©2011 IEEE. Reprinted, with permission, from : Reggie Davidrajuh; Representing Resources in Petri Net Models : Hardwiring or Soft-coding?, *2011 IEEE International Conference on Service Operations, Logistics, and Informatics (SOLI)*, 2011; Beijing, China.
This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Stavanger's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org.
By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

Representing Resources in Petri Net Models: Hardwiring or Soft-coding?

Reggie Davidrajuh
Department of Electrical and Computer Engineering University of Stavanger, Stavanger, Norway
reggie.davidrajuh@uis.no

Abstract
This paper presents an interesting design problem in developing a new tool for discrete-event dynamic systems (DEDS). A new tool known as GPenSIM was developed for modeling and simulation of DEDS; GPenSIM is based on Petri Nets. The design issue this paper talks about is whether to represent resources in DEDS hardwired as a part of the Petri net structure (which is the widespread practice) or to soft code as common variables in the program code. This paper shows that soft coding resources give benefits such as simpler and skinny models.

Key words : discrete event dynamic systems, Petri net, modeling and simulation, resources in DED, GPenSIM

I. INTRODUCTION

This paper presents an interesting design issue during development of a new tool for modeling and simulation of discrete event dynamic systems (DEDS). The new tool is known as GPenSIM , General Purpose Petri net Simulator; obviously, it is based on Petri nets [3]. For the sake of completion, the next section (section II: Petri nets) presents a very short introduction to Petri nets; Section III presents the design goals of GPenSIM.

As a tool for modeling and simulation of DEDS, GPenSIM must provide robust support for resource usage, as resources are one of the primary elements of any DEDS. Since GPenSIM is based on Petri net, it was obvious to represent resources as tokens, as it was the widespread practice; however, it was found out the *hardwiring* of resources as tokens in Petri nets produces bulky Petri net models. Alternatively (and untraditionally), GPenSIM also supports representing resources as variables in programming code (*soft-coding* as opposed to hardwiring); soft-coding, as shown in section IV of this paper, brings benefits such as simpler and much smaller models, ease of programming, ease of extending or tailoring the models. etc.

II. PETRI NETS

A. Place/Transition Petri nets

Ever since its inception in 1960s, Petri nets have been used as a primary tool for modeling and simulation; this is because of Petri nets characteristics such as simple mathematical model, visual (graphical) language, yet clear and simple semantics [1]. P/T Petri net (aka Ordinary Petri net) is defined as follows [1]:

Definition 1.1: Petri net is defined by the quintuple:
PN = (P, T, A, W, Mo,)

Where:

- *P* is the set of places; places are passive elements like conveyor belts, input and output buffers, etc;
- *T* is the set of transitions; transitions are active elements like machines, humans, robots, CPUs, etc;
- *A* is the set of directed arcs; an arc connects either a place to transitions or a transition to places;
- *W* is the set of weights of the arcs, and
- *Mo* is the number of tokens initially in places.

P/T Petri nets have some limitations. One of the limitations is 'homogenous' tokens: let's say that the tokens inside a place represent resources; then in P/T Petri nets, all these resources are of the same type and cannot be differentiated. Another limitation of P/T Petri net is that it is not possible impose additional logical functions ('firing conditions') for a transition to fulfill. Colored Petri nets (CPN) [5], or better - Petri net Interpreted for Control (PIC) [4] removes these limitations.

B. Petri Net for Interpreted Control (PIC)

[4]: Petri net for interpreted Control (PIC) is defined as follows

Definition 1.2: A Petri net interpreted for the control is given by the quintuple:

PIC = (PN, Ψ, LOG, ζ, COM)

Where,

- PN is the Petri net given by the definition 1.1, consisting of a set of places, a set of transitions, a set of bipartite arcs with arc weights, and a set of initial markings;
- Ψ: T - LOG is a function mapping the transition set T onto a set of logical assertions containing logical variables, predicates, events, and the empty symbol;
- ζ: P -COM is a function mapping the set of places onto a set of value assignments to control variables including the empty variable, aid of events, the value assignments and events are realized when the place marking changes from 0 to a non-zero value.

To put it simply,

- *Ψ: T→LOG* is the set of logical conditions for firing (firing conditions), and
- *COM is a subset of places* which serves as a set of control commands; COM places (aka COM variables) are the most interesting places of a Petri net model; the other places play a secondary (supportive) role.
    - Token in a COM place means the system is instructing to send a command the outside world; or, it could also means, the outside world has instructed a command to the system via the COM variable.

