi.org/10.1007/978-3-319-73102-5.
48. Davidrajuh, R. *GPenSIM Reference Manual*; University of Stavanger, 2024.
49. Feng$^1$, Y.; Samolej, S.; Davidrajuh, R. Application Interface for GPenSIM. In Proceedings of the Systems Modelling and Simulation: First International Symposium, SMS 2024, Johor Bahru, Malaysia, December 16–17, 2024, Proceedings. Springer Nature, 2025, Vol. 2483, p. 237.
50. Forrester, J.W. Principles of Systems. *MIT Press* **1968**. Later expanded into the book "Principles of Systems" (1968).
51. Roci, A.; Davidrajuh, R. Developing a Bridge from GPenSIM to NuSMV for Model Checking. *European Modeling & Simulation Symposium* **2021**.

disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.