III. GPENSIM

GPenSIM, developed by the author of this paper, is a new simulator for modeling and simulation of DEDS [3]. In the following subsections, a short introduction to GPenSIM is given.

A. *Why GPenSIM*

The reason for developing GPenSIM is two-folded:

- For basic users: to provide a tool that is easy to understand and easy to use, even for users with minimal mathematical and programming skills;
- For advanced users: allow seamless integration of models made with GPenSIM with the other toolboxes that are readily available on the MATLAB platform; allow easy extension of GPenSIM functions

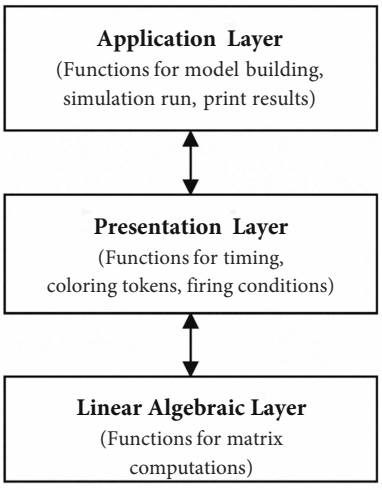

Figure 1. Layered Architecture

B. *Layerd Architecture*

GPenSIM is built following a 3-layer architecture; see Figure 1. The bottom layer deals with Petri net run-time dynamics; this layer computes newer states with the help of linear algebraic equations and matrix manipulations. The middle layer adds more high-level functionality such as stochastic timing, coloring of tokens, firing conditions ('guard-conditions' in some literature), etc. The top layer offers applications such building a Petri net based model, running simulations, determining coverability tree, printing the simulation results, etc.

C. *Modular Componets*

A model of a discrete event system developed with GPenSIM consists of a number of files. The main simulation file (MSF) is the file that will be run directly by the MATLAB engine. In addition to the main simulation file, there will be one or more Petri net definition files (PDFs); definition of a Petri net graph (static details) is given in the Petri net Definition File. There may be a number of PDFs, if the Petri net model is divided into many modules, and each module is defined in a separate PDF. While the Petri net definition file has the static details, the main simulation file contains the dynamic information (such as initial tokens in places, firing times of transitions) of the Petri net [3]. In addition to these MSF and PDF fIles, there can be a number of transition definition files (TDFs) too.

A transition definition file consists of additional conditions that determine whether an enabled transition can fire or not [2][4].

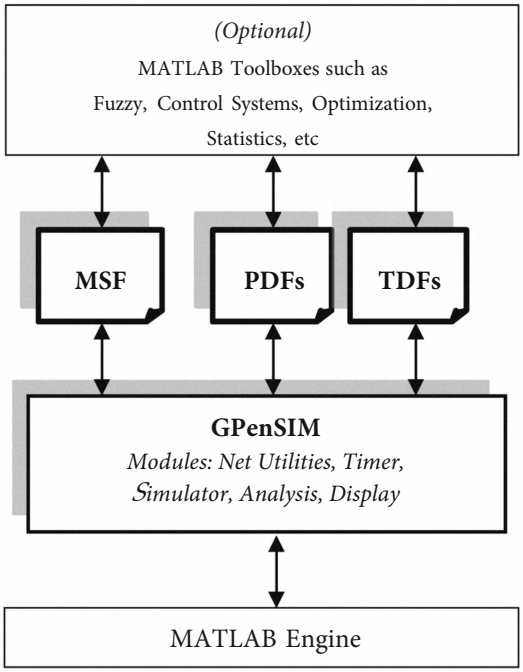

Figure 2. The files for simulation

The additional conditions are called 'firing condition' in GPenSIM terminology, whereas in some other literature (e.g.

Colored Petri Net (CPN)) it is referred to as 'guard-conditions'). There can be a separate transition definition file for each transition in a Petri net model, or a combined file.

D. Natural Language Interface

Users need not know Petri net mathematics when creating a Petri net model of a discrete event system. GPenSIM offers a natural language interface with which model building mainly deals with identifying the basic elements of a system and establishing the connections between these elements.

E. Representing Resource using GPenSIM

DEDS possess active elements such as machines, passive elements such as buffers, as well as resources. Resources (e.g. machine operators, work stations) limit utilization of systems, hence are the reasons for bottle necks in systems. In addition to resources, we need mechanisms to change the priorities (of the transitions) in order to avoid e.g. starvation and aging of competing entities [9].

As mentioned in the introduction, the scope of this paper is GPenSIM's two approaches for representing resources in Petri net models: 1) as elements (e.g. as tokens) in the Petri net structure ('hardwiring'), and, 2) as variables in program code ('soft-coding'). For the latter approach, GPenSIM provides the following functionality:

i) *Declaring resources,*

ii) *Utilizing resources* (functions for requesting (reserving), allocating, and releasing resources),

iii) *Declaring Priorities* of different transitions,

iv) Changing priorities of transitions: functions for increasing or decreasing priority of a transition, and comparing priorities of transitions, and

v) *Reporting resource usage*: new print functions that show total resource usage, idle time, etc.

The following fundamental assumption was made in realizing the additional functions for resource modeling:

A resource is a 'critical section' meaning a resource can be used by only one transition at a time; this means, resources posses 'mutual exclusion' property.

(Though a resource can be used by only one transition at a time, a transition can use as many resource as it wants, limited only by availability).

F. Summary: Methodology for Modeling and Simulation with GPenSIM

Creating a Petri net model consists of three steps: 1) Defining the static Petri net graph (in PDFs), and 2) Defining firing conditions, if any (in TDFs), and 3) Assigning initial dynamics (in MSF).

IV. CASE STUDY: A RESOURCE SHARING PROBLEM

The case study presents a very simple problem of resource sharing. This problem is adapted from Hruz and Zhou (2008) [4], whom adapted this problem further from Starke (1990) [10].

Figure 3 shows a multi-processor system in which there are three CPUs wanting to communicate between them as well as with the outside world, with the help of two communication channels.

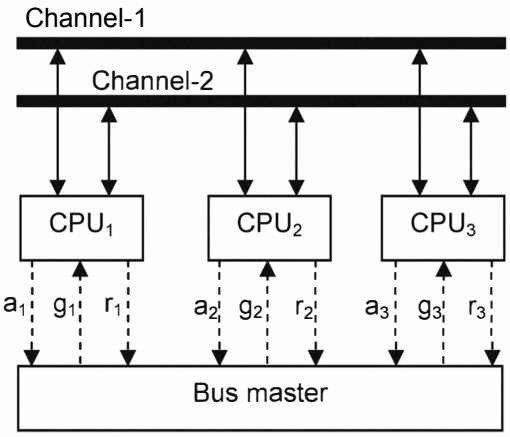

Figure 3. A multi-processor system (adapted from [4][10])

Obviously, communication channel is a bottle neck here, as there are three CPUs and each may compete for a channel when only two are available. In figure-3, 'a' is the signal from CPUs to bus master to 'acquire (request)' a channel, 'g' is the signal from bus master to CPUs about grant of a channel, and 'r' is the signal from CPUs to bus master about release of a channel.

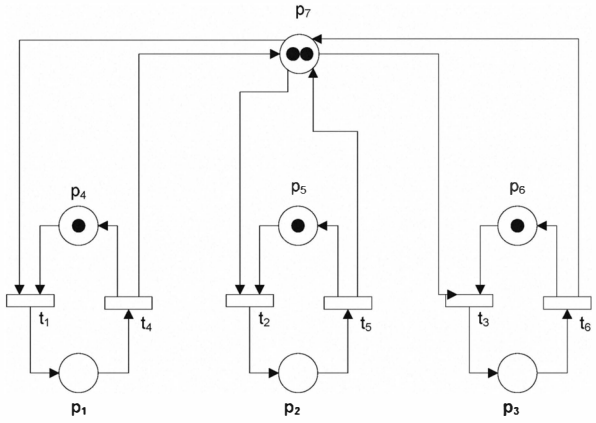

Figure 4. P/T Petri Net Model (adapted from [4])

A. P/T Petri Net Model

In this simple system, CPUs are the active elements (transitions) and the number of available channels, which is two, can be represented by two tokens in the system. The Petri net model is shown in figure 4.

Note that the Petri net structure has to satisfy two properties:

1) Conservation of tokens: total number of tokens representing resoteces has to be a constant at any time (equals to 2, in this example), and

2) Semafor: Use of a channel (resource) has to be guarded so that only one CPU can use it at a time.

To satisfy the first property, the place P_7 is included with two tokens. To satisfy the second property, three additional loops ('semafor loops' containing places P_4, P_5, and P_6) are included in the model.

B PIC Petri Net Model

In the P/T Petri net model shown in figure-4, the most interesting places are P_1, P_2, and P_3 as these places show whether a CPU is using a channel or not. Thus, these three places become part of the COM variables set. However, these three places are for monitoring only: the system dynamics cannot be influenced by manipulating these places (e.g. we cannot inject a token in P_1 and claim that CPU is occupying a channel). To control the dynamics of the system, we need three pair of additional places : Pa_1, Pr_1,..., Pa_3, Pr_3; see figure 5.

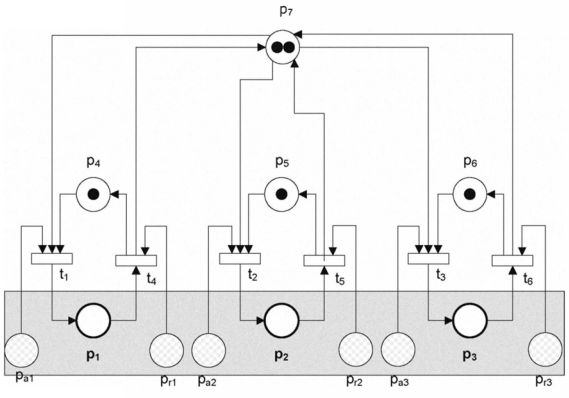

Figure 5, PIC Petri Net Model (adapted from [4])

These places Pa_1, Pr_1, ..., Pa_3, Pr_3 also become part of the COM variable set; this because, by manipulating these places we can control the system - for example, placing a token into the place Pa1 commands the system to allocate a channel (if available) to CPU_1; placing a token into place Pr_1 commands the system to release the channel occupied by CPU_1 and mark the channel as available.

Thus, the P/T model shown in figure-4 explains the behavior of the system, whereas the PIC model shown in figure-5 can be used to control the system.

Note: some designers may opt to include the place P_7 too in COM variables set, as P_7 always shows how many free channels are available at any instant of time.

C. Petri Net Model by GPenSIM Approach

GPenSIM supports representing resources as tokens as shown in the P/T model (figure-4) and PIC model (figure-5). However, if the modeler wants to create a compact model, the GPenSIM allows keeping resources away from the structure of the Petri net model. In this case, the functionality available in the software assures 1) the conservation of the number of resources throughout the simulation, and 2) 'mutual exclusion' in the resource use.

1) GPenSIM assures conservation of resources

Since the GPenSIM software assures conservation of the resources, there is no need to conserve the number of tokens in the model. Thus, all the structural elements that are added (place P_7 with the tokens, and the connections between P_7 with the rest if the system) just to ensure the conservation of tokens can now be deleted from the model; figure-6 shows that the place P_7 and the arcs that are connection the place P_7 to the rest of the system are now obsolete.

2) GPenSIM assures mutual exclusion in resource usage

Since the GPenSIM software inherently assures mutual exclusion in the usage of resources, there is no now need for semafor loops around resource usage. Hence, we can safely remove all the semafor loops in the model. Figure-7 shows that the places P_4, P_5 and P_6, and the arcs connecting these places to the rest of the system now becomes obsolete.

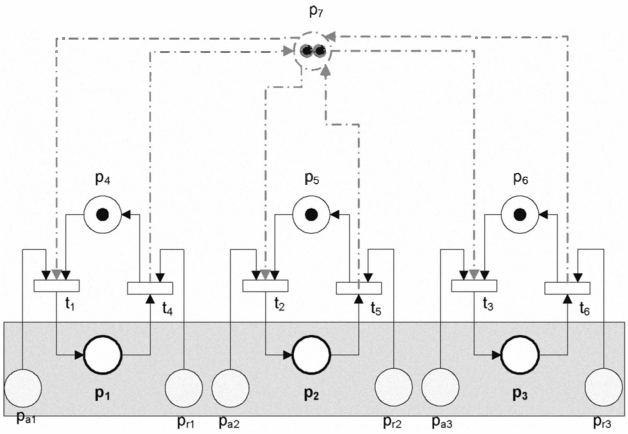

Figure 6 Structural elements for conservation of tokens are obsolete

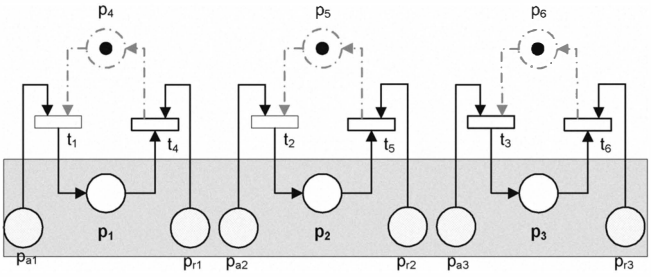

Figure 7. Structural elements for mutual exclusion are obsolete

D. Simulation Program for the Case Study

For brevity, the simulation program is not shown in this paper. Appendix shows some interesting code snippets. Interested reader is referred to the website [8] where complete code for the simulation program can be found.

v. DISCUSSION AND CONCLUSION

This paper presents GPenSIM's two approach for representing resources in Petri net models, namely hardwiring and soft-coding.

Figure-8 summarizes the advantages and disadvantages of the two approaches. The soft-coding approach definitely reduces the size of the Petri net: considering the application example, if there are *n* numbers of CPUs, then the model by soft-coding needs only *3n* places and *4n* arcs; whereas, the hardwiring approach needs *4n* places and *8n* arcs, in addition to the common place P_7. Though reduction in the connection between the elements in the physical system); this is a distinct advantage of using hardwiring approach. The model by soft-coding approach looks completely different from the actual physical system.

Soft-coding is not an entirely a new approach as various Petri nets based tools for DEDS simulations allow option to soft-code rather than hardwire; for example, CPN tool allow programming using a special language known as the ML language [5]. However, the alternative between hardwiring and soft-coding is very decisive in GPenSIM as programming in GPenSIM involves industry MATLAB programming, which is compact, easy, and efficient as it has a massive collection of in-number of elements in a model itself is an

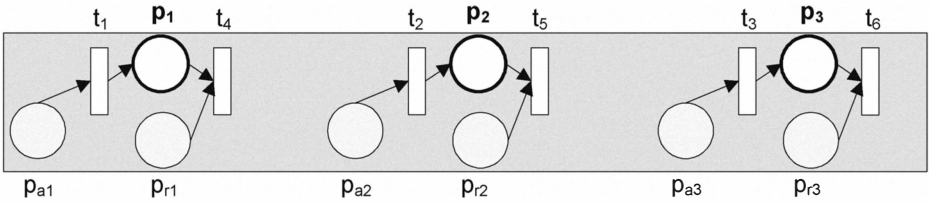

Figure 8a. Petri net model by soft-coding resources

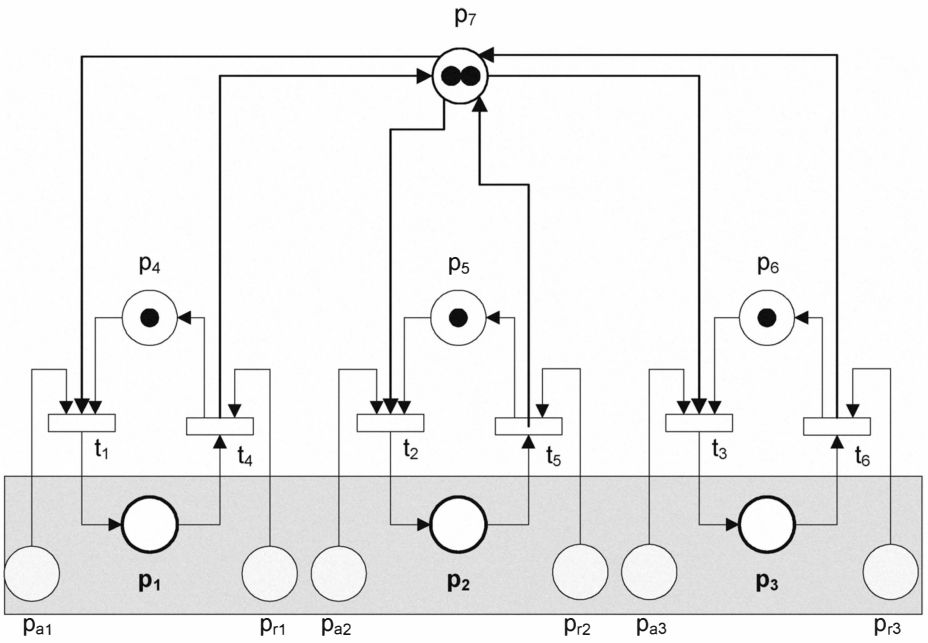

Figure 8b. Petri net model by hardwiring resources

Figure 8. Summarizing GPenSIM's two approach for representing resources in Petri net models

advantage, there will be additional benefits due to the size reduction such as ease of programming, ease of extending the model, debugging the model, etc.

However, the model by hardwiring approach explicitly resembles the physical system (or rather the topology of built functions for nearly all the functionalities needed in DEDS simulations, in addition to the core functions offered by GPenSIM.

This paper does not discuss the advantages and disadvantages of the two approaches on other aspects such as real-time applications; this is because the current version of GPenSIM (version 6) is not suitable for real-time applications.

VI. APPENDIX A: PROGRAMMING WITH GPENSIM

A. *Using Resources*

The resources are to be declared first in MSF. For example, if there three (human) resources named AI, Bob, and Chuck, then they are declared in MSF as follows:

```
dynamic.resources={'Al','Bob','Chuck'};
```

Reserving a resource can be done through the function 'resource request'. For example:

```
%T1 seeks specific resources, both 'Al'
%and 'Bob'
[acquired, PN] = resource_request(PN, 'T1',
{'Al', 'Bob'});

% T1 seeks any one resource
[acquired,PN]=resource_reuqest(PN, 'T1');
```

Releasing the resources: after firing, a transition has to release all the resources it is holding:

```
%release all resources (if any) held by 'T1'
 [released, PN] = resource_ release(PN, 'T1');
```

Function 'print_schedule' prints the resource usage; it printouts which transitions were using a specific resource and for how long, etc.

B. *Manipulating Priorities*

In discrete systems, we need to increase or decrease priority of an event or events, in order to give fair chance to the competing events. There are some basic facilities in GPenSIM to change priorities of transitions.

Declaration of initial priorities can be done in the main simulation file; initial priorities can be coded in the initial dynamic part:

```
% setting initial priority
% t3 has top priority (5), followed by t1 (3)
dyn.initial_priority = {'t1',3, 't3',5};
```

Increasing priority of a specific transition can be done using the function 'priority_increment', which will increase the value just by 1.

```
% increase priority of 't1' by 1
PN = priority_increment(PN, 't1');
```

Decreasing priority of a specific transition can be done using the function 'priority_decrement', which will reduce the value by 1.

```
% decrease priority of 't3' by 1
PN = priority_decrement(PN, 't3');
```

We can assign any priority to a transition using the function 'priority_assign':

```
% set priority of 't1' to 10
PN = priority_assign(PN, 't1', 10);
```

We can also get current priority of a transition using the function 'get_priority':

```
%get the priority of 't1'
prio_val=get priority(PN, 't1');
```

Finally, we can also compare priority of two transitions using the function 'priority_compare':

```
%compare priorities of 't1' and 't3'
HEL=priority_compare(PN, 't3', 't1');
```

If **t3** has higher priority than **t1**, then the returned value will be 1; if both have equal priority then a value of zero will be returned; otherwise, -1 will be returned.

REFERENCES

[1] Cassandras, G. and LaFortune, S. (1999) *Introduction to Discrete Event Systems*. Hague, Kluwer Academic Publications

[2] Davidrajuh, R. (2009) *Modeling and Simulation of Discrete Event Systems: A Hands-On Approach with GPenSIM*. Pub1isher: VDMVer1ag; ISBN: 978-3-639-19566-8

[3] GPenSIM (2010). Availab1e: http://www.davidrajuh.net/gpensim/

[4] Hruz, B. and Zhou. M. (2007) *Modening and Control of Discrete-event Dynamic Systems: with Petri Nets and other Tool.* Springer-Ver1ag, London

[5] Jensen, K. (1997) *Colored Petri nets*. Vol. I., II., III. Second edition. Springer, Berlin

[6] MATLAB (2009). Available: http://www.mathworks.com

[7] Petri net world (2009). Available: http://www.informatik.uni-hamburg.de/TGI PetriNets

[8] Resource Sharing Example (2011): Available: **http://www.davidrajuh.net/gpensim/resource-sharing-SO LI20 11**

[9] Silberschatz, A., Galvin, B., and Gagne, G. (2009) *Operating System Concepts*, 7th Edition, John Wiley & Sons, Inc, NY.

[10] Starke, P. (1990) *Analyse von Petri-netz-modellen*. B. G. Teubner, Stuttgart

[11] Wikipedia (2009) Available: **http://www.wikipedia.org